Rapport technique

Apprentissage par renforcement : A-t-on écouté Joelle Pineau en 2018 ?

Un sondage des articles de 2018 à 2026 et des expériences préenregistrées, chaque nombre recalculé d'une seule commande

Warith Harchaoui
Deraison.ai · warithmetics@deraison.ai

21 novembre 2026

Code, données et protocoles : `github.com/warith-harchaoui/pineau-rl`
Article de blog : `deraison.ai/blog/ecouter-joelle-pineau`
Licence BSD 3-Clause pour le code

Deraison.ai

Résumé

La question. En 2018, à ICLR puis à NeurIPS, Joelle Pineau a montré pourquoi les résultats publiés en apprentissage par renforcement profond (RL) inspiraient peu confiance. Nous y relevons seize failles et demandons, pour chacune, si elle a été écoutée.

La méthode. Deux instruments servent, sous des protocoles écrits d'avance dont chaque écart est daté. Le sondage relit des articles avec la grille de Pineau (Q1 à Q8), sept critères tirés de sa conférence d'ICLR (X1 à X7) et une seconde grille (Y1 à Y12). Des agents de modèle de langue lisent chaque article entier et justifient chaque décision par une citation paginée. Le verdict compare 50 articles de RL classique de 2026 (hors modèles de langue) à notre lecture de 50 articles de 2018, avec les mêmes grilles et les mêmes lecteurs. Vingt articles par année lue entre 2019 et 2024 datent le changement. Les expériences refont celles de 2018 avec PPO (en PyTorch et en JAX), SAC et CrossQ sur quatre tâches MuJoCo, dix graines aléatoires par programme et par tâche, puis vingt. Un volet entraîne deux petits modèles de langue avec GRPO.

Les résultats du sondage. Notre lecture de 2018 s'écarte des chiffres de Pineau : 55 % des articles donnent tous les hyperparamètres de leur méthode, quand elle en comptait 90 %. Les verdicts comparent donc nos deux lectures. Dans le RL classique, quatre pratiques ont changé : le lien vers le code passe de 31 % à 74 % ($+43$ points, $[+23\,;\,+58]$), la sensibilité à un réglage montrée de 44 % à 80 %, le calcul chiffré de 27 % à 58 %, une incertitude rapportée de 63 % à 88 %. Cette dernière est fragile : sans treize articles lus en partie par un autre modèle de langue, elle échoue à la correction de Holm ($p_{\text{Holm}} = 0{,}07$). Les quatre montent lentement, sans saut après la liste de contrôle de NeurIPS de 2019. Le test statistique (de 6 % à 12 %), les références décrites avec tous leurs réglages (35 % contre 29 %), l'évaluation tenue à part (33 % puis 45 %) et la version des bancs d'essai (23 % puis 26 %) n'ont pas bougé de façon détectable. Pineau a été écoutée sur ce qui se publie à côté du résultat, beaucoup moins sur la manière de le juger.

Les résultats des expériences. Sur trente essais de PPO, deux groupes de cinq tirés au hasard trompent un test de Welch dans 1,2 % des cas, triés dans 31 %. Garder les trois meilleurs essais sur dix gonfle la moyenne de 33 %. Le script de Peter Henderson, qui ignore la dispersion entre graines aléatoires, explique très probablement le t de 9 de 2018 : sur nos essais de TRPO, il dépasse 9,09 dans 38 % des tirages. À budget égal, avec vingt graines aléatoires, CrossQ mène sur les quatre tâches. Une physique modifiée de 20 % envoie le meneur au dernier rang sur Hopper. Sur Qwen, avec cinq graines aléatoires, GRPO gagne $+14$ points, intervalle de -3 à $+34$, qui contient zéro.

Code et données. Dépôt `github.com/warith-harchaoui/pineau-rl` (BSD 3-Clause) : protocoles et lectures citées (`study/`), entraînements (`sim/`), statistiques (`src/pineau_rl/`). `make verify` régénère et vérifie chaque résultat.

Table des matières

Résumé **i**

1 Introduction **1**
1.1 Deux conférences en 2018 . . . 1
1.2 La question . . . 2
1.3 Deux instruments . . . 2
1.4 Contributions . . . 2
1.5 Guide de lecture . . . 3

2 Contexte : l'apprentissage par renforcement et les seize failles **4**
2.1 Le cadre formel . . . 4
2.2 Les quatre programmes et TRPO . . . 5
2.3 Les tâches MuJoCo . . . 6
2.4 Ce que change une graine aléatoire . . . 6
2.5 Évaluer un agent . . . 7
2.6 Les seize failles montrées en 2018 . . . 8
2.6.1 Le matériel : ce qu'il faut pour refaire . . . 9
2.6.2 Le réglage : hyperparamètres et références . . . 10
2.6.3 Les graines aléatoires et la statistique . . . 10
2.6.4 La portée des conclusions . . . 11
2.7 Travaux voisins . . . 12
2.8 Le vocabulaire des modèles de langue . . . 13

3 Le sondage : méthode **14**
3.1 Le protocole et son calendrier . . . 14
3.2 Populations et tirages . . . 17
3.3 Les grilles de lecture . . . 18
3.4 Les lecteurs . . . 22
3.5 Les doubles lectures à l'aveugle . . . 23
3.6 Le pilote et l'harmonisation . . . 24
3.7 Les contrôles des citations et des décisions négatives . . . 25
3.8 La calibration sur 2018 et la règle de verdict . . . 25
3.9 Les statistiques . . . 26
3.10 Les analyses de robustesse . . . 27
3.11 Les écarts au protocole . . . 29

4 Le sondage : résultats **30**
4.1 Les échantillons obtenus . . . 30
4.2 La comparaison principale, à grille égale . . . 31
4.3 Relecture faille par faille . . . 33
4.4 Ce qui résiste aux analyses de robustesse . . . 36
4.5 Notre lecture de 2018 contre les chiffres de Pineau . . . 38
4.6 Le premier tirage et les modèles de langue . . . 39
4.7 La trajectoire de 2018 à 2026 . . . 41
4.8 L'accord entre lecteurs . . . 45

4.9 Ce que les articles déclarent . . . 48
4.10 La sensibilité au filtre de titre . . . 48
4.11 Synthèse . . . 49

5 Les expériences : protocole et méthodes 50
5.1 Un protocole écrit avant le premier entraînement . . . 50
5.2 Les outils et leurs versions . . . 51
5.3 Les tâches . . . 51
5.4 Les programmes . . . 52
5.5 Les budgets . . . 54
5.6 Les graines aléatoires . . . 55
5.7 L'évaluation, tenue à part . . . 56
5.8 Le format des essais . . . 57
5.9 Les statistiques . . . 58
5.9.1 La moyenne interquartile . . . 58
5.9.2 L'intervalle bootstrap stratifié . . . 58
5.9.3 La probabilité d'amélioration . . . 58
5.9.4 La règle du vainqueur . . . 59
5.9.5 Les tests t de Student et de Welch . . . 59
5.9.6 Trois lectures du t moyen de 2018 . . . 59
5.9.7 La statistique du script de Henderson . . . 60
5.9.8 Corrections pour comparaisons multiples . . . 60
5.9.9 Autres tests . . . 60
5.9.10 Des intervalles reproductibles . . . 60
5.10 Les expériences, une à une . . . 61
5.10.1 E1 : même code, autre graine aléatoire . . . 63
5.10.2 B1 : l'énigme du t de 9 . . . 63
5.10.3 E2 : le classement change-t-il de tâche en tâche ? . . . 64
5.10.4 E3 : deux programmes du même algorithme . . . 65
5.10.5 E4 : les trois meilleurs sur dix . . . 65
5.10.6 E5 : combien de graines aléatoires ? . . . 65
5.10.7 E6 : la référence réglée . . . 65
5.10.8 E7 : la physique modifiée . . . 66
5.10.9 B3 : la mesure change-t-elle le classement ? . . . 66
5.10.10 La consolidation : vingt graines aléatoires et budget égal . . . 67
5.11 La file d'attente et le calcul . . . 67
5.12 Ce qui a été écrit avant les résultats, ce qui l'a été après . . . 70

6 Les expériences : résultats 72
6.1 E1 : même code, autre graine aléatoire . . . 72
6.2 B1 : l'énigme du t de 9 . . . 75
6.3 E2 : le programme en tête change de tâche en tâche . . . 78
6.4 E3 : deux programmes du même algorithme . . . 84
6.5 E4 : les trois meilleurs sur dix . . . 85
6.6 E5 : combien de graines aléatoires ? . . . 86
6.7 E6 : une méthode de référence un peu trop belle . . . 87
6.8 E7 : ce que coûte une physique modifiée . . . 90
6.9 La physique modifiée par degrés . . . 94
6.10 B3 : la mesure change-t-elle le classement ? . . . 96
6.11 Dix ou vingt graines aléatoires, budget égal : ce qui tient et ce qui change 98
6.12 Ce que ces expériences ne disent pas . . . 100

7 Le volet des modèles de langue **101**
7.1 Le vocabulaire . . . 101
7.2 Pourquoi ce volet . . . 101
7.3 L'algorithme GRPO, tel que nous l'avons écrit . . . 102
7.3.1 La tâche et la récompense . . . 102
7.3.2 Un pas d'entraînement . . . 102
7.3.3 Les garde-fous contre un calcul corrompu . . . 103
7.3.4 La mesure . . . 104
7.4 Le protocole et ses essais . . . 104
7.5 Les résultats . . . 105
7.5.1 La récompense tirée au sort . . . 105
7.5.2 La vraie récompense, en deux temps . . . 107
7.5.3 Le matériel et la graine aléatoire, mêlés . . . 109
7.6 Le piège de la coupure . . . 109
7.6.1 Le diagnostic . . . 109
7.6.2 Ce que montrent les vingt essais . . . 111
7.7 Ce que ce volet établit et ce qu'il n'établit pas . . . 111

8 Logiciel et reproductibilité **112**
8.1 L'organisation du dépôt . . . 112
8.2 Le paquet `pineau_rl` . . . 112
8.2.1 `pineau_rl.stats` . . . 112
8.2.2 `pineau_rl.bootstrap` . . . 116
8.2.3 `pineau_rl.runs` . . . 116
8.2.4 `pineau_rl.grid` . . . 116
8.2.5 `pineau_rl.fmt` . . . 117
8.3 Les formats de fichiers . . . 117
8.4 Chaque nombre vérifié contre son fichier . . . 118
8.5 Les cibles du `Makefile` . . . 118
8.6 L'intégration continue . . . 119
8.7 Les tests . . . 120
8.8 Les environnements figés . . . 121
8.9 Les politiques entraînées . . . 121
8.10 Le matériel et le calcul . . . 122
8.11 Ce qui n'est pas dans le dépôt . . . 122
8.12 Licence et citation . . . 122

9 Les écarts aux protocoles **123**
9.1 Le principe . . . 123
9.2 Ce qui repose sur notre parole . . . 123
9.3 Les registres des tentatives . . . 124
9.4 La liste des écarts . . . 124
9.5 Les écarts du sondage . . . 127
9.6 Les écarts des simulations . . . 130
9.7 Les écarts du volet GRPO . . . 131
9.8 Ce que ces écarts disent du travail . . . 132

10 Limites et menaces sur la validité **133**
10.1 Validité interne . . . 133
10.1.1 Les lecteurs du sondage sont des agents de modèle de langue . . . 133
10.1.2 Les décisions prises après les résultats . . . 134

10.1.3 Les essais des simulations . . . 134
10.1.4 Le matériel et la graine aléatoire du volet GRPO . . . 134
10.2 Validité externe . . . 134
10.2.1 La population des articles . . . 135
10.2.2 Les simulations . . . 135
10.2.3 Les budgets . . . 135
10.2.4 Une seule plateforme . . . 136
10.2.5 L'échelle du volet GRPO . . . 136
10.3 Validité de construit . . . 136
10.4 Validité de la conclusion statistique . . . 137
10.5 Ce que testerait un nouvel entraînement sur une autre machine . . . 137

11 Conclusion 139
11.1 La réponse . . . 139
11.2 Les nombres principaux . . . 139
11.3 Recommandations . . . 140
11.3.1 Pour les auteurs . . . 140
11.3.2 Pour les relecteurs . . . 141
11.3.3 Pour les conférences . . . 141
11.4 Travaux futurs . . . 142

A La grille de lecture complète 143
A.1 Ce que fait le lecteur . . . 143
A.2 La grille de Pineau (Q1 à Q8) . . . 143
A.3 Nos ajouts de la grille v1 (X1 à X7) . . . 144
A.4 La grille v2 (Y1 à Y12) . . . 145
A.5 L'harmonisation . . . 147

B Tableaux complémentaires 148
B.1 Le RL de grands modèles de langue dans le sondage . . . 148
B.2 Index des tableaux générés . . . 148
B.3 Les fichiers de résultats . . . 149

C Les articles lus 151
C.1 Les articles de 2026 . . . 151
C.2 Les articles de 2018 . . . 155

D Les environnements logiciels 158
D.1 La plateforme . . . 158
D.2 Les quatre environnements . . . 158
D.3 Les pièges de l'installation . . . 158
D.4 La chaîne des documents . . . 160

Bibliographie 161

CHAPITRE 1

Introduction

1.1 Deux conférences en 2018

Vancouver, puis Montréal. Au printemps 2018, Joelle Pineau, professeure à l'université McGill de Montréal et directrice du laboratoire montréalais de recherche en intelligence artificielle de Facebook, donne une conférence invitée à ICLR (*International Conference on Learning Representations*), à Vancouver (Pineau 2018a). La conférence s'intitule « Reproductibilité, réutilisabilité et robustesse en apprentissage par renforcement profond ». En décembre, Pineau reprend et augmente son propos à NeurIPS (*Neural Information Processing Systems*), à Montréal, comme conférence Posner : « Apprentissage par renforcement reproductible, réutilisable et robuste » (Pineau 2018b). ICLR, NeurIPS et ICML (*International Conference on Machine Learning*) sont les trois congrès principaux de l'apprentissage automatique.

La question de départ. Si une équipe publie un résultat, une autre équipe qui suit la recette obtient-elle le même ? Pineau part d'un sondage de la revue *Nature* : sur 1 576 chercheurs de toutes disciplines, plus de 70 % disaient avoir déjà échoué à reproduire l'expérience d'un collègue, 52 % voyaient une crise « significative » de la reproductibilité et 38 % une crise « légère » (Baker 2016). Lors d'un atelier en décembre 2017, elle a posé la même question à vingt spécialistes de l'apprentissage automatique : 80 % y voient une crise, légère ou significative (Pineau 2018a). Elle cite ensuite Buckheit et Donoho (1995) : un article scientifique sur le calcul est seulement la publicité de la recherche. La recherche elle-même est l'environnement logiciel complet et l'ensemble des instructions qui ont produit les figures.

Un domaine qui grandit, des erreurs qui coûtent. Pineau rappelle que le RL a cité pendant près de vingt ans un seul succès marquant, le programme de backgammon TD-Gammon (Tesauro 1995), avant AlphaGo (Silver et al. 2016). Vers 2000, un atelier sur le RL réunissait trente-cinq personnes. En 2016, elle compte environ treize mille articles de RL publiés dans l'année (Pineau 2018a). Elle ajoute que le RL s'approche de problèmes qui comptent : la conception de traitements médicaux, la gestion des ressources naturelles, des réseaux d'énergie et des systèmes financiers. Le doute venait aussi des praticiens. En février 2018, Alex Irpan, de Google Brain, écrivait qu'il répondait non à qui lui demandait si le RL pouvait résoudre son problème. Il pensait avoir raison au moins 70 % du temps (Irpan 2018). Pineau projette l'adresse de ce billet à NeurIPS.

Les seize failles. Pineau montre ensuite, figure après figure, ce qui manque au RL pour mériter la confiance. Les figures viennent pour la plupart de l'article de son équipe signé Peter Henderson et ses coauteurs (Henderson et al. 2018). Nous y avons relevé seize failles, numérotées F1 à F16 et rangées en quatre familles : le matériel nécessaire pour refaire une expérience (code, plan, logiciels), le réglage des hyperparamètres et des références, les graines aléatoires et la statistique, la portée des conclusions. La plus célèbre est la faille F11 : dix entraînements du même programme, répartis en deux

groupes de cinq, donnent deux courbes qu'un test déclare différentes, avec $t = -9{,}09$ et $p = 0{,}0016$. Le chapitre 2 décrit les seize failles une à une (tableau 2.2). Pineau ne s'arrête pas au constat. Elle propose une liste de contrôle de reproductibilité, que NeurIPS adopte en 2019 (Pineau et al. 2021). Elle présente aussi le défi de reproduction d'ICLR 2018, où n'importe qui pouvait tenter de refaire un article soumis.

1.2 La question

Huit ans ont passé. Le RL a changé de taille et de terrain : en 2025 et 2026, plus de la moitié des articles de RL de notre premier tirage appliquent ce qu'ils appellent RL à de grands modèles de langue. La question de ce rapport est simple : *a-t-on écouté Joelle Pineau ?* Pour chacune des seize failles, nous demandons si les articles publiés entre 2018 et 2026 la montrent encore. Nous mesurons aussi ce que coûte la faille quand on refait les expériences de 2018 avec les algorithmes de 2026. Nous prenons sa liste comme une grille et nous l'appliquons aux articles publiés, puis à nos propres expériences, avec la même sévérité.

1.3 Deux instruments

Le sondage. Le premier instrument relit des articles avec la grille de lecture de Pineau (critères Q1 à Q8), complétée par sept critères tirés de sa conférence d'ICLR (X1 à X7) et par une seconde grille de douze critères pour les failles qu'elle laissait sans mesure (Y1 à Y12). Nous avons lu un premier tirage de 50 articles de RL de 2025 et 2026, parmi les 1 252 articles de RL d'ICLR 2026, d'ICML 2026 et de NeurIPS 2025, puis 50 articles de RL classique de 2026 (robotique, jeux, contrôle, hors modèles de langue), 50 articles de 2018 et 20 articles par année en 2019, 2020, 2022 et 2024. Des agents de modèle de langue lisent les articles. Chaque décision repose sur une citation exacte et sa page. Le verdict compare le RL classique de 2026 à notre propre lecture de 2018, avec les mêmes grilles et les mêmes lecteurs. Les chapitres 3 et 4 décrivent la méthode et les résultats.

Les expériences. Le second instrument refait les expériences de 2018 sous un protocole écrit avant le premier entraînement : quatre programmes (PPO en PyTorch et en JAX, SAC, CrossQ) sur quatre tâches MuJoCo, avec dix graines aléatoires par programme et par tâche, puis vingt. S'y ajoutent trente essais de PPO et trente de TRPO sur HalfCheetah, une évaluation sur des graines aléatoires jamais vues à l'entraînement et une autre sous une physique modifiée. Un volet illustre les mêmes failles sur des modèles de langue entraînés avec GRPO. Les chapitres 5, 6 et 7 les décrivent.

1.4 Contributions

1. **Une mesure à grille égale.** La grille de Pineau, réappliquée aux articles de 2026 et calibrée sur notre propre lecture de 50 articles de 2018. À grille égale, quatre pratiques ont changé dans le RL classique : le lien vers le code (31 % puis 74 %), la sensibilité à un réglage montrée (44 % puis 80 %), le calcul chiffré (27 % puis 58 %), une incertitude rapportée (63 % puis 88 %). D'autres n'ont pas bougé de façon détectable, dont le test statistique (6 % puis 12 %).

2. **Une seconde grille préenregistrée.** Douze critères (Y1 à Y12), écrits avec leurs règles de décision avant tout codage, qui couvrent la mesure de performance, les tâches, les implémentations, les versions, la généralisation, la sensibilité et les outils statistiques récents.
3. **Une trajectoire.** Quatre années lues entre 2018 et 2026 : les quatre pratiques montent lentement, sans saut juste après la liste de contrôle de 2019.
4. **Une batterie de contrôles.** Double lecture à l'aveugle, audit des citations, recherche mécanique contre les décisions négatives, provenance publiée de chaque lecteur, coupes par provenance, conférence et lecteur, sensibilité au modèle des lecteurs et au filtre de titre, confrontation aux listes de contrôle remplies par les auteurs.
5. **Les expériences de 2018, refaites.** Deux groupes de cinq essais tirés au hasard ne trompent un test de Welch que dans 1,2 % des cas. Triés, ils le trompent dans 31 %. Garder les trois meilleurs essais sur dix gonfle la moyenne de 33 %. Le meilleur programme change avec la tâche, sauf à budget égal avec vingt graines aléatoires, où CrossQ mène sur les quatre tâches. Une physique modifiée de 20 % fait passer le premier au dernier rang sur Hopper.
6. **Une explication du t de 9.** Le script publié par Peter Henderson en 2017 mesure la dispersion entre épisodes et ignore celle entre graines aléatoires. Appliqué à nos essais de TRPO, il dépasse 9,09 dans 38 % des tirages au hasard.
7. **Un pont vers les modèles de langue.** Un GRPO écrit par nous, sur deux petits modèles ouverts, avec la vraie récompense (celle qui vérifie la réponse) et une récompense tirée à pile ou face. Avec cinq graines aléatoires, le gain moyen de Qwen avec la vraie récompense vaut $+14$ points, avec un intervalle de -3 à $+34$ qui contient zéro.
8. **Un dépôt qui se vérifie.** Les protocoles datés, les lectures avec leurs citations, les résultats des entraînements, les scripts d'analyse et des contrôles qui confrontent chaque nombre du document argumenté (`docs/conclusions.md`) et de l'article de blog aux fichiers de résultats (chapitre 8).

1.5 Guide de lecture

Le chapitre 2 pose le cadre formel du RL, décrit les programmes et les tâches, dresse l'inventaire des seize failles et situe le travail parmi les travaux voisins. Les chapitres 3 et 4 donnent la méthode et les résultats du sondage. Les chapitres 5 et 6 font de même pour les expériences sur MuJoCo. Le chapitre 7 décrit le volet GRPO. Le chapitre 8 décrit le logiciel, de l'entraînement aux documents. Le chapitre 9 liste tous les écarts aux trois protocoles. Le chapitre 10 rassemble les limites. Le chapitre 11 reprend chaque faille avec son verdict. Les annexes donnent les règles complètes de la grille (annexe A), des tableaux supplémentaires (annexe B), la liste des articles lus (annexe C) et les environnements logiciels (annexe D).

Un lecteur pressé lira le résumé, le tableau 2.2, la section 4.2 et le chapitre 11. Un lecteur qui veut refaire un calcul trouvera, dans chaque légende, le fichier de résultats qui le contient.

CHAPITRE 2

Contexte : l'apprentissage par renforcement et les seize failles

Ce chapitre pose les notions dont dépend la suite du rapport. La section 2.1 donne le cadre formel de l'apprentissage par renforcement. La section 2.2 décrit les quatre programmes de nos expériences et l'algorithme TRPO, la section 2.3 les tâches simulées. Les sections 2.4 et 2.5 précisent ce qu'une graine aléatoire fixe et comment on évalue un agent. La section 2.6 dresse l'inventaire des seize failles que Joelle Pineau a montrées en 2018. La section 2.7 situe notre travail parmi les travaux voisins. La section 2.8 fixe le vocabulaire employé pour les modèles de langue.

2.1 Le cadre formel

Le processus de décision markovien. L'apprentissage par renforcement (RL, pour *reinforcement learning*) étudie un programme, l'agent, qui agit de façon répétée dans un environnement. Le cadre standard est le processus de décision markovien (Puterman 1994 ; Sutton et Barto 2018), un quintuplet

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma), \tag{2.1}$$

où $\mathcal{S}$ est l'ensemble des états, $\mathcal{A}$ l'ensemble des actions, $P(s' \mid s, a)$ la probabilité de passer à l'état s' après l'action a dans l'état s, $r(s,a)$ la récompense reçue et $\gamma \in [0,1)$ le facteur d'actualisation, qui pèse les récompenses futures. Une distribution ρ_0 tire l'état initial. Dans les tâches de ce rapport, $\mathcal{S}$ et $\mathcal{A}$ sont des sous-ensembles de $\mathbb{R}^d$: l'état décrit les positions et les vitesses des articulations d'un corps simulé, l'action les couples appliqués aux moteurs.

Politique, épisode et retour. Une politique $\pi_\theta(a \mid s)$ donne la loi de l'action dans chaque état. En RL profond, elle est un réseau de neurones de paramètres θ. Un épisode est une suite $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_T)$ obtenue en tirant $s_0 \sim \rho_0$, puis $a_t \sim \pi_\theta(\cdot \mid s_t)$ et $s_{t+1} \sim P(\cdot \mid s_t, a_t)$, jusqu'à un état terminal ou jusqu'à une durée maximale. Le retour actualisé d'un épisode et l'objectif de l'apprentissage s'écrivent

$$G(\tau) = \sum_{t=0}^{T-1} \gamma^t \, r(s_t, a_t), \qquad J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\big[G(\tau)\big]. \tag{2.2}$$

L'apprentissage cherche θ qui maximise J. Le score rapporté dans les articles et dans nos expériences est presque toujours le retour non actualisé $\sum_t r(s_t, a_t)$ d'un épisode, en moyenne sur plusieurs épisodes d'évaluation.

Fonctions de valeur. La valeur d'un état et la valeur d'une action sous la politique π sont

$$V^\pi(s) = \mathbb{E}_\pi\Big[\sum_{k \geq 0} \gamma^k r_{t+k} \;\Big|\; s_t = s\Big], \qquad Q^\pi(s,a) = \mathbb{E}_\pi\Big[\sum_{k \geq 0} \gamma^k r_{t+k} \;\Big|\; s_t = s, a_t = a\Big]. \tag{2.3}$$

L'avantage $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$ mesure ce qu'une action rapporte de plus que la moyenne des actions de la politique dans cet état. Un algorithme acteur-critique apprend à la fois une politique (l'acteur) et une estimation d'une fonction de valeur (le critique), qui sert à réduire la variance du signal d'apprentissage de l'acteur.

Gradient de politique. Pour une politique dérivable en θ, le théorème du gradient de politique (Sutton et Barto 2018) donne

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\, A^{\pi_\theta}(s_t, a_t)\right]. \tag{2.4}$$

Les méthodes dites *on-policy*, comme TRPO et PPO, estiment cette espérance avec des épisodes tirés de la politique courante, puis jettent ces épisodes après la mise à jour. Les méthodes *off-policy*, comme SAC et CrossQ, gardent les transitions passées dans une mémoire de rejeu et apprennent un critique Q à partir de ces transitions. Elles réutilisent chaque transition plusieurs fois. Pour un même nombre de pas d'environnement, elles apprennent en général plus vite, au prix d'un calcul plus lourd par pas.

Hyperparamètres. Les paramètres θ s'apprennent. D'autres réglages restent fixés par l'expérimentateur : le taux d'apprentissage, γ, la taille des réseaux, la taille des lots, le nombre de pas collectés avant chaque mise à jour. Ce sont les hyperparamètres. Les failles F4 à F7 de la section 2.6 les concernent.

2.2 Les quatre programmes et TRPO

Nos expériences emploient quatre programmes, c'est-à-dire quatre implémentations précises d'un algorithme, dans une bibliothèque et une version données. Le mot « algorithme » désigne la méthode décrite dans un article. Le mot « programme » désigne le code qui l'exécute. Deux programmes du même algorithme donnent parfois des résultats différents : c'est la faille F14. Les versions figées de chaque bibliothèque sont données au tableau D.2. Le chapitre 5 détaille les budgets et l'évaluation.

PPO, en PyTorch et en JAX. PPO (*Proximal Policy Optimization*) (Schulman et al. 2017) est un algorithme acteur-critique *on-policy*. Avec le rapport de probabilités $\rho_t(\theta) = \pi_\theta(a_t \mid s_t)/\pi_{\theta_{\text{old}}}(a_t \mid s_t)$ et un estimateur $\hat{A}_t$ de l'avantage, il maximise l'objectif tronqué

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t\Big[\min\big(\rho_t(\theta)\,\hat{A}_t,\ \text{clip}\big(\rho_t(\theta), 1-\epsilon, 1+\epsilon\big)\,\hat{A}_t\big)\Big], \tag{2.5}$$

auquel s'ajoutent une perte sur le critique et un bonus d'entropie. La troncature supprime l'intérêt, pour une mise à jour, d'éloigner trop la nouvelle politique de l'ancienne. Le programme PPO PyTorch est celui de Stable-Baselines3 (Raffin et al. 2021). Le programme PPO JAX est celui de sbx, la réécriture de Stable-Baselines en JAX (Bradbury et al. 2018 ; Raffin 2023). Les deux programmes partent des mêmes valeurs par défaut autant que les deux bibliothèques le permettent. Ils collectent l'expérience par paquets de 2 048 pas. PPO était la référence de 2018. Le protocole le retient parce qu'il reste une référence courante en 2026 et que GRPO, l'algorithme de notre volet sur les modèles de langue (chapitre 7), en dérive.

SAC. SAC (*Soft Actor-Critic*) (Haarnoja et al. 2018) est un algorithme acteur-critique *off-policy*. Il maximise le retour augmenté de l'entropie de la politique,

$$J_{\mathrm{SAC}}(\pi) = \sum_{t} \mathbb{E}_{(s_t,a_t)\sim\pi}\big[r(s_t,a_t) + \alpha\,\mathcal{H}\big(\pi(\cdot \mid s_t)\big)\big], \tag{2.6}$$

où le coefficient α règle l'exploration et s'ajuste pendant l'entraînement. Deux critiques Q appris en parallèle limitent la surestimation des valeurs. Des copies retardées de ces critiques, les réseaux cibles, stabilisent les cibles d'apprentissage. Le programme SAC de nos expériences est celui de sbx. SAC sert de référence *off-policy* courante en contrôle continu.

CrossQ. CrossQ (Bhatt et al. 2024) descend de SAC. Il retire les réseaux cibles et ajoute une normalisation par lots aux critiques. Les paires (s, a) et (s', a') passent ensemble dans le critique, pour que la normalisation voie les deux distributions. Ses auteurs annoncent une efficacité en données au moins égale à celle de méthodes beaucoup plus coûteuses en calcul. Le programme CrossQ de nos expériences est celui de sbx.

TRPO. TRPO (*Trust Region Policy Optimization*) (Schulman et al. 2015) est l'algorithme de la figure la plus célèbre de 2018 (faille F11). Il maximise un objectif de substitution sous une contrainte de divergence de Kullback-Leibler,

$$\max_{\theta}\ \hat{\mathbb{E}}_t\big[\rho_t(\theta)\,\hat{A}_t\big] \quad \text{sous} \quad \hat{\mathbb{E}}_t\big[\mathrm{KL}\big(\pi_{\theta_{\mathrm{old}}}(\cdot \mid s_t)\,\big\|\,\pi_{\theta}(\cdot \mid s_t)\big)\big] \le \delta, \tag{2.7}$$

résolue par un gradient conjugué suivi d'une recherche linéaire. PPO en est une simplification. Nous l'employons seulement dans l'expérience B1, avec le programme de sb3-contrib 2.9.2 et ses valeurs par défaut, sans aucun réglage (chapitre 5).

2.3 Les tâches MuJoCo

Henderson et ses coauteurs comparaient leurs algorithmes sur des tâches du simulateur MuJoCo (*Multi-Joint dynamics with Contact*) (Henderson et al. 2018 ; Todorov et al. 2012). Nous prenons les mêmes tâches dans leur version de 2026, celle de Gymnasium (Towers et al. 2024). Le tableau 2.1 donne les dimensions des espaces d'état et d'action, lues dans l'environnement figé de nos expériences. Chaque épisode dure au plus 1 000 pas. Hopper et Walker2d s'arrêtent plus tôt quand le corps tombe.

Pineau jugeait ces bancs d'essai « ridiculement simples » : des états décrits par peu de nombres, des transitions déterministes, un monde qui pèse moins de 100 kilo-octets (Pineau 2018b). Cette critique fonde la faille F16. Nous gardons ces tâches, puisque le but est de refaire les expériences de 2018 telles qu'elles furent faites.

2.4 Ce que change une graine aléatoire

Définition. Un entraînement tire beaucoup de nombres au hasard. Un ordinateur fabrique ce hasard à partir d'un entier de départ, la graine aléatoire (*random seed*). Dans nos programmes, la graine aléatoire est passée à la bibliothèque (`seed=SEED` dans `sim/run.py`). La bibliothèque en dérive l'initialisation des poids des réseaux, le tirage des actions d'exploration, l'ordre des mini-lots et la suite des états initiaux de l'environnement d'entraînement. Dans ce rapport, un *essai* désigne un entraînement complet avec une graine aléatoire donnée.

TABLE 2.1. Les quatre tâches de nos expériences, dans Gymnasium 1.1.1 et MuJoCo. Les dimensions et la durée maximale d'un épisode sont lues dans l'environnement de nos expériences (`sim/env`).

Tâche	état	action	Corps simulé et récompense
HalfCheetah-v5	17	6	guépard articulé en deux dimensions. Récompense : vitesse vers l'avant, moins un coût des commandes
Hopper-v5	11	3	sauteur à une jambe. Récompense : vitesse vers l'avant, bonus tant que le corps reste debout
Walker2d-v5	17	6	marcheur à deux jambes. Récompense : mêmes termes que Hopper
Swimmer-v5	8	2	nageur à trois segments dans un fluide visqueux. Récompense : vitesse vers l'avant

Ce qui reste fixe. Même programme, même graine aléatoire, mêmes versions des bibliothèques et même machine : même résultat. Nos journaux le vérifient pour PPO PyTorch et PPO JAX : un essai arrêté puis relancé retrouve à l'identique ses points d'évaluation (`sim/SIM_PROTOCOL.md`, écart 12 (e)). Changer la graine aléatoire change tous ces tirages. Le score final d'un même programme varie alors beaucoup. Sur nos trente essais de PPO PyTorch sur HalfCheetah, l'écart type vaut 34 % de la moyenne (chapitre 6).

L'unité statistique est l'essai. La graine aléatoire est une source de variance au même titre que le tirage d'un échantillon dans une enquête. Une comparaison entre deux programmes repose donc sur plusieurs essais indépendants par programme, avec une incertitude calculée sur ces essais, sinon elle mesure le hasard du tirage. L'épisode ou le point d'une courbe ne compte pas pour une observation indépendante, puisqu'il vient d'un seul essai. La faille F11 et l'expérience B1 montrent ce qui arrive quand on prend l'épisode pour l'unité (chapitre 6).

2.5 Évaluer un agent

Le score d'un essai. Évaluer un agent revient à jouer des épisodes avec sa politique, sans apprentissage, puis à faire la moyenne de leurs retours. Trois choix restent à faire. La faille F8 est l'article qui ne les dit pas.

- Le point de mesure : le score à la fin de l'entraînement, le meilleur score atteint le long de la courbe, ou la moyenne de la courbe, qui approche l'aire sous la courbe divisée par sa longueur.
- Le mode de la politique : stochastique comme à l'entraînement, ou déterministe (l'action la plus probable).
- L'environnement : celui de l'entraînement, ou des conditions tenues à part, comme d'autres graines d'environnement ou une physique modifiée (faille F15).

Nos programmes sont évalués tous les 50 000 pas sur 10 épisodes, en mode déterministe, avec des graines d'environnement $10\,000 + i$, où i numérote les épisodes d'évaluation, qui ne servent jamais à l'entraînement (chapitre 5).

Agréger les essais. Avec n essais de scores $x_{(1)} \le \cdots \le x_{(n)}$, la moyenne et la médiane réagissent différemment aux essais extrêmes. Agarwal et al. (2021) recommandent la moyenne interquartile (IQM, *interquartile mean*), qui écarte le quart le plus bas et le quart le plus haut des essais puis fait la moyenne du reste :

$$\mathrm{IQM}(x) = \frac{1}{n-2k} \sum_{i=k+1}^{n-k} x_{(i)}, \qquad k = \lfloor n/4 \rfloor. \tag{2.8}$$

C'est la moyenne tronquée à 25 % de `scipy.stats.trim_mean`, que la bibliothèque rliable appelle : avec dix essais, l'IQM est la moyenne des six essais du milieu. Son intervalle de confiance se calcule par un bootstrap stratifié : on rééchantillonne les essais avec remise au sein de chaque tâche, on recalcule l'IQM, puis on prend les quantiles à 2,5 % et 97,5 % des 2 000 répliques (`src/pineau_rl/bootstrap.py`).

Comparer deux programmes. La probabilité d'amélioration de X sur Y est la probabilité qu'un essai de X batte un essai de Y tiré indépendamment :

$$P(X > Y) = \frac{1}{n_X n_Y} \sum_{i=1}^{n_X} \sum_{j=1}^{n_Y} \Big(\mathbb{1}[x_i > y_j] + \tfrac{1}{2} \mathbb{1}[x_i = y_j] \Big). \tag{2.9}$$

Cette probabilité vaut la statistique U de Mann-Whitney divisée par $n_X n_Y$. La valeur 0,5 signale deux programmes indiscernables. Nos expériences déclarent un écart établi quand l'intervalle bootstrap à 95 % de $P(X > Y)$ exclut 0,5. Le test t de Welch (Welch 1947), qui compare deux moyennes sans supposer les variances égales, sert seulement dans les expériences E1 et B1, puisque c'est lui que la figure de 2018 employait.

2.6 Les seize failles montrées en 2018

Dans ses deux conférences de 2018, à ICLR puis à NeurIPS (Pineau 2018a,b), Pineau montre chaque faille par un exemple, souvent par une figure tirée de l'article de son équipe signé Peter Henderson et ses coauteurs (Henderson et al. 2018). Nous avons relevé seize failles, dans l'ordre des deux conférences, puis nous les avons rangées en quatre familles. Le tableau 2.2 les liste avec la diapositive ou la page qui les montre et l'instrument de ce rapport qui les mesure. Le numéro F1 à F16 sert seulement à retrouver une faille dans les tableaux. L'annexe A donne les règles de décision de chaque critère de lecture.

Table 2.2. Les seize failles que Joelle Pineau a montrées en 2018 dans l'apprentissage par renforcement profond. Pour chacune : la diapositive de sa conférence d'ICLR ou la page de son document de NeurIPS qui la montre, puis les critères de lecture (Q, X, Y, chapitre 3) et les expériences (E, B, chapitre 5) qui la mesurent dans ce rapport.

	Faille	Où Pineau la montre	Ce qui la mesure ici
Le matériel : ce qu'il faut pour refaire			
F1	L'article publié sans son code	diapositive 6 (ICLR), page 29 (NeurIPS)	critère Q5
F2	Aucun plan pour refaire l'expérience, comme pour AlphaGo	diapositives 9 à 12 (ICLR)	critères Q3 et X7

	Faille	Où Pineau la montre	Ce qui la mesure ici
F3	Le matériel et les logiciels, qui font partie du résultat	pages 34 et 35 (NeurIPS)	critères X7 et Y10
Le réglage : hyperparamètres et références			
F4	Les hyperparamètres de la méthode, donnés en partie	page 29 (NeurIPS)	critère Q3
F5	La façon de choisir les hyperparamètres, tue	page 29 (NeurIPS)	critère Q6
F6	Des réglages qui changent tout et qui interagissent	diapositives 24 et 25 (ICLR)	critère Y9
F7	Des références moins bien réglées que la méthode	diapositive 25 (ICLR), page 29 (NeurIPS)	critères Q4 et X4, expérience E6
Les graines aléatoires et la statistique			
F8	Une mesure de performance jamais définie	diapositives 26 et 27 (ICLR)	critères Y1 et Y2, expérience B3
F9	Trop peu d'essais : trois ou cinq, jamais dix dans son tableau	diapositive 28 (ICLR)	critère X1, expérience E5
F10	Les meilleurs essais sur N, seuls publiés	diapositives 29 et 30 (ICLR)	critère X3, expériences E1 et E4
F11	Même code, autre graine aléatoire : deux « algorithmes »	diapositive 31 (ICLR), page 22 (NeurIPS)	expériences E1 et B1
F12	Aucun test statistique	page 29 (NeurIPS)	critères Q8, X2, Y11 et Y12
La portée des conclusions			
F13	Le meilleur algorithme qui change avec la tâche	diapositives 20 et 21 (ICLR)	critères Y3 et Y4, expérience E2
F14	Deux implémentations du même algorithme qui divergent	diapositives 22 et 23 (ICLR)	critère Y5, expérience E3
F15	L'évaluation sur les conditions d'entraînement	pages 36 à 51 (NeurIPS)	critères Q7 et Y8, expérience E7
F16	Des bancs d'essai trop simples, faciles à apprendre par cœur	pages 43 à 50 (NeurIPS)	critères X6 et Y7

2.6.1 Le matériel : ce qu'il faut pour refaire

F1, l'article publié sans son code. Un article de RL décrit sa méthode en quelques pages. Le programme qui a produit les résultats compte des milliers de lignes, pleines de choix que la prose ne mentionne pas. Pineau reprend la formule de Buckheit et Donoho (1995) citée au chapitre 1, où ils condensaient une idée de Jon Claerbout : l'article est la publicité, la recherche est le code et les instructions qui ont produit les figures. Sur les 50 articles de RL de 2018 qu'elle a lus, 55 % donnent un lien vers leur code.

F2, aucun plan pour refaire l'expérience. Le code ne suffit pas sans les réglages exacts et une idée du calcul nécessaire. L'exemple de Pineau est AlphaGo (Silver et al. 2016). L'article est détaillé, avec une longue annexe. Sa diapositive demande pourtant si cela suffit : il faudrait aussi les parties d'experts, le code, les hyperparamètres, les ordinateurs et le protocole d'évaluation. Une découverte qu'aucun laboratoire extérieur n'a les moyens de refaire reste une affirmation.

F3, le matériel et les logiciels. Le même code lancé sur deux machines donne parfois deux résultats : les cartes graphiques n'arrondissent pas les calculs comme les processeurs, une nouvelle version d'une bibliothèque change une valeur par défaut. La liste de contrôle de Pineau demande l'infrastructure de calcul employée et le code « avec toutes ses dépendances ».

2.6.2 Le réglage : hyperparamètres et références

F4, les hyperparamètres de la méthode, donnés en partie. Un seul réglage oublié suffit à empêcher la reproduction : le lecteur prend la valeur par défaut de sa bibliothèque et obtient autre chose. Pineau compte que 90 % des articles de 2018 donnent tous les hyperparamètres de leur méthode.

F5, la façon de choisir les hyperparamètres, tue. Une valeur finale ne dit pas comment on l'a trouvée. Une équipe qui a essayé cent combinaisons et gardé la meilleure a optimisé sur son banc d'essai. Le lecteur a besoin du budget de recherche. Seuls 20 % des articles de 2018 le disent.

F6, des réglages qui changent tout. Henderson et ses coauteurs changent un réglage à la fois. Pour PPO sur HalfCheetah, passer d'un réseau de deux couches de 64 neurones à un réseau de 400 puis 300 neurones fait tomber le score de 2 201 à −1 180. Les réglages interagissent : le meilleur réseau pour un algorithme cesse de l'être pour un autre. Engstrom et al. (2020) puis Andrychowicz et al. (2021) ont confirmé cette sensibilité à grande échelle.

F7, des références moins bien réglées que la méthode. L'auteur d'une méthode nouvelle la règle avec soin. Il prend souvent la référence avec ses valeurs par défaut. La comparaison oppose alors une méthode réglée à une méthode qui ne l'est pas. Pineau compte que 60 % des articles de 2018 donnent les hyperparamètres de leurs références. Eimer et al. (2023) réclament le même budget de réglage pour les deux.

2.6.3 Les graines aléatoires et la statistique

F8, une mesure de performance jamais définie. Le score final, la moyenne sur l'entraînement et le meilleur score atteint ne classent pas toujours les méthodes dans le même ordre (section 2.5). Un article qui ne dit pas laquelle il emploie, ni ce que représentent ses barres d'erreur, ne se compare à aucun autre.

F9, trop peu d'essais. Pineau montre un tableau d'articles de 2015 à 2017 : ils lancent trois ou cinq graines aléatoires, aucun n'atteint dix. Avec trois essais, une moyenne dépend beaucoup du hasard d'un seul essai. Colas et al. (2018) calculent le nombre de graines aléatoires nécessaire pour détecter un écart donné.

F10, les meilleurs essais sur N. Une équipe lance dix essais, garde les meilleurs et publie leur moyenne. Chaque chiffre est vrai, mais la moyenne publiée dépasse ce que la méthode donne en moyenne. La dispersion paraît aussi plus petite qu'elle n'est. Pineau l'illustre par dix courbes dont elle garde les trois meilleures.

F11, même code, autre graine aléatoire. Henderson et ses coauteurs entraînent dix fois TRPO sur HalfCheetah en changeant seulement la graine aléatoire, forment deux groupes de cinq et tracent les deux courbes moyennes. Elles sont nettement séparées. Un test les déclare différentes ($t = -9{,}09$, $p = 0{,}0016$). Pineau précise sur scène que les groupes ne sont pas triés. Les chapitres 5 et 6 reviennent longuement sur ce chiffre (expériences E1 et B1).

F12, aucun test statistique. Quand deux méthodes diffèrent de quelques points, un test ou un intervalle dit si l'écart dépasse ce que le hasard des graines produit seul. Seuls 5 % des articles de 2018 appliquent un test à leurs comparaisons.

2.6.4 La portée des conclusions

F13, le meilleur algorithme qui change avec la tâche. Henderson et ses coauteurs classent quatre algorithmes sur quatre tâches. L'algorithme en tête sur HalfCheetah cesse de l'être sur Hopper ou Swimmer. Une conclusion générale tirée d'une ou deux tâches généralise un cas particulier.

F14, deux implémentations qui divergent. Trois implémentations de TRPO, avec le même réseau, finissent sur HalfCheetah à 1 939, 834 et 205. Comparer sa méthode à « TRPO » sans dire quelle implémentation on a prise revient à la comparer à un programme anonyme.

F15, l'évaluation sur les conditions d'entraînement. En apprentissage supervisé, on évalue sur des données jamais vues. En RL, l'agent s'évalue souvent dans le monde même de son entraînement. Pineau projette une phrase qu'elle met en débat, selon laquelle le RL serait le seul cas de l'apprentissage automatique où tester sur son ensemble d'entraînement est acceptable. Elle propose de séparer les graines aléatoires d'entraînement et de test, puis les tâches (Zhang et al. 2018a,b). Seuls 10 % des articles de 2018 évaluent sur des conditions tenues à part.

F16, des bancs d'essai trop simples. Un agent assez gros retient la bonne suite d'actions d'un banc d'essai déterministe sans rien comprendre du monde (Machado et al. 2018). Pineau réclame aussi la version de chaque banc d'essai, pour qu'un résultat se refasse dans le même monde.

Les remèdes de 2018. Aux seize failles, Pineau oppose d'abord une liste de contrôle de reproductibilité (pages 31 à 33 à NeurIPS). Pour chaque algorithme : une description claire, une analyse de complexité, le lien vers le code et ses dépendances. Pour chaque résultat théorique : l'énoncé, les hypothèses, la preuve. Pour chaque figure : la collecte des données, la répartition entre entraînement, validation et test, les données exclues, la plage et la méthode de choix des hyperparamètres, le nombre exact d'essais, la définition des statistiques et des barres d'erreur, le matériel employé. Elle présente aussi le défi de reproduction d'ICLR 2018 (diapositives 37 à 43 à ICLR), où n'importe qui pouvait tenter de refaire un article soumis. NeurIPS a adopté la liste en 2019 (Pineau et al. 2021).

2.7 Travaux voisins

Le 9 novembre 2026, nous avons cherché sur arXiv, sur OpenReview et dans les actes des congrès. À notre connaissance, aucun travail ne réapplique la grille de Pineau à des articles de RL récents en la calibrant sur 2018. Aucun ne refait le protocole de Henderson et ses coauteurs avec les algorithmes de 2026 sous un protocole écrit d'avance. Les travaux voisins se rangent en cinq groupes.

L'expérience de 2018. Henderson et al. (2018) fondent la plupart des failles F6 à F14 : sensibilité aux hyperparamètres, aux implémentations, aux graines aléatoires, aux tâches. Islam et al. (2017) les précèdent sur les mêmes tâches. Lynnerup et al. (2020) refont des algorithmes sur un vrai bras robotique. Bjorck et al. (2022) montrent, pour le contrôle à partir d'images, qu'une part de la variance entre graines vient d'essais qui échouent tôt par instabilité numérique.

La méthode statistique. Colas et al. (2018) font une analyse de puissance du nombre de graines aléatoires. Jordan et al. (2020) discutent la mesure de performance elle-même, puis Jordan et al. (2024) le coût d'une comparaison rigoureuse. Agarwal et al. (2021) apportent l'IQM, les intervalles par bootstrap stratifié, les profils de performance et la probabilité d'amélioration, réunis dans la bibliothèque rliable. Ils montrent que plusieurs améliorations annoncées sur les jeux Atari se fondent dans l'incertitude une fois les intervalles calculés. Patterson et al. (2024) en tirent un guide. Eimer et al. (2023) réclament un budget de réglage égal pour les références. Ces travaux disent quoi faire. Notre sondage mesure si les articles le font.

Les listes de contrôle et les défis de reproduction. Pineau et al. (2021) décrivent le programme de reproductibilité de NeurIPS 2019 : la liste de contrôle obligatoire, le défi de reproduction et la politique de dépôt du code. La part d'articles acceptés avec du code passe de moins de 50 % à 74,4 %. Magnusson et al. (2023) mesurent l'effet de la liste de contrôle des congrès de traitement automatique des langues.

Les audits de la littérature. Gundersen et Kjensmo (2018) notent 400 articles des congrès IJCAI et AAAI sur la documentation des méthodes, des données et des expériences. Coakley et al. (2026) suivent 56 800 articles de cinq congrès d'intelligence artificielle de 2014 à 2024 : la part qui partage code et données passe de 11 % à 64 %. La hausse commence avant les listes de contrôle. En RL, l'audit le plus proche est celui de Gorsane et al. (2022) : 75 articles de RL coopératif à plusieurs agents, de 2016 à 2022, dont environ un tiers ne quantifie aucune incertitude. Aucun de ces audits ne vise le RL à un seul agent ni la grille de 2018.

Les modèles de langue. Hochlehnert et al. (2025) réévaluent des modèles de raisonnement et trouvent des écarts types de 5 à 15 points de pourcentage d'une graine aléatoire d'échantillonnage à l'autre sur de petits bancs d'essai comme AIME 2024. Chandak et al. (2025) relèvent, dans un billet non relu par des pairs, des modèles de départ sous-évalués dans sept articles, l'équivalent moderne de la faille F7. Shao et al. (2025) montrent qu'une récompense tirée au sort améliore les modèles Qwen en mathématiques, mais pas les modèles Llama ou OLMo. Yue et al. (2025) montrent que la mesure pass@k renverse un classement selon k, un exemple de la faille F8.

2.8 Le vocabulaire des modèles de langue

Le vocabulaire du RL a gagné les modèles d'IA générative qui écrivent du texte. On emprunte au RL certains de ses algorithmes, avec une ressemblance vague, pour affiner les réponses d'un modèle de langue préentraîné, sans agent qui agit dans un monde. Ce rapport fixe ses mots comme suit.

- **RL classique.** Le RL au sens de la section 2.1 : un agent, souvent entraîné de zéro, agit dans un environnement doté d'états qui évoluent, en robotique, dans les jeux ou en contrôle. Le RL hors ligne en fait partie. C'est la famille `classic` du critère X5.
- **RL de grands modèles.** Une étiquette de commodité pour les articles qui appellent RL ce qu'ils appliquent à un modèle génératif préentraîné conditionné par le langage (modèle de langue, modèle de vision-langage, modèle texte-image). C'est la famille `llm` du critère X5 (chapitre 3). L'étiquette reprend le mot des articles. Elle ne vaut pas jugement sur la nature de ce qu'ils font.
- **RLHF.** L'apprentissage à partir de retours humains (*reinforcement learning from human feedback*) (Christiano et al. 2017 ; Ouyang et al. 2022). Des annotateurs classent des réponses, un modèle de récompense apprend leurs préférences, puis le modèle de langue est affiné pour maximiser ce score. Le sigle contient les lettres RL. Ce rapport ne range pas pour autant RLHF dans le RL : la politique optimisée produit un texte en réponse à une invite, puis reçoit une seule note, sans environnement dont l'état évolue sous ses actions.
- **RLVR et GRPO.** Le RLVR (*reinforcement learning with verifiable rewards*) remplace le modèle de récompense par un programme qui vérifie la réponse : la réponse finale d'un problème de mathématiques comparée à la bonne, un code soumis à des tests (DeepSeek-AI 2025). GRPO (*Group Relative Policy Optimization*) (Shao et al. 2024) est l'algorithme le plus répandu de cette famille. Il dérive de PPO : il tire un groupe de réponses pour chaque invite et remplace le critique par la récompense de chaque réponse centrée et réduite au sein du groupe. Notre volet GRPO est décrit au chapitre 7.

Les grands modèles de langue sont, dans ce rapport, des modèles d'IA générative. Les lecteurs de notre sondage sont des agents bâtis sur un tel modèle (chapitre 3). Le mot « agent » désigne alors un programme qui lit un PDF et remplit une grille, pas l'agent d'un processus de décision markovien.

CHAPITRE 3

Le sondage : méthode

Le premier instrument de ce rapport relit des articles publiés avec la grille de lecture de Pineau, complétée par sept critères tirés de sa conférence d'ICLR et par une seconde grille à nous. Ce chapitre décrit la méthode en entier : le calendrier et le protocole (section 3.1), les populations et les tirages (section 3.2), les grilles et leurs règles de décision (section 3.3), les lecteurs (section 3.4), les doubles lectures (section 3.5), l'harmonisation (section 3.6), les contrôles des citations et des décisions négatives (section 3.7), la calibration sur 2018 et la règle de verdict (section 3.8), puis les statistiques (section 3.9) et les analyses de robustesse (section 3.10). La section 3.11 résume les écarts au protocole. Le chapitre 4 donne les résultats.

3.1 Le protocole et son calendrier

Un protocole écrit avant la lecture. Le protocole du sondage (`study/PROTOCOL.md`) est daté du 8 novembre 2026, avant la lecture du premier article. Il fixe la question, les populations, les tirages, la grille, les règles de décision, la double lecture et l'analyse. Tout changement ultérieur s'inscrit en bas du fichier, dans une liste numérotée d'« écarts ». Chaque écart est daté et dit s'il a été décidé avant ou après que le résultat concerné a été connu. Le fichier en compte 29 au 21 novembre 2026. La fiche remise aux lecteurs (`study/GRILLE.md`) résume le protocole. Le plan de la seconde grille (`study/planification_v2.md`), écrit le 9 novembre 2026 avant tout codage, a été rendu public tel quel le 15 novembre 2026.

Les étapes. La figure 3.1 montre le flux complet, des populations aux comptes finaux. Les étapes se sont succédé ainsi.

1. **8 novembre.** Protocole, premier tirage de 50 articles de 2025 et 2026, calibration sur 12 articles de 2018, lecture avec la grille v1 (Q1 à Q8, X1 à X7) par sept lecteurs, dont un second lecteur.
2. **9 novembre.** Grille v2 (Y1 à Y12) et ses règles de décision, écrites avant tout codage (écarts 11 et 14). Élargissement des deux échantillons, écrit avant tout tirage (écart 12). Pilote (écarts 16 et 17), lecture élargie, audit des lectures brutes, harmonisation (écarts 18 et 19), puis premiers pourcentages. Règle de verdict à grille égale (écart 20).
3. **10 et 11 novembre.** Analyses de robustesse (écart 21), déclarations des listes de contrôle (A5, écart 22), sensibilité au filtre de titre (A6, écart 24), trajectoire de 2019 à 2024 (A4, écarts 25 et 26).
4. **15 novembre.** Recherche mécanique contre les décisions négatives (écart 27), provenance des lecteurs (écart 28), sensibilité au changement de modèle (écart 29).

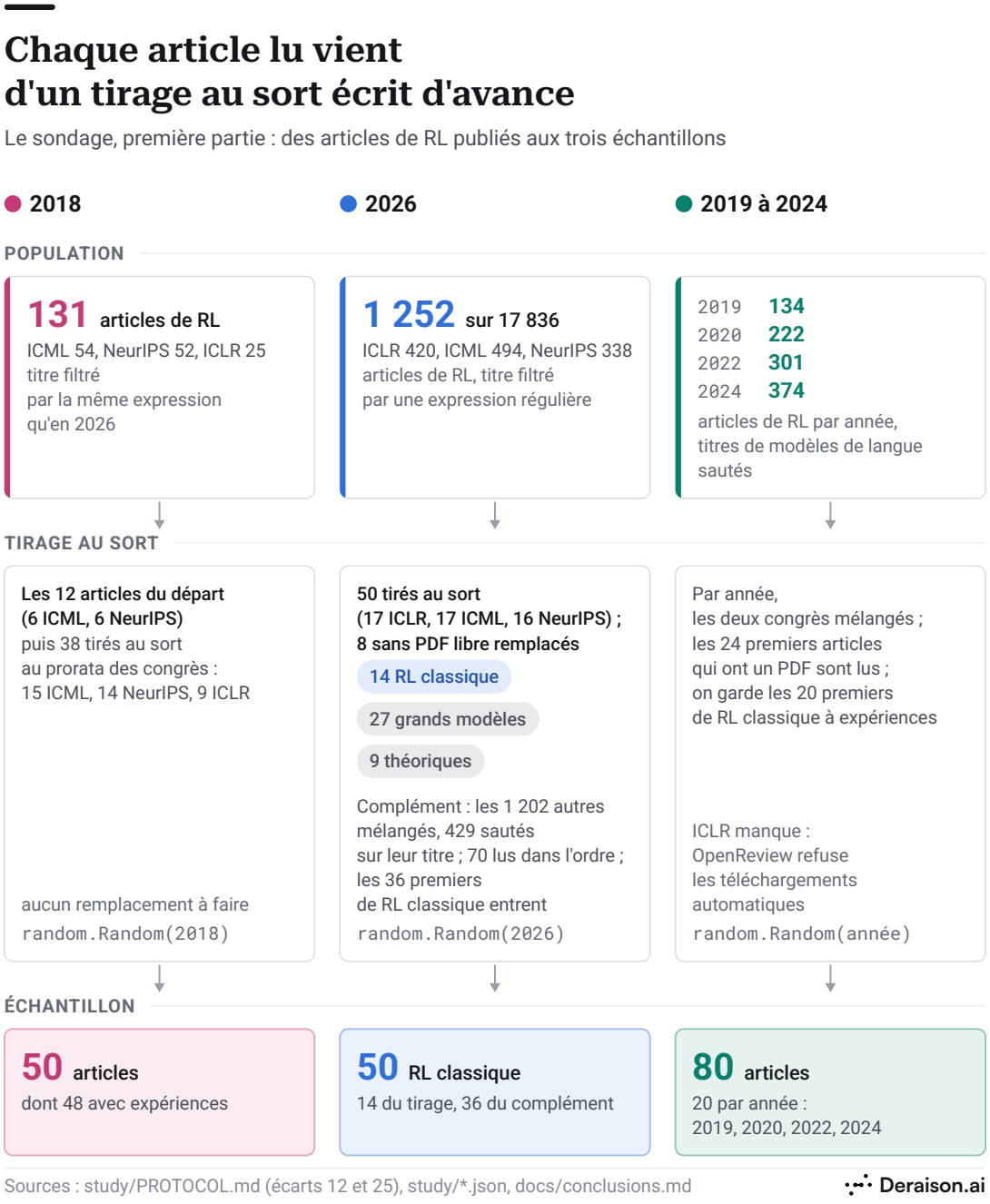


FIGURE 3.1. Le sondage, première partie : des populations aux échantillons. Les articles de RL d'ICML, de NeurIPS et d'ICLR 2018, ceux d'ICLR 2026, d'ICML 2026 et de NeurIPS 2025, ceux d'ICML et de NeurIPS en 2019, 2020, 2022 et 2024. Chaque échantillon vient d'un tirage dont la graine aléatoire est écrite dans le protocole : 50 articles de 2018 (48 expérimentaux), 50 articles de RL classique de 2026, 20 articles par année pour 2019, 2020, 2022 et 2024. Le premier tirage de 2026 est gardé comme contexte.



Figure 3.2. Le sondage, seconde partie : de la lecture aux comparaisons. Chaque décision des lecteurs cite une phrase de l'article avec sa page. Trois séries de doubles lectures à l'aveugle, l'harmonisation des règles, l'audit des citations et la recherche de ce qui pourrait démentir chaque « non » mettent ces décisions à l'épreuve avant les comparaisons finales.

3.2 Populations et tirages

Ce qu'est un article de RL. Un article appartient à la population s'il est accepté au programme principal d'une conférence et si son titre correspond, sans tenir compte de la casse, à l'expression régulière

```
reinforcement learning|\bRL\b|\bRLHF\b|\bRLVR\b|policy optimi[sz]ation|
          policy gradient|q-learning|actor-critic|\bMDPs?\b
```

La même expression sert pour toutes les années. Les trois scripts de tirage la recopient à l'identique : `study/sample_v2.py`, `study/trajectory_sample.py` et `study/title_filter.py`. Un filtre sur le titre rate les articles de RL dont le titre ne le dit pas. L'analyse A6 mesure cette perte (section 3.10).

Les articles de 2026. Les trois conférences les plus récentes au 8 novembre 2026 sont ICLR 2026, ICML 2026 et NeurIPS 2025. Leurs listes publiques comptent 17 836 articles, dont 1 252 de RL : 420 à ICLR, 494 à ICML, 338 à NeurIPS. La liste des articles de RL est `study/rl_population_2026.json`. Le premier tirage prend 17 articles d'ICLR, 17 d'ICML et 16 de NeurIPS avec `random.Random(2018)`, dans cet ordre, par `sample()` sur la liste de chaque conférence. Le script `study/draw_2026.py` refait ce tirage et vérifie qu'il retrouve les mêmes articles dans le même ordre. Les PDF viennent des actes officiels pour ICLR, d'arXiv pour ICML et NeurIPS, OpenReview refusant les téléchargements automatiques. Un article arXiv est retenu quand son titre ressemble à 85 % au moins à celui de la liste (`difflib.SequenceMatcher`). Huit articles sans PDF librement accessible sont remplacés, dans l'ordre, par un tirage `random.Random(2026)` dans le reste de la même conférence (`study/replacements_2026.json`). L'échantillon final est `study/final_2026.json`.

La calibration initiale sur 2018. Pour vérifier que nos lecteurs retrouvent à peu près les chiffres de Pineau, le protocole ajoute 6 articles de RL d'ICML 2018 et 6 de NeurIPS 2018, avec le même filtre et la même graine (`study/calib_2018.json`). Les annexes séparées de 2018 sont lues : le matériel supplémentaire de PMLR pour ICML, les archives des actes pour NeurIPS (écart 8).

L'élargissement A2 : 50 articles de 2018. Avec 12 articles de 2018 et 14 articles de RL classique de 2026, l'audit statistique du premier tirage (`study/audit_stats.py`) n'a pas la puissance de détecter un écart de 30 points. L'écart 12, écrit le 9 novembre 2026 avant tout tirage et toute lecture des articles ajoutés, élargit les deux échantillons. Pour 2018, la population réunit les trois conférences que Pineau annonce : 131 articles de RL, dont 54 à ICML, 52 à NeurIPS et 25 à ICLR (`study/rl_population_2018.json`). La liste d'ICLR 2018 vient du programme officiel de la conférence, les interfaces d'OpenReview et de dblp refusant les requêtes automatiques le 9 novembre 2026 : 336 articles du programme principal, oraux et affiches réunis. Les 50 articles se répartissent entre conférences en proportion de leur taille, par la méthode des plus forts restes : 21 pour ICML, 20 pour NeurIPS, 9 pour ICLR. Les 12 articles initiaux restent. Les 38 autres (15, 14 et 9 par conférence) sont pris dans l'ordre d'un mélange `random.Random(2018)` du reste de la population. Un article sans PDF librement accessible serait remplacé par le suivant dans cet ordre. Aucun remplacement n'a été nécessaire (`study/replacements_2018.json`). Les PDF viennent des actes de PMLR et de NeurIPS, d'une copie de l'Internet Archive quand le site des actes ne répondait pas, ou d'arXiv par le titre. Pour ICLR

2018, OpenReview refusant les téléchargements automatiques, le PDF vient de la copie de l'Internet Archive de la même adresse OpenReview, sinon d'arXiv. L'échantillon est `study/calib_2018_v2.json`.

L'élargissement A3 : 50 articles de RL classique de 2026. La population de départ est celle des 1 252 articles, moins les 50 du premier tirage, soit 1 202 articles, mélangés avec `random.Random(2026)`. Un tri sur le seul titre, fixé dans l'écart 12, saute les titres qui contiennent « LLM », « language model », « reasoning », « RLHF », « RLVR », « GRPO », « preference », « alignment », « reward model », « chain-of-thought », « math » ou « agentic ». Le tri écarte 429 articles (`study/classic_screened_out_2026.json`) et laisse une file ordonnée de 773 articles (`study/classic_queue_2026.json`). Les lecteurs codent la famille de chaque article de la file (critère X5) dans l'ordre. La lecture s'arrête quand 36 articles `classic` sont codés. Les 9 articles d'ICLR 2026 en tête de file étaient injoignables le 9 novembre 2026, le site des actes étant en panne : ils sont sautés. La file est figée pour que la sélection ne dépende pas de la date de lecture (écart 17). Au total, 70 articles de la file ont été lus : 36 entrent dans l'échantillon de RL classique, 18 relèvent des grands modèles de langue et 3 de la théorie, les 13 derniers ont été lus après le dernier article retenu (tableau C.1). Les 36 premiers articles `classic` dans l'ordre du tirage forment l'échantillon. Les 18 articles `llm` et les 3 articles `theory` lus avant le trente-sixième `classic` restent codés et rapportés à part. Ils ne forment pas un échantillon, puisque le tri sur le titre en a écarté beaucoup d'autres. L'échantillon de RL classique de 2026 réunit les 14 articles `classic` du premier tirage et ces 36 articles.

La trajectoire A4 : 2019, 2020, 2022 et 2024. L'écart 25, écrit le 10 novembre 2026 avant tout tirage, ajoute quatre années pour dater le changement. La population de chaque année réunit les articles de RL des actes principaux d'ICML (volumes PMLR 97, 119, 162 et 235) et de NeurIPS. ICLR manque : OpenReview refuse les téléchargements automatiques. Après le tri sur le titre de l'écart 12, la population compte 134 articles en 2019, 222 en 2020, 301 en 2022 et 374 en 2024 (`study/trajectory_queue.json`). L'ordre de lecture est un mélange `random.Random(<année>)` des deux conférences réunies. Les lecteurs lisent les 24 premiers articles de chaque année qui ont un PDF. L'échantillon prend, par année, les 20 premiers articles `classic` à expériences dans l'ordre. En 2019 et 2020, les 24 premiers articles n'en comptaient que 19 et 18 : les trois suivants de chaque année ont été lus, comme prévu. Huit lecteurs (M1 à M8) lisent chacun douze articles pris dans toutes les années, pour ne pas confondre un lecteur avec une année. Les annexes de NeurIPS 2019 à 2022 sont des archives zip, que le premier téléchargement avait laissées de côté. Les lectures brutes de la première passe sont enregistrées dans le dépôt (commit `0d3519a`), puis chaque lecteur relit ses articles avec l'annexe et change les décisions qu'elle justifie.

Les analyses A5 et A6. L'analyse A5 (écart 22) prend les 30 articles de NeurIPS 2025 déjà lus, sans tirage : 16 du premier tirage et 14 de la file A3. L'analyse A6 (écart 24) tire 400 articles parmi les articles des trois listes de 2026 dont le titre ne correspond pas à l'expression, avec `random.Random(2029)`.

3.3 Les grilles de lecture

Deux versions de la grille. Le tableau 3.2 liste les 27 critères. La grille v1 réunit les huit critères de Pineau (Q1 à Q8), tels qu'elle les présente à NeurIPS (page 29), puis sept

TABLE 3.1. Les échantillons du sondage. Pour chacun : la population, la règle de tirage avec sa graine aléatoire, ce qui a été lu et ce qui entre dans l'analyse. Sources : les fichiers JSON de `study/` nommés dans le texte.

Échantillon	Population	Tirage	Analysé
Premier tirage 2026	1 252 articles de RL d'ICLR 2026, ICML 2026, NeurIPS 2025	`Random(2018)` : 17, 17 et 16, puis remplacements par `Random(2026)`	50 articles, 45 expérimentaux
Calibration initiale 2018	RL d'ICML et de NeurIPS 2018	`Random(2018)` : 6 et 6	12 articles
A2, 2018	131 articles de RL d'ICML, NeurIPS et ICLR 2018	quotas 21, 20 et 9, mélange `Random(2018)`	50 articles, 48 expérimentaux
A3, RL classique 2026	1 202 articles restants, 773 après tri sur le titre	mélange `Random(2026)`, arrêt à 36 `classic`	70 lus, 14 + 36 = 50 retenus
A4, 2019 à 2024	134, 222, 301 et 374 articles après tri	mélange `Random(<année>)`	20 articles par année
A5, listes de contrôle	30 articles de NeurIPS 2025 déjà lus	aucun	30 articles
A6, filtre de titre	articles de 2026 hors filtre	`Random(2029)` : 400	400 résumés, 20 articles lus

critères tirés de sa conférence d'ICLR (X1 à X7). La grille v2 (Y1 à Y12) couvre les failles que la v1 laisse sans mesure. Elle a été écrite le 9 novembre 2026 (écart 11), avec ses règles de décision (écart 14), avant tout codage v2. Y11 et Y12 ont été ajoutés le même jour, après la recherche de travaux voisins, toujours avant tout codage. L'annexe A reproduit les règles en entier.

TABLE 3.2. Les critères de lecture. La colonne « Valeurs » donne les réponses permises. Un critère `yes/no/na` vaut `na` quand il ne s'applique pas. La dernière colonne donne la faille du tableau 2.2 que le critère mesure, ou son rôle.

Code	Critère	Valeurs	Faille ou rôle
Grille v1 : les critères de Pineau			
Q1	L'article a des expériences	oui, non	portée
Q2	Il emploie des réseaux de neurones	oui, non, na	population
Q3	Tous les hyperparamètres de la méthode sont donnés	oui, non, na	F2, F4
Q4	Tous les hyperparamètres des références sont donnés	oui, non, na	F7
Q5	Le code est en lien	oui, non, na	F1
Q6	La façon de choisir les hyperparamètres est décrite	oui, non, na	F5
Q7	L'évaluation se fait sur des conditions tenues à part	oui, non, na	F15
Q8	Un test statistique est appliqué à une comparaison	oui, non, na	F12
Grille v1 : nos ajouts tirés de la conférence d'ICLR			
X1	Nombre d'entraînements indépendants des résultats principaux	entier	F9

Code	Critère	Valeurs	Faille ou rôle
X2	Une incertitude est rapportée sur les résultats principaux	oui, non, na	F12
X3	Les meilleurs essais sont sélectionnés	oui, non, na	F10
X4	L'article dit avoir réglé les références avec le même soin	oui, non, na	F7
X5	Famille : `llm`, `classic` ou `theory`	une des trois	strate
X6	Pour `llm` : nombre de familles de modèles de base	entier	F16
X7	Le calcul employé est chiffré	oui, non, na	F2, F3
Grille v2			
Y1	La mesure de performance est définie	oui, non, na	F8
Y2	Les barres d'erreur sont définies	oui, non, na	F8
Y3	Nombre de tâches de la conclusion principale	entier	F13
Y4	Une tâche où la méthode ne gagne pas est montrée	oui, non, na	F13
Y5	L'implémentation des références est identifiée	oui, non, na	F14
Y6	Les exclusions sont dites, ou leur absence	oui, non, na	liste de contrôle
Y7	La version des environnements ou bancs d'essai est donnée	oui, non, na	F16
Y8	Un test de généralisation hors des conditions d'entraînement	oui, non, na	F15
Y9	La sensibilité à au moins un hyperparamètre est montrée	oui, non, na	F6
Y10	Les dépendances logicielles sont listées avec leurs versions	oui, non, na	F3
Y11	Le type d'incertitude est nommé	oui, non, na	F12
Y12	Un outil statistique postérieur à 2018 est employé	oui, non, na	F12

Règles de décision de la grille v1. Le protocole fixe les règles suivantes, complétées par les écarts 1 à 6.

- **Q3 et Q4.** `yes` si un tableau ou un paragraphe donne les hyperparamètres de toutes les méthodes principales. « Valeurs par défaut de [bibliothèque] » sans les lister compte `yes` seulement si la version de la bibliothèque est donnée. Q4 vaut `na` sans référence comparée.
- **Q5.** Un lien vers du code dans l'article. « Sur demande », « sera publié » et « will be made public at <URL> » valent `no`. Une page de projet compte `yes` seulement si elle mène à du code, ce qui a été vérifié en visitant chaque page (écarts 6 et 19).
- **Q6.** La méthode de choix est dite : grille, recherche aléatoire, budget, critère. « Tuned » sans plus vaut `no`.

- **Q7.** Évaluation sur des tâches, environnements, graines d'environnement ou jeux de données non vus à l'entraînement ni au réglage. Pour `llm`, des bancs d'essai distincts des données d'entraînement RL comptent `yes`. Un article `theory` à expériences jouets vaut no (écart 4).
- **Q8.** Un test appliqué à une comparaison : t, Welch, Mann-Whitney, bootstrap avec intervalle, IQM de rliable avec intervalles stratifiés. Des barres d'erreur seules valent no pour Q8 et `yes` pour X2.
- **X1.** Le plus petit nombre d'entraînements indépendants annoncé pour un tableau ou une courbe principale. Pour `llm`, les échantillons d'évaluation (pass@k, « 16 samples ») ne comptent pas.
- **X3.** `yes` aussi quand les tableaux de l'article montrent que le résultat principal est le meilleur de plusieurs entraînements identiques, même si le texte ne le dit pas (écart 5).
- **X4.** `yes` exige un réglage décrit pour les références ou un budget de réglage égal annoncé. « Même configuration que la méthode », sans réglage décrit pour personne, vaut no (écart 1).
- **X5.** La famille se lit sur le modèle que le RL entraîne (écart 18) : `llm` si la politique optimisée est un modèle génératif préentraîné conditionné par le langage, `classic` sinon, même au service d'un grand modèle. `theory` est réservé aux articles sans expérience ou à expériences jouets seulement.
- **X7.** `yes` exige au moins une quantité : heures, GPU-heures, FLOP, ou un nombre de GPU avec une durée. Le seul nom du matériel vaut no (écart 2).

Règles de décision de la grille v2. L'écart 14 fixe les règles, l'écart 17 les précise après le pilote, l'écart 19 après l'audit des lectures brutes.

- **Portée.** Si Q1 vaut no, tous les autres critères valent na. Y2 et Y11 valent na si et seulement si X2 vaut no. Y4 vaut na si Y3 vaut 1. Y5 vaut na si et seulement si Q4 vaut na.
- **Y1.** L'article dit ce que contient le nombre rapporté et à quel point il est pris : retour final, moyenne sur les N derniers épisodes, meilleur point, aire sous la courbe, pass@k avec k et le nombre d'échantillons. « Return » ou « accuracy » sans plus vaut no. Une courbe dont les points d'évaluation sont dits vaut un point de mesure. Pour une quantité qui n'est pas le score d'une politique, la quantité définie suffit (écart 19).
- **Y2.** Ce que représentent la bande ou l'intervalle est dit. Des barres présentes mais non définies valent no.
- **Y3.** Un entier, le nombre de cadres d'entraînement distincts de la conclusion principale. Deux niveaux de difficulté d'un même jeu, ou les variantes de qualité d'un jeu de données hors ligne, comptent pour un. Un méta-entraînement compte pour un. Un cadre présent seulement en annexe ne compte pas. L'analyse transforme Y3 en `yes` quand l'entier vaut au moins 2.
- **Y4.** Un résultat tâche par tâche montre au moins une tâche où la méthode ne gagne pas.

- **Y5.** Pour chaque référence principale, l'origine du code est dite : dépôt, bibliothèque nommée, « réimplémenté par nous », chiffres recopiés avec leur source, modèle publié désigné par son nom exact. Une réimplémentation sous-entendue ne compte pas.
- **Y6.** Les essais, graines aléatoires ou données retirés sont dits, ou leur absence est dite. Le silence vaut no.
- **Y7.** La version de chaque environnement ou banc d'essai principal : suffixe Gymnasium, version de la suite, nom qui fixe un seul ensemble publié. « MuJoCo » seul vaut no. « MATH-500 » vaut yes, « MATH » seul vaut no.
- **Y8.** Au moins une évaluation sur des conditions absentes de l'entraînement et du réglage. Plus strict que Q7 : pour `llm`, un banc différent compte seulement s'il est présenté comme un test hors distribution. De nouvelles graines aléatoires d'entraînement ne sont pas des conditions tenues à part.
- **Y9.** Au moins un hyperparamètre de la méthode est balayé et le résultat montré. Une ablation de composant ne compte pas.
- **Y10.** Les versions des bibliothèques principales sont données, ou un fichier d'environnement est cité. Le dépôt n'est pas visité, sauf renvoi explicite au fichier.
- **Y11.** Le type d'incertitude est nommé quelque part, avec un champ `kind` (`std`, `se`, `ci`, `iqr`, `minmax`, `other`).
- **Y12.** Au moins un outil parmi l'IQM, le bootstrap stratifié, rliable, les profils de performance, la probabilité d'amélioration. Un bootstrap simple qui produit un intervalle rapporté compte.

Ce qu'une part mesure. Pour un critère et un échantillon, la part est le nombre de yes divisé par le nombre de yes et de no. Les articles sans expérience (Q1 no) et les valeurs na sortent du dénominateur (`tally()` dans `src/pineau_rl/grid.py`). Un dénominateur plus petit que la taille de l'échantillon signale donc des articles où le critère ne s'applique pas, par exemple un article sans référence comparée pour Q4. Pineau annonce 100 % d'articles expérimentaux : ses dénominateurs sont les 50 articles.

3.4 Les lecteurs

Des agents de modèle de langue. La lecture est confiée à des agents de modèle de langue : des sous-agents de Claude Code, chacun lancé avec une consigne écrite, une liste d'identifiants et un dossier où écrire. Le dossier `study/readers/` donne la provenance de chaque agent, reconstruite le 15 novembre 2026 à partir des transcriptions des sessions (écart 28) : la consigne exacte (`prompts/<lecteur>.md`), le modèle relevé dans la transcription, les dates, les articles écrits et la manière de lire les PDF (`readers.csv`). Les 38 dossiers de lecture correspondent chacun à un agent et à ses articles, sans écart (`folder_check.csv`). Tous les lecteurs ont tourné sur le modèle `claude-opus-5-5`, sauf deux qui ont fini sur `claude-opus-5` après qu'un filtre de contenu du fournisseur a interrompu une réponse : C2 pour les articles ML14 à ML17 et M4 pour neuf articles de A4, soit treize articles en tout.

Ce qu'un lecteur fait. Chaque lecteur lit l'article entier, annexes comprises. Il convertit le PDF en texte avec `pdftotext` et regarde souvent les images des pages. Il écrit un fichier

JSON par article, dans `study/codes/<lecteur>/` pour la grille v1 et `study/codes_v2/<lecteur>/` pour la v2. Pour chaque critère, le fichier donne la valeur, une citation exacte de l'article copiée mot pour mot et la page. Si rien ne justifie un `yes`, la citation est « absent après lecture des sections expériences, annexes et liens » et la page vaut `null`. Le lecteur note ses hésitations dans un champ `doubts`. Il ne lit jamais les codes d'un autre lecteur, ni les résultats, ni les documents de `docs/`. Pour un article déjà lu en v1, le lecteur v2 reçoit seulement les valeurs v1 de Q1, Q4 et X2, qui fixent les règles de portée.

Le contrôle mécanique. Le script `study/check_codes.py` contrôle chaque fichier v2 avant tout pourcentage. Il vérifie le schéma (critères présents, valeurs permises, entiers, champ `kind` de Y11) et les règles de portée (`validate()` dans `src/pineau_rl/grid.py`). Il cherche ensuite chaque citation dans le texte du PDF et de son annexe. Les notes du lecteur entre crochets sont retirées, une citation cousue de morceaux joints par « […] » est coupée en ses morceaux. Un morceau de moins de 12 lettres est trop court pour être testé. La comparaison se fait sur les lettres et les chiffres seuls, pour que les espaces, les césures, les ligatures et les symboles mathématiques ne comptent pas. Un morceau introuvable tel quel est cherché de façon floue : il passe si 90 % de ses caractères se retrouvent dans une fenêtre du texte. Un fichier en échec retourne à son lecteur. Un deuxième échec sur le même critère est tranché depuis le PDF et noté dans `study/overrides_v2.json`.

Trois faits que le protocole ne disait pas. La reconstruction de l'écart 28 a établi trois faits que le protocole ne disait pas. Les lecteurs C2 et M4 ont fini sur le modèle `claude-opus-5`. Le lecteur principal M7 a ouvert le fichier K002 du lecteur K1 comme exemple de format. K002 n'est pas à lui et n'entre dans aucune double lecture. Les lectures de la grille v1 (lecteurs A à G) ont eu lieu le 8 novembre dans une session ouverte dans un autre dossier que le dépôt. Les seconds lecteurs G, G2, R2 et R3 n'ont ouvert aucun fichier d'un autre lecteur.

3.5 Les doubles lectures à l'aveugle

Quatre seconds lecteurs. Un second lecteur relit des articles sans voir la première lecture ni savoir qui l'a faite.

- **G (grille v1) et G2 (grille v2).** Les 10 premiers articles du premier tirage par ordre d'identifiant après un mélange `random.Random(7)`.
- **R2 (deux grilles).** 10 articles tirés par `random.Random(7)` parmi les articles ajoutés par A2 et A3 : 5 de 2018 et 5 de 2026.
- **R3 (deux grilles).** 10 articles tirés par `random.Random(2030)` parmi les 80 de A4, relus avec les annexes (écart 26).

Ce qu'on compare. L'accord se calcule sur les lectures brutes, avant toute harmonisation : l'harmonisation s'applique aux deux lectures et gonflerait l'accord. Pour chaque critère, `study/agree_v2.py` garde les articles où les deux lecteurs ont répondu `yes` ou `no` (Y3 transformé comme dans l'analyse), puis calcule la part de réponses identiques et le kappa de Cohen quand les deux lecteurs varient. Un critère sous 70 % d'accord est

dit fragile. L'accord de la grille v1 sur le premier tirage vient de `study/audit_stats.py`, qui compte aussi les valeurs `na`.

La relecture humaine prévue, puis abandonnée. L'écart 13 prévoyait qu'un humain, l'auteur, relise seul 10 articles de 2026 sur sept critères, sans voir les lectures des agents. Les articles avaient été tirés par `random.Random(2027)`. L'écart 23 abandonne cette relecture le 10 novembre 2026, avant qu'elle commence. Le dossier `study/relecture_humaine/` reste comme trace du plan. Aucun humain n'a donc relu les articles à l'aveugle. Chaque décision reste vérifiable par sa citation et sa page.

3.6 Le pilote et l'harmonisation

Le pilote. Avant les lecteurs principaux, un lecteur pilote code trois articles (écart 16) : LR08 (`llm`) et ML15 (`classic`), choisis par `random.Random(11)` parmi les 40 articles de 2026 hors double lecture, puis C218, premier des 38 nouveaux articles de 2018 après un mélange `random.Random(11)`. Le pilote sert seulement à trouver les règles ambiguës et à tester `check_codes.py`. Ses fichiers (`codes_v2/pilot/`) n'entrent dans aucune analyse. Les trois articles sont recodés par leur lecteur normal. L'écart 17 écrit les précisions qui en découlent, avant le codage principal et avant tout pourcentage v2.

L'harmonisation de la grille v1. Les écarts 1 à 6 ont été écrits pendant le codage du premier tirage, avant l'agrégation. Ils s'appliquent depuis les citations, dans `study/overrides.json`. La vérification des citations introuvables a montré le 9 novembre 2026 que ces règles n'avaient été appliquées qu'aux articles de 2026 (écart 15). La comparaison entre années mêlait donc deux grilles un peu différentes. Les mêmes règles ont été appliquées aux 12 articles de 2018, sans règle nouvelle. Pour X4, C11, C13, C14 et C16 passent à no. Pour Q7, C14 passe de na à no. Pour X7, C14 passe à no. Le script `study/analyze_final.py` régénère désormais `study/results_final.json`, jusque-là écrit sans script. Cette correction a été décidée après avoir vu les pourcentages, ce que le protocole déclare.

L'harmonisation de la grille v2. Après la lecture élargie, un audit relit tous les fichiers v2 des lecteurs principaux, critère par critère, sans calculer aucune part (`study/harmonisation_audit-fr.md`, 170 fichiers). Il relève des codes contraires aux règles et dix cas qu'aucune règle ne tranchait. L'écart 19, écrit le 9 novembre 2026 avant tout pourcentage, tranche ces cas pour les deux années. Ses décisions s'appliquent depuis les citations, chacune avec sa raison, dans `study/overrides_v2.json` : 80 changements, dont Y1 21, Y3 5 (et Y4 5 par la règle de portée), Y5 10, Y6 3, Y7 19, Y8 6, Y9 9 et Q5 2. Les pages de projet données comme lien vers le code ont été visitées le 9 novembre 2026. Cinq mènent au code, deux non. Le lien de K090 mène à un miroir anonyme expiré (HTTP 410). Celui de C19 mène à une bibliothèque publiée après l'article, sans ses scripts. Deux cas ont été tranchés par l'auteur du protocole : pour NE16 et ML14, dont toutes les expériences sont en annexe, la règle sur les cadres présents seulement en annexe ne s'applique pas. L'écart 18 précise X5 avant de figer l'échantillon de RL classique : appliquée aux 120 articles lus, la règle ne change qu'un code, K006, de `llm` à `classic`.

Trois critères sans règle nouvelle. Pour X3, X4 et X7, l'audit a relevé des cas limites, comme un calcul réduit au temps d'inférence ou des graines ratées écartées. Aucune

règle nouvelle n'a été ajoutée. Une règle écrite à ce moment ne se serait appliquée qu'aux articles de la lecture élargie, ce que l'écart 15 interdit : les deux années auraient de nouveau été lues avec deux grilles différentes. Ces cas restent codés comme le lecteur les a lus.

La valeur finale d'un critère. La fonction `value()` de `src/pineau_rl/grid.py` calcule la valeur qui entre dans l'analyse. Elle prend la lecture v2 du lecteur principal quand elle contient le critère, avec `overrides_v2.json`, sinon la lecture v1, avec `overrides.json`. Les lectures des seconds lecteurs et du pilote n'entrent jamais dans l'analyse.

3.7 Les contrôles des citations et des décisions négatives

L'audit des citations de la grille v1. Le premier passage du contrôle automatique sur les fichiers v1 a laissé 89 citations introuvables. Un agent d'audit les a toutes vérifiées dans les PDF (`study/v1_quotes_audit.md`). Aucune n'est inventée : 83 sont présentes mot pour mot, 1 est une paraphrase fidèle, 5 ont une valeur contraire aux règles. Les échecs venaient de commentaires en français ajoutés par le lecteur à une citation anglaise exacte, de la mise en page à deux colonnes que `pdftotext` entrelace, enfin de trois annexes de NeurIPS 2018 absentes du dossier `study/pdf/`. Les cinq valeurs contraires aux règles ont conduit à l'écart 15.

La recherche mécanique contre les « non ». Une décision `yes` s'appuie sur une citation et une page. Une décision `no` s'appuie sur rien. Or les verdicts « a changé » reposent surtout sur les `no` de 2018. L'écart 27, écrit le 15 novembre 2026 après les résultats et avant toute recherche, confronte au texte chaque décision finale `no` des deux échantillons de la comparaison principale, sur les sept critères Q4, Q5, Q7, Q8, X2, X7 et Y9. Le script `study/falsify_no.py` extrait le texte de l'article et de son annexe, retire la section des références, puis cherche des motifs fixés dans l'écart et jamais modifiés ensuite. Par exemple, le motif de Q5 est

```
    github|gitlab|bitbucket|anonymous\.4open|code (is |are |will be )?
(publicly )?available|source code|our (code|implementation)|open[- ]source
```

Chaque occurrence est gardée avec 300 caractères de contexte. Un arbitre, agent qui n'a lu aucun des deux échantillons et ne voit ni l'article ni l'année, dit selon les règles de `GRILLE.md` si l'extrait suffit à faire passer la décision à `yes`, ou s'il reste douteux. Les lots qu'il a vus et ses réponses brutes sont dans `study/falsify_batches/`. Le script `study/falsify_stats.py` refait ensuite la comparaison principale avec les décisions retournées, sans changer de test, comme l'écart l'annonce. Il la refait aussi avec les décisions retournées et les décisions douteuses, une borne que l'écart n'annonçait pas.

3.8 La calibration sur 2018 et la règle de verdict

Pourquoi relire 2018. Pineau ne décrit ni son tirage ni sa façon de trancher les cas douteux. Comparer nos lectures de 2026 à ses chiffres mêlerait deux changements : celui des pratiques et celui de la grille. La calibration sert à mesurer le second. Dès les 12 premiers articles, notre grille paraissait plus sévère que la sienne sur Q3 et plus large sur Q7 (écart 7). Le protocole prévoyait alors de ne conclure à un changement que s'il tenait à la fois contre ses chiffres et contre notre propre lecture de 2018.

La règle de verdict à grille égale. Avec 50 articles de 2018, la comparaison secondaire de l'écart 12 a confirmé que notre grille lit 2018 autrement que Pineau (chapitre 4). L'écart 20, décidé le 9 novembre 2026 après ces premiers résultats, en tire la règle du rapport : un verdict repose seulement sur la comparaison principale de l'écart 12, nos 50 articles de RL classique de 2026 contre notre lecture de 50 articles de 2018, avec les mêmes grilles et les mêmes lecteurs. Cette comparaison avait été fixée avant toute lecture des articles ajoutés. Les comparaisons avec les chiffres de Pineau restent montrées comme contexte. Le changement de règle a été décidé après avoir vu ces résultats. Le protocole le déclare. Un critère « a changé » quand l'écart passe le seuil de 5 % après la correction de Holm.

3.9 Les statistiques

Toutes les fonctions statistiques sont dans `src/pineau_rl/stats.py`. Elles appellent SciPy et statsmodels et sont déterministes. Les scripts qui les emploient sont `study/extension_stats.py` (comparaison principale et comparaisons secondaires), `study/trajectory.py` (A4), `study/agree_v2.py` et `study/audit_stats.py` (accord), `study/robustness.py` (robustesse), `study/falsify_stats.py` (décisions négatives). Dans ce qui suit, k est le nombre de `yes`, n le dénominateur et $\hat{p} = k/n$.

Intervalle de Wilson. L'intervalle à 95 % d'une part est celui de Wilson (1927), adapté aux petits échantillons et aux parts proches de 0 ou de 1. Avec $z = \Phi^{-1}(0{,}975)$,

$$\frac{\hat{p} + \frac{z^2}{2n}}{1 + \frac{z^2}{n}} \pm \frac{z}{1 + \frac{z^2}{n}} \sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}. \tag{3.1}$$

Intervalle de Newcombe pour une différence. Les 50 articles de Pineau forment eux aussi un échantillon. Traiter ses parts comme exactes, les 55 % d'articles avec code par exemple, gonflerait la sûreté des verdicts. L'écart $d = \hat{p}_a - \hat{p}_b$ entre deux échantillons indépendants reçoit donc l'intervalle hybride de Newcombe (1998) (sa méthode 10). Avec les intervalles de Wilson (l_a, u_a) et (l_b, u_b) de chaque part,

$$\left[d - \sqrt{(\hat{p}_a - l_a)^2 + (u_b - \hat{p}_b)^2} \; ; \; d + \sqrt{(u_a - \hat{p}_a)^2 + (\hat{p}_b - l_b)^2} \right]. \tag{3.2}$$

Le calcul est celui de `confint_proportions_2indep(method="newcomb")` de statsmodels. Pour un chiffre de Pineau, la part entre dans le calcul comme $k = \text{round}(50\,p)$ sur $n = 50$.

Test exact de Fisher. La valeur p d'une comparaison vient du test exact de Fisher (1922) sur le tableau $\left(\begin{smallmatrix} k_a & n_a - k_a \\ k_b & n_b - k_b \end{smallmatrix}\right)$. Sous l'hypothèse nulle et à marges fixées, k_a suit une loi hypergéométrique. La valeur p bilatérale somme les probabilités des tableaux aussi probables ou moins probables que celui observé (`scipy.stats.fisher_exact`).

Correction de Holm. Tester plusieurs critères à la fois multiplie les occasions de franchir le seuil par hasard. La correction de Holm (1979) contrôle la probabilité d'au moins un faux positif dans une famille de m tests. Avec les valeurs p triées $p_{(1)} \le \cdots \le p_{(m)}$, la valeur ajustée est

$$\tilde{p}_{(i)} = \min\Big(1, \; \max_{j \le i} (m - j + 1)\, p_{(j)}\Big). \tag{3.3}$$

Les familles sont fixées dans les scripts. La comparaison principale en a deux : les 11 critères v1 testés (Q2 à Q8, X2, X3, X4, X7) et, séparément, les 12 critères v2. Les comparaisons secondaires, 2026 contre Pineau et notre 2018 contre Pineau, ont chacune pour famille les six critères Q3 à Q8 où Pineau donne un chiffre comparable. Pour la trajectoire, chaque famille réunit les cinq critères testés (Q5, Y9, X7, X2, Q8).

Test de tendance de Cochran-Armitage. Pour la trajectoire A4, le test de Cochran (1954) et Armitage (1955) cherche une tendance linéaire de la part au fil des années. Avec $K = \sum_i k_i$, $N = \sum_i n_i$, $\bar{p} = K/N$, les scores x_i égaux aux années (2018, 2019, 2020, 2022, 2024, 2026) et $\bar{x} = \sum_i x_i n_i / N$,

$$T = \sum_i x_i \,(k_i - n_i \bar{p}), \qquad \mathrm{Var}(T) = \bar{p}(1-\bar{p}) \sum_i n_i (x_i - \bar{x})^2, \qquad z = T / \sqrt{\mathrm{Var}(T)}. \tag{3.4}$$

La valeur p bilatérale vient de la loi normale. Deux écarts de Newcombe complètent ce test, avec un test de Fisher : 2019 et 2020 réunies contre 2018 (l'effet immédiat de la liste de contrôle), puis 2022 et 2024 réunies contre 2019 et 2020 (l'effet tardif).

Kappa de Cohen. L'accord brut entre deux lecteurs dépend de la fréquence des réponses : deux lecteurs qui répondent presque toujours no s'accordent souvent par hasard. Le kappa de Cohen (1960) retire cette part :

$$\kappa = \frac{p_o - p_e}{1 - p_e}, \qquad p_e = \sum_c \pi_a(c)\,\pi_b(c), \tag{3.5}$$

où p_o est la part de réponses identiques et $\pi_a(c)$ la part des réponses c du lecteur a. Le kappa n'est pas défini quand $p_e = 1$. Le script `agree_v2.py` ne le calcule pas quand l'un des deux lecteurs donne toujours la même réponse. Sur dix articles, un ou deux désaccords suffisent à faire tomber le kappa vers 0,5 ou 0,6 quand un lecteur répond presque toujours de la même façon. Le kappa se lit donc avec la part d'accord et le nombre d'articles.

Test de Cochran-Mantel-Haenszel et corrélation de Spearman. Pour les analyses de robustesse, un test de Cochran-Mantel-Haenszel (Mantel et Haenszel 1959) combine les tableaux 2×2 de chaque conférence (ICML, NeurIPS, ICLR), avec une correction de Haldane de 0,5 par case et une correction de continuité (`statsmodels.stats.contingency_tables.StratifiedTable`). La corrélation de Spearman entre les notes de coût ou de risque et les écarts observés reçoit une valeur p par permutation : 10 000 permutations des écarts, générateur NumPy de graine 2026.

3.10 Les analyses de robustesse

Toutes ces analyses sont exploratoires. Aucune ne change un verdict de l'écart 20. Elles disent seulement si un verdict est fragile.

Quatre coupes des verdicts (écart 21, a). Pour les quatre critères qui ont changé (Q5, Y9, X7, X2) et les deux intervalles qui évitent zéro sans passer Holm (Q6, Y12), `study/robustness.py` refait la comparaison de quatre façons.

- **(a1) La provenance.** Le RL classique de 2026 est séparé selon sa provenance : les 14 articles du premier tirage, sans tri sur le titre, contre les 36 de la file A3, avec tri. La comparaison principale est refaite avec chaque provenance seule.
- **(a2) La source du PDF.** Les articles de 2026 sont séparés selon la source du PDF lu, arXiv ou site de la conférence.
- **(a3) La conférence.** Un test de Cochran-Mantel-Haenszel stratifie par conférence, pour écarter un effet de composition.
- **(a4) Le lecteur.** La comparaison est refaite en retirant tour à tour chaque lecteur principal.

Un verdict est dit fragile si l'écart change de signe dans une coupe, ou si l'intervalle de Newcombe refait contient zéro alors que chaque côté garde au moins 35 articles. La coupe (a2) a d'abord lu une copie locale abîmée d'un fichier. Refaite sur le fichier du dépôt, elle oppose 45 articles lus sur arXiv à 5 lus sur le site d'une conférence et ne peut rien trancher (écart 21, c).

Le coût et le risque de chaque pratique (écart 21, b). Une version antérieure de nos conclusions avançait que Pineau avait été écoutée là où l'écouter coûtait peu. Ce classement venait de nous, après avoir vu les écarts. Pour le mettre à l'épreuve, trois évaluateurs indépendants notent chacun des 17 critères de pratique de la comparaison principale sur deux échelles de 1 à 5 : le coût C pour un auteur qui a déjà ses résultats principaux, puis le risque R que la pratique affaiblisse ou renverse sa conclusion. Ce sont trois agents de modèle de langue de trois tailles différentes, sans accès aux résultats ni au dépôt, qui reçoivent seulement l'intitulé et la règle de chaque critère (`study/blind_cost/`). Le test est la corrélation de Spearman entre la médiane des trois notes et l'écart observé, X3 retourné parce qu'une part plus basse y est un progrès. Une corrélation négative nette soutiendrait cette affirmation antérieure.

Sans les treize articles lus en partie par un autre modèle (écart 29). Les treize articles finis par `claude-opus-5` sont retirés, puis la comparaison principale et la trajectoire sont refaites avec les mêmes tests (variable d'environnement `EXCLUDE`, sorties `extension_stats_sans_opus5.json` et `trajectory_sans_opus5.json`). Un verdict change si un écart ou une tendance passe d'un côté à l'autre du seuil après correction de Holm.

Le filtre de titre (A6, écart 24). Pour chacun des 400 articles tirés hors du filtre, le résumé est lu sur la page officielle. Un article est « de RL caché » si son résumé correspond à la même expression régulière que les titres. Le script `study/title_filter.py` donne la part de RL caché avec son intervalle de Wilson, le nombre estimé d'articles manqués (cette part multipliée par la taille de la population hors filtre) et leur fraction du total. Les 20 premiers articles de RL caché dans l'ordre du tirage sont lus avec les deux grilles, puis comparés de façon descriptive à l'échantillon de 2026 sur les quatre critères qui ont changé (`study/title_filter_compare.py`). Quatre d'entre eux n'avaient pas de PDF librement accessible et ont été remplacés par les suivants dans l'ordre.

Ce que les articles déclarent (A5, écart 22). Les articles de NeurIPS 2025 joignent la liste de contrôle remplie par leurs auteurs. Le script `study/checklist.py` retrouve la version des actes officiels de chacun des 30 articles par son titre, puis extrait la réponse et la justification de quatre questions : « Open access to data and code », « Experimental setting/details », « Experiment statistical significance », « Experiments compute

resources ». La comparaison, fixée d'avance, les oppose à nos codes harmonisés : Q5, Q3, X2 pour une incertitude et Q8 pour un test, X7. La mesure est la part des articles qui déclarent « Yes » alors que notre code vaut no, avec son intervalle de Wilson. Pour chaque désaccord sur le code, le script cherche un lien vers du code dans la version des actes, parce que nos codes viennent souvent de la version arXiv.

3.11 Les écarts au protocole

Le protocole du sondage compte 29 écarts. Le tableau 9.1 les donne tous avec leur date et leur position par rapport aux résultats, à côté de ceux des deux autres protocoles (chapitre 9). Ils se rangent en cinq groupes.

- **Règles précisées avant tout pourcentage** : écarts 1 à 6 (grille v1), 14, 16 à 19 (grille v2, pilote, famille X5, harmonisation). Ils lèvent des ambiguïtés sans viser un chiffre.
- **Instruments ajoutés avant leur propre calcul** : écarts 11 et 12 (grille v2, élargissement), 13 (relecture humaine, abandonnée par l'écart 23), 22, 24, 25 et 26 (A5, A6, A4 et sa seconde lecture).
- **Décisions prises après avoir vu des résultats** : écart 7 (calibration initiale), 15 (harmonisation de 2018 rattrapée), 20 (règle de verdict à grille égale), 21, 27, 28 et 29 (analyses de réponse à une relecture critique, écrites avant leur propre calcul). Le protocole les déclare comme telles.
- **Incidents sans effet sur les données** : écarts 9 (adresse de contact retirée de l'en-tête des téléchargements) et 10 (scripts de travail écrasés dans un dossier temporaire partagé, codages intacts).
- **Faits de lecture** : écart 8 (annexes de 2018 et de 2026).

CHAPITRE 4

Le sondage : résultats

Ce chapitre donne tous les résultats du sondage. La section 4.1 décrit les échantillons obtenus. La section 4.2 donne la comparaison principale à grille égale, critère par critère, puis la section 4.3 la relit faille par faille. La section 4.4 dit ce qui résiste aux analyses de robustesse. La section 4.5 compare notre lecture de 2018 aux chiffres de Pineau. La section 4.6 décrit le premier tirage de 2026 et les articles sur les modèles de langue. La section 4.7 date le changement de 2019 à 2024. La section 4.8 mesure l'accord entre lecteurs. Les sections 4.9 et 4.10 donnent les analyses A5 et A6. Tous les nombres viennent des fichiers JSON de `study/` nommés dans chaque légende.

4.1 Les échantillons obtenus

Le premier tirage de 2026. Les 50 articles tirés parmi les 1 252 articles de RL de 2025 et 2026 se répartissent ainsi : 27 appliquent ce qu'ils appellent RL à de grands modèles de langue (famille `llm`), 14 relèvent du RL classique (`classic`) et 9 sont théoriques (`theory`), dont 5 sans aucune expérience. Les pourcentages se calculent sur les 45 articles qui ont des expériences.

Notre lecture de 2018. Les 50 articles de 2018 (21 d'ICML, 20 de NeurIPS, 9 d'ICLR) comptent 48 articles expérimentaux. Sur ces 48 articles, un article (C14) compare une méthode exacte sans hyperparamètre : Q3 et Q6 y valent `na`, d'où un dénominateur de 47 pour ces deux critères. Q4 et Y5 valent `na` quand un article ne compare aucune référence.

Le RL classique de 2026. Les 50 articles réunissent les 14 articles `classic` du premier tirage et les 36 premiers articles `classic` de la file A3 (`extension_stats.json`, champ `samples`). Les 50 ont des expériences. Atteindre le trente-sixième article `classic` de la file a demandé 57 lectures, dont 18 articles `llm` et 3 articles `theory` lus en route malgré le tri sur le titre, puis 13 articles ont été lus après lui, soit 70 lectures.

« 50 contre 48 ». L'article de blog et nos figures parlent de « 50 articles de RL classique de 2026 » contre « 50 articles de 2018 ». Les deux échantillons comptent bien 50 articles lus. Les pourcentages de 2018 se calculent pourtant sur 48 articles, parfois 47, parce que deux articles de 2018 n'ont aucune expérience. Ceux de 2026 se calculent sur 50 articles, parfois 49 quand un critère ne s'applique pas. Le dénominateur exact de chaque part figure dans les tableaux de ce chapitre. Le libellé « 50 contre 50 » décrit donc les échantillons tirés, le libellé « 48 contre 50 » les échantillons analysés pour la plupart des critères.

4.2 La comparaison principale, à grille égale

Ce qui est comparé. La comparaison principale oppose nos 50 articles de RL classique de 2026 à notre lecture de 50 articles de 2018, avec les mêmes grilles, les mêmes règles harmonisées et les mêmes lecteurs (section 3.8). Le tableau 4.1 donne les critères de la grille v1, le tableau 4.2 ceux de la grille v2. Pour chaque critère : la part en 2018 et en 2026 avec leurs intervalles de Wilson, l'écart avec son intervalle de Newcombe, la valeur p du test exact de Fisher et la valeur ajustée par la correction de Holm, appliquée séparément aux 11 critères v1 et aux 12 critères v2. La figure 4.1 dessine ces écarts critère par critère. La figure 4.2 reprend les huit pratiques de l'article de blog.

Quatre pratiques ont changé. Quatre écarts passent la correction de Holm.

- **Le code en lien (Q5)** passe de 31 % (15 sur 48) à 74 % (37 sur 50) : $+43$ points, $[+23\,;\,+58]$, $p_{\text{Holm}} = 0{,}0004$.
- **La sensibilité à un hyperparamètre montrée (Y9)** passe de 44 % (21 sur 48) à 80 % (40 sur 50) : $+36$ points, $[+17\,;\,+52]$, $p_{\text{Holm}} = 0{,}004$.
- **Le calcul chiffré (X7)** passe de 27 % (13 sur 48) à 58 % (29 sur 50) : $+31$ points, $[+11\,;\,+47]$, $p_{\text{Holm}} = 0{,}02$.
- **Une incertitude rapportée (X2)** passe de 63 % (30 sur 48) à 88 % (44 sur 50) : $+26$ points, $[+8\,;\,+41]$, $p_{\text{Holm}} = 0{,}04$. C'est le verdict le plus fragile (section 4.4).

Un cinquième critère passe aussi la correction : l'emploi de réseaux de neurones (Q2), de 79 % (38 sur 48) à 98 % (49 sur 50), $+19$ points, $[+6\,;\,+32]$, $p_{\text{Holm}} = 0{,}03$. Q2 décrit la population et non une pratique de reproductibilité. L'écart dit que le RL classique de 2026 est presque entièrement du RL profond, ce qu'il n'était pas tout à fait en 2018. Le critère ne mesure aucune faille.

Deux intervalles qui évitent zéro sans passer la correction. La façon de choisir les hyperparamètres (Q6) passe de 26 % (12 sur 47) à 48 % (24 sur 50) : $+22$ points, $[+3\,;\,+39]$, $p = 0{,}03$ avant correction, $p_{\text{Holm}} = 0{,}24$. Les outils statistiques postérieurs à 2018 (Y12) passent de 2 % (1 sur 48) à 14 % (7 sur 50) : $+12$ points, $[+1\,;\,+24]$, $p_{\text{Holm}} = 0{,}66$. Pris seuls, ces deux intervalles évitent zéro. Avec 11 et 12 critères testés à la fois, ils ne passent pas le seuil ajusté. Nous ne détectons donc aucun changement après correction, sans pouvoir en exclure un.

Aucun changement détectable pour les autres critères. Les 16 autres critères ont des intervalles qui contiennent zéro. Parmi ceux qui mesurent une faille : les hyperparamètres de la méthode (Q3, 55 % puis 72 %), ceux des références (Q4, 35 % puis 29 %), des références réglées avec le même soin (X4, 13 % puis 16 %), la mesure définie (Y1, 79 % puis 78 %), les meilleurs essais gardés (X3, 19 % puis 14 %), le test statistique (Q8, 6 % puis 12 %), l'origine du code des références (Y5, 46 % puis 41 %), l'évaluation tenue à part (Q7, 33 % puis 45 %), le test de généralisation (Y8, 23 % puis 34 %), la version des bancs d'essai (Y7, 23 % puis 26 %), les versions des bibliothèques (Y10, 2 % puis 8 %).

La puissance du sondage. Chaque intervalle de Newcombe s'étend sur vingt à quarante points. Avec 50 articles de chaque côté, un changement de dix points peut encore passer inaperçu. « Aucun changement détectable » ne veut donc pas dire « aucun changement ». Pour Q3 par exemple, l'écart observé vaut $+17$ points, avec un intervalle

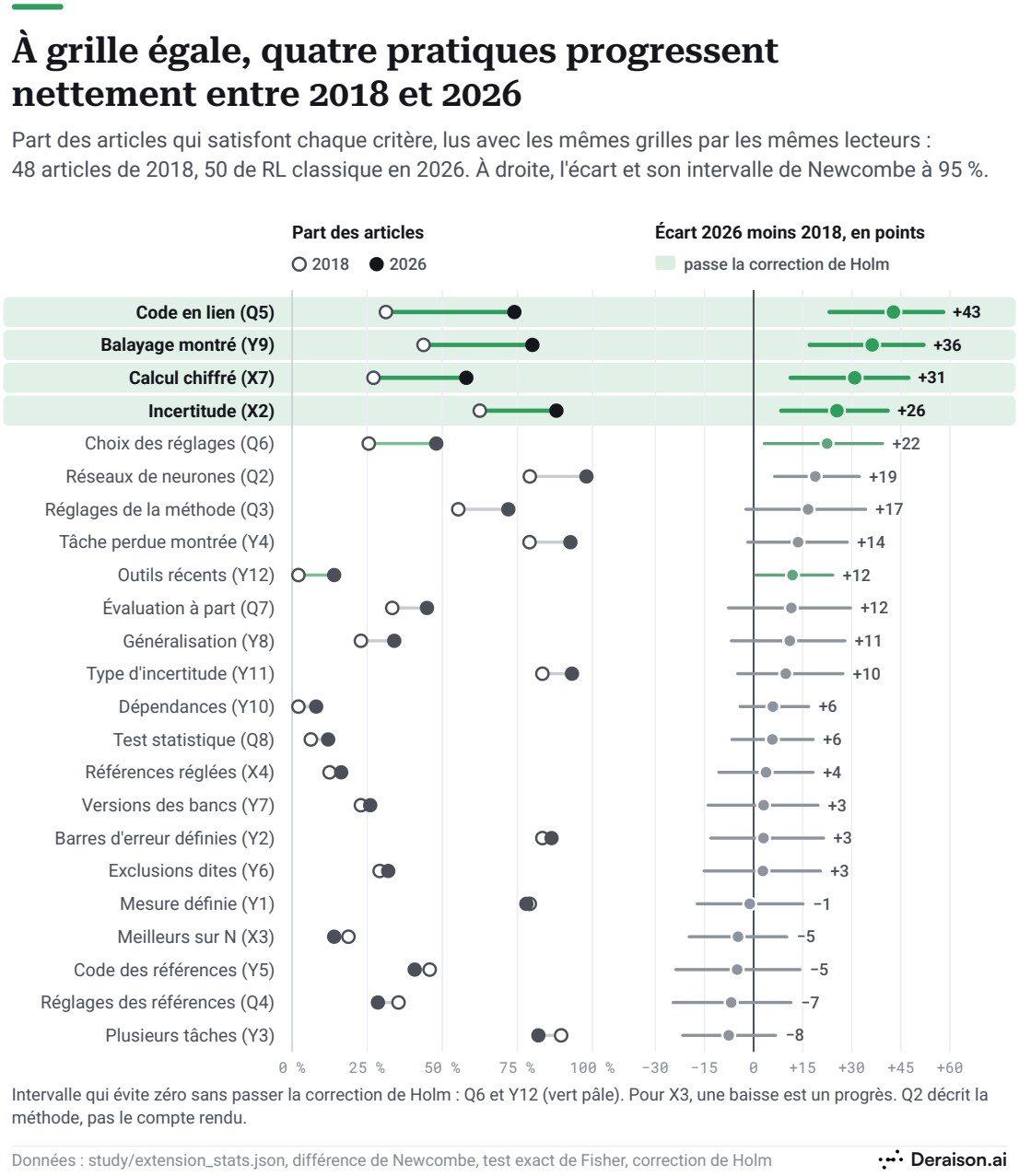


FIGURE 4.1. Tous les critères de lecture, grille v1 (Q, X) et grille v2 (Y), dans notre lecture de 2018 (48 articles expérimentaux) et dans nos 50 articles de RL classique de 2026, lus avec les mêmes grilles par les mêmes lecteurs. Pour chaque critère, les deux points donnent les deux parts, la barre l'intervalle de Newcombe à 95 % de leur écart. Les critères dont l'écart passe la correction de Holm sont mis en évidence. Pour X3, une part plus basse est un progrès. Données : `study/extension_stats.json`.

TABLE 4.1. Grille de Pineau (critères Q2 à Q8) et critères X tirés de sa conférence ICLR 2018, lus avec la même grille et les mêmes lecteurs sur deux échantillons : notre lecture de 50 articles de 2018 (ICML, NeurIPS, ICLR) et 50 articles de RL classique de 2026 (14 du premier tirage, 36 de la file A3). C'est la comparaison principale, la seule qui fonde un verdict (écart 20 de `study/PROTOCOL.md`). k/n : articles qui remplissent le critère sur les articles où il s'applique (articles à expériences, Q1 différent de « non ») ; la part en pourcentage porte son intervalle de Wilson à 95 %. Écart : RL classique 2026 moins 2018, en points de pourcentage, avec l'intervalle de Newcombe à 95 % ; p : test exact de Fisher bilatéral ; p Holm : correction de Holm sur les onze critères de la grille v1. **En gras**, les critères dont l'écart passe la correction de Holm (p Holm $< 0{,}05$). Pour X3, une part plus basse est un progrès ; Q2 (réseaux de neurones) décrit la population étudiée, pas une pratique de reproductibilité. Les chiffres de Pineau elle-même sont au tableau 4.4. Source : `study/extension_stats.json` (script `study/extension_stats.py`, écrit avant toute lecture des articles ajoutés).

		Notre lecture de 2018		RL classique 2026		Écart		
Code	Critère	k/n	% [Wilson]	k/n	% [Wilson]	[Newcombe]	p	p Holm
Q2	**Réseaux de neurones employés**	**38/48**	**79 [66 ; 88]**	**49/50**	**98 [90 ; 100]**	**+19 [+6 ; +32]**	**0,003**	**0,03**
Q3	Tous les hyperparamètres de la méthode	26/47	55 [41 ; 69]	36/50	72 [58 ; 83]	+17 [−2 ; +34]	0,10	0,58
Q4	Tous les hyperparamètres des références	17/48	35 [23 ; 50]	14/49	29 [18 ; 42]	−7 [−25 ; +11]	0,52	1
Q5	**Code en lien**	**15/48**	**31 [20 ; 45]**	**37/50**	**74 [60 ; 84]**	**+43 [+23 ; +58]**	**4×10^{-5}**	**0,0004**
Q6	Choix des hyperparamètres décrit	12/47	26 [15 ; 40]	24/50	48 [35 ; 61]	+22 [+3 ; +39]	0,03	0,24
Q7	Évaluation sur conditions tenues à part	16/48	33 [22 ; 47]	22/49	45 [32 ; 59]	+12 [−8 ; +30]	0,30	1
Q8	Test statistique appliqué	3/48	6 [2 ; 17]	6/50	12 [6 ; 24]	+6 [−7 ; +18]	0,49	1
X2	**Incertitude rapportée**	**30/48**	**63 [48 ; 75]**	**44/50**	**88 [76 ; 94]**	**+26 [+8 ; +41]**	**0,005**	**0,04**
X3	Seuls les meilleurs essais gardés	9/48	19 [10 ; 32]	7/50	14 [7 ; 26]	−5 [−20 ; +10]	0,59	1
X4	Références réglées avec le même soin	6/48	13 [6 ; 25]	8/49	16 [9 ; 29]	+4 [−11 ; +18]	0,77	1
X7	**Calcul chiffré**	**13/48**	**27 [17 ; 41]**	**29/50**	**58 [44 ; 71]**	**+31 [+11 ; +47]**	**0,002**	**0,02**

de -2 à $+34$: la donnée reste compatible avec une hausse de trente points comme avec une stagnation.

4.3 Relecture faille par faille

Cette section reprend les résultats dans l'ordre des failles du tableau 2.2. Les chiffres de 2018 et de 2026 sont ceux de la comparaison principale, sauf mention du premier tirage.

F1, le code. Le lien vers le code passe de 31 % à 74 % des articles. La faille est écoutée. Le bilan officiel du programme de reproductibilité de NeurIPS 2019 va dans le même sens : la part d'articles acceptés avec du code passe de moins de 50 % à 74,4 % (Pineau et al. 2021). Dans tout le premier tirage, grands modèles de langue compris, la part vaut 67 % (30 sur 45). Un article de RL classique sur quatre publie encore son résultat sans son code.

Table 4.2. Grille v2 (critères Y1 à Y12, écarts 11 et 14 de `study/PROTOCOL.md`, écrite le 9 novembre 2026 avant toute lecture v2), sur les mêmes échantillons que le tableau 4.1 : 50 articles de 2018 lus par nous et 50 articles de RL classique de 2026. Mêmes colonnes : k/n sur les articles où le critère s'applique, part avec intervalle de Wilson à 95 %, écart en points avec l'intervalle de Newcombe à 95 %, test exact de Fisher, correction de Holm sur les douze critères de la grille v2 (famille distincte de la grille v1). Y3 vaut « oui » quand la conclusion principale porte sur au moins deux tâches ; Y2 et Y11 ne s'appliquent qu'aux articles qui rapportent une incertitude, Y4 qu'aux articles à plusieurs tâches. **En gras**, les critères dont l'écart passe la correction de Holm : seul Y9. Valeurs harmonisées (`study/overrides_v2.json`). Source : `study/extension_stats.json`.

		Notre lecture de 2018		RL classique 2026		Écart		
Code	Critère	k/n	% [Wilson]	k/n	% [Wilson]	[Newcombe]	p	p Holm
Y1	Mesure de performance définie	38/48	79 [66 ; 88]	39/50	78 [65 ; 87]	−1 [−17 ; +15]	1	1
Y2	Barres d'erreur définies	25/30	83 [66 ; 93]	38/44	86 [73 ; 94]	+3 [−13 ; +21]	0,75	1
Y3	Plusieurs tâches (au moins 2)	43/48	90 [78 ; 95]	41/50	82 [69 ; 90]	−8 [−22 ; +7]	0,39	1
Y4	Une tâche perdue montrée	34/43	79 [65 ; 89]	38/41	93 [81 ; 97]	+14 [−2 ; +29]	0,12	1
Y5	Origine du code des références	22/48	46 [33 ; 60]	20/49	41 [28 ; 55]	−5 [−24 ; +14]	0,68	1
Y6	Exclusions dites	14/48	29 [18 ; 43]	16/50	32 [21 ; 46]	+3 [−15 ; +20]	0,83	1
Y7	Version des bancs d'essai	11/48	23 [13 ; 37]	13/50	26 [16 ; 40]	+3 [−14 ; +20]	0,82	1
Y8	Test de généralisation	11/48	23 [13 ; 37]	17/50	34 [22 ; 48]	+11 [−7 ; +28]	0,27	1
Y9	**Sensibilité à un hyperparamètre**	**21/48**	**44 [31 ; 58]**	**40/50**	**80 [67 ; 89]**	**+36 [+17 ; +52]**	**0,0003**	**0,004**
Y10	Versions des bibliothèques	1/48	2 [0 ; 11]	4/50	8 [3 ; 19]	+6 [−4 ; +17]	0,36	1
Y11	Type d'incertitude nommé	25/30	83 [66 ; 93]	41/44	93 [82 ; 98]	+10 [−5 ; +27]	0,26	1
Y12	Outil statistique postérieur à 2018	1/48	2 [0 ; 11]	7/50	14 [7 ; 26]	+12 [+1 ; +24]	0,06	0,66

F2 et F3, le plan pour refaire et le matériel. Le calcul est chiffré dans 58 % des articles de 2026 contre 27 % en 2018. Les hyperparamètres de la méthode sont tous donnés dans 72 % des articles contre 55 %, sans changement détectable ($p_{\text{Holm}} = 0{,}58$). Les versions des bibliothèques restent rares : 4 articles de RL classique sur 50 les donnent en 2026, contre 1 sur 48 en 2018 (Y10). Dans tout le premier tirage, un seul article sur 45 les donne. Les deux failles sont écoutées en partie : le calcul se chiffre, les versions des logiciels se taisent.

F4 et F5, les hyperparamètres de la méthode et leur choix. Le niveau de Q3 ne bouge pas de façon détectable. La façon de choisir les hyperparamètres est décrite dans 48 % des articles de 2026 contre 26 % en 2018, un écart qui ne passe pas la correction. Moins d'un article sur deux dit encore comment il a trouvé ses réglages.

F6, la sensibilité aux réglages. Au moins un hyperparamètre de la méthode est balayé, résultat à l'appui, dans 80 % des articles de 2026 contre 44 % en 2018. La faille est écoutée. Le premier tirage, grands modèles compris, donne un niveau voisin : 32 articles sur 45.

F7, les références. Les hyperparamètres des références sont tous donnés dans 29 % des articles de 2026 (14 sur 49), contre 35 % en 2018 (17 sur 48). Les articles qui disent avoir réglé leurs références avec le même soin que leur méthode passent de 13 % à 16 %. La pratique reste rare les deux années, sans changement détectable. Contre les 60 % de

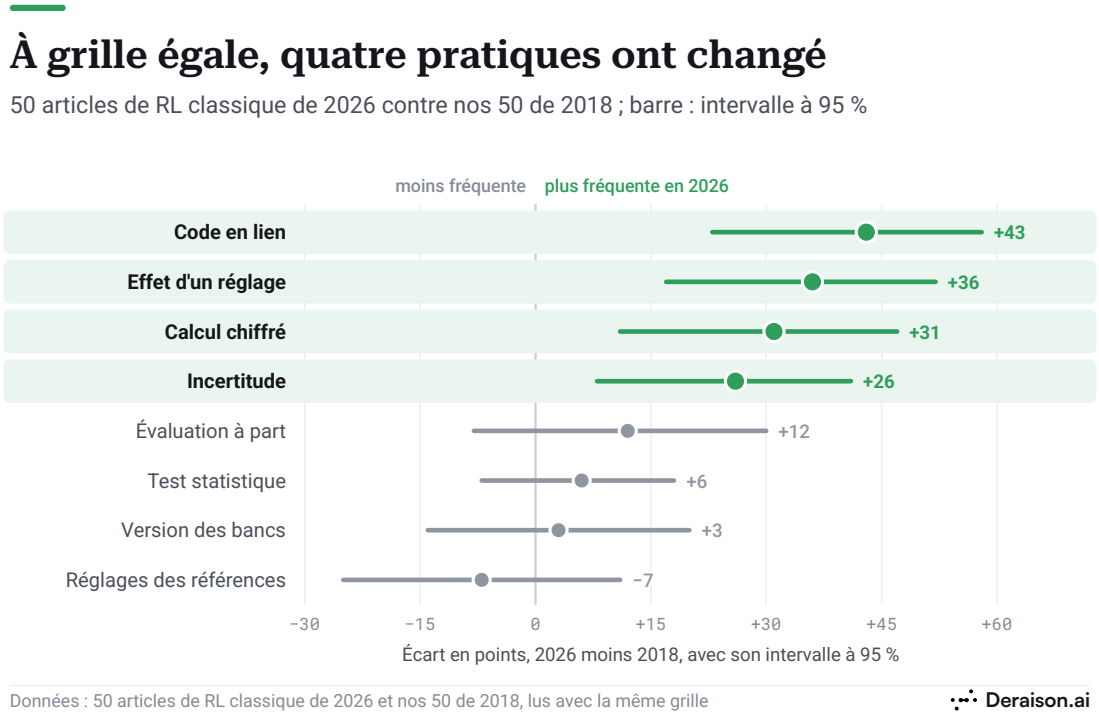


FIGURE 4.2. Écart, en points de pourcentage, entre 50 articles de RL classique de 2026 et 50 articles de 2018 (48 expérimentaux), lus avec les mêmes grilles par les mêmes lecteurs, pour les huit pratiques de l'article de blog. Le point donne l'écart observé, la barre son intervalle de Newcombe à 95 %. Les quatre pratiques en vert passent la correction de Holm : le lien vers le code (Q5), l'effet d'un réglage montré (Y9), le calcul chiffré (X7), une incertitude rapportée (X2). Les quatre autres, dont le test statistique (Q8), ont des intervalles qui contiennent zéro. Données : `study/extension_stats.json`.

Pineau, la baisse paraissait nette (section 4.5). Cette baisse apparente vient pour une part de notre grille, qui trouve 35 % en 2018, pour l'autre des articles sur les modèles de langue, qui décrivent rarement leurs références : tout le premier tirage tombe à 18 % (8 sur 44).

F8, la mesure de performance. La mesure est définie dans 78 % des articles de 2026 contre 79 % en 2018 (Y1). Quand des barres d'erreur figurent, leur sens est défini dans 38 articles sur 44 en 2026 contre 25 sur 30 en 2018 (Y2). La mesure est le plus souvent définie, les deux années, sans changement détectable. Y1 reste notre critère le plus fragile : sur le premier tirage, deux lecteurs ne s'accordaient que sur 60 % des articles avant que ses règles soient précisées (section 4.8).

F9, le nombre d'essais. Sur les 41 articles de RL expérimentaux du premier tirage (familles `llm` et `classic`), 20 ne disent pas combien d'entraînements indépendants soutiennent leurs résultats (X1). Quatre n'en ont qu'un. La médiane des autres vaut cinq, la valeur la plus fréquente du tableau que Pineau montrait en 2018. La faille n'est pas écoutée. Le chapitre 6 mesure avec l'expérience E5 ce que cinq graines permettent de trancher.

F10, les meilleurs essais. La sélection des meilleurs essais se voit dans 14 % des articles de 2026 contre 19 % en 2018 (X3), sans changement détectable. Dans le premier tirage, 2

articles sur 45 la laissent voir. Un article qui garde ses meilleurs essais sans le dire ni le montrer échappe à notre lecture : ces parts sont des minimums.

F12, l'incertitude et le test. La faille se coupe en deux. Rapporter une incertitude est devenu courant : 88 % des articles de 2026 contre 63 % en 2018. Le type d'incertitude est nommé dans 41 des 44 articles de 2026 qui en rapportent une, contre 25 sur 30 en 2018 (Y11). Le test statistique n'a pas bougé de façon détectable : 12 % des articles de 2026 contre 6 % en 2018 (Q8). Les outils apparus après 2018, comme l'IQM ou la bibliothèque rliable, passent de 2 % à 14 % (Y12), sans passer la correction. Les articles tracent des barres, ils testent rarement.

F13 et F14, les tâches et les implémentations. La conclusion principale repose sur plusieurs tâches dans 41 articles sur 50 en 2026, contre 43 sur 48 en 2018 (Y3). Parmi les articles à plusieurs tâches, 38 sur 41 montrent au moins une tâche où leur méthode ne gagne pas, contre 34 sur 43 en 2018 (Y4). L'origine du code des références est dite dans 41 % des articles de 2026 contre 46 % en 2018 (Y5). Aucun des trois critères ne change de façon détectable. Les deux premières pratiques étaient déjà courantes en 2018, la troisième reste minoritaire. Dans le premier tirage, 25 articles sur 45 seulement comptent plusieurs tâches : beaucoup d'articles sur les modèles de langue s'entraînent sur un seul jeu de données.

F15, l'évaluation tenue à part. À grille égale, le RL classique passe de 33 % à 45 % pour Q7 et de 23 % à 34 % pour le test de généralisation, plus exigeant (Y8). Aucun des deux écarts n'est détectable. La forte hausse apparente contre les 10 % de Pineau vient de la grille et du changement de terrain (sections 4.5 et 4.6).

F16, les bancs d'essai. La version de chaque banc d'essai est donnée dans 26 % des articles de 2026 contre 23 % en 2018 (Y7), sans changement détectable. Dans le premier tirage, 16 articles sur 27 de la famille `llm` n'essaient qu'une seule famille de modèles de langue (X6).

4.4 Ce qui résiste aux analyses de robustesse

Le tableau 4.3 rassemble chaque analyse alternative et dit, pour chaque verdict, s'il la traverse. Les paragraphes qui suivent les détaillent.

La provenance des articles de 2026 (a1). Au sein de 2026, les 14 articles du premier tirage et les 36 de la file A3 ne diffèrent sur aucun des quatre critères (Fisher, $p = 0{,}73$ pour Q5, 1 pour Y9 et X7, $0{,}17$ pour X2). Refaite avec chaque provenance seule contre 2018, la comparaison garde son signe et un intervalle qui exclut zéro dans les huit cas. Pour Q5 : $+47$ points $[+18\,;\,+65]$ avec le premier tirage, $+41$ $[+20\,;\,+57]$ avec la file. Pour X2, l'intervalle de la file seule s'approche de zéro : $+21$ points $[+1\,;\,+38]$.

La composition par conférence (a3). Le test de Cochran-Mantel-Haenszel, stratifié par conférence, garde les quatre verdicts : $p = 5 \times 10^{-5}$ pour Q5, $0{,}001$ pour Y9, $0{,}005$ pour X7, $0{,}006$ pour X2. Il ne soutient pas Q6 ($p = 0{,}06$) ni Y12 ($p = 0{,}15$).

TABLE 4.3. Contrôles de robustesse du sondage, tous exploratoires (écarts 21, 27 et 29 de `study/PROTOCOL.md`) : ils disent si un verdict est fragile, sans en changer aucun. Colonnes : les quatre critères qui ont changé (Q5, Y9, X7, X2) et les deux dont l'intervalle évite zéro sans passer Holm (Q6, Y12). Écarts en points (RL classique 2026 moins notre lecture de 2018), intervalle de Newcombe à 95 %, test de Fisher, Holm sur la famille de la grille du critère comme dans la comparaison principale. (a1) le RL classique de 2026 coupé selon sa provenance (premier tirage sans tri sur le titre, file A3 avec tri), puis p de Fisher entre les deux provenances au sein de 2026 ; (a2) p de Fisher entre articles de 2026 lus sur arXiv (45) et sur le site d'une conférence (5), une coupe trop déséquilibrée pour trancher (écart 21 c) ; (a3) test de Cochran-Mantel-Haenszel stratifié par conférence (correction de Haldane) ; (a4) comparaison refaite sans chaque lecteur principal tour à tour : étendue des écarts et plus basse borne inférieure ; un critère est fragile quand l'écart change de signe ou que l'intervalle contient zéro avec au moins 35 articles de chaque côté. Partie basse : sans les treize articles lus en partie par `claude-opus-5` (écart 29) ; avec les 10 décisions « non » qu'un arbitre à l'aveugle a retournées après la recherche par mots-clés, puis avec aussi les 24 décisions jugées douteuses, sur 377 décisions « non » examinées (écart 27). **En gras**, les écarts dont l'intervalle exclut zéro et les p sous 0,05 (sauf les p d'homogénéité de (a1) et (a2), où un p élevé est attendu). La thèse « écoutée là où cela coûtait peu » est rejetée par le test à l'aveugle (b) : corrélation de Spearman entre l'écart observé et le coût médian noté par trois évaluateurs +0,16 (p par permutation 0,52), avec le risque −0,04 (p = 0,89). Sources : `study/robustness.json`, `study/extension_stats_sans_opus5.json`, `study/extension_stats_falsify_turned.json`, `study/extension_stats_falsify_bound.json`, `study/falsify_no.json`.

Contrôle	Q5	Y9	X7	X2	Q6	Y12
Comparaison principale : écart [IC]	**+43 [+23 ; +58]**	**+36 [+17 ; +52]**	**+31 [+11 ; +47]**	**+26 [+8 ; +41]**	**+22 [+3 ; +39]**	**+12 [+1 ; +24]**
p Holm	**0,0004**	**0,004**	**0,02**	**0,04**	0,24	0,66
(a1) premier tirage seul (14)	**+47 [+18 ; +65]**	**+35 [+5 ; +54]**	**+30 [+2 ; +54]**	**+38 [+13 ; +52]**	+17 [−8 ; +44]	+12 [−1 ; +38]
(a1) file A3 seule (36)	**+41 [+19 ; +57]**	**+37 [+16 ; +53]**	**+31 [+10 ; +49]**	**+21 [+1 ; +37]**	**+24 [+4 ; +43]**	**+12 [0 ; +27]**
(a1) premier tirage contre A3 : p	0,73	1	1	0,17	0,76	1
(a2) arXiv contre site : p	1	1	0,15	1	1	1
(a3) Mantel-Haenszel : rapport de cotes	6,3	4,4	3,5	3,8	2,4	3,6
p	**5×10^{-5}**	**0,001**	**0,005**	**0,006**	0,06	0,15
(a4) un lecteur retiré : écarts	+37 à +50	+30 à +43	+27 à +37	+21 à +31	+18 à +25	+9 à +14
(a4) plus basse borne de l'IC	**+16**	**+10**	**+7**	**+4**	−3	−2
Fragile selon (a)	non	non	non	non	oui (1)	oui (8)
Sans les 13 articles opus-5	**+46 [+26 ; +61]**	**+35 [+16 ; +51]**	**+31 [+11 ; +48]**	**+25 [+8 ; +41]**	**+22 [+3 ; +40]**	**+13 [+1 ; +25]**
p Holm	**0,0001**	**0,008**	**0,04**	0,07	0,23	0,65
« Non » démentis retournés : p Holm	**0,001**	**0,008**	**0,003**	**0,04**	0,24	0,66
et douteux aussi : p Holm	**0,002**	0,05	**0,005**	0,07	0,24	0,66

La source du PDF (a2). Cette coupe oppose 45 articles de 2026 lus sur arXiv à 5 lus sur le site d'une conférence. Elle ne peut rien trancher. Le rapport ne s'appuie pas sur elle. La différence de source reste une limite : une version arXiv est parfois plus longue ou plus récente que celle des actes, alors que les articles de 2018 ont été lus dans les actes.

Un lecteur retiré à la fois (a4). Retirer tour à tour chacun des seize lecteurs principaux ne rend fragile aucun des quatre verdicts. La borne basse la plus faible de l'intervalle de Newcombe vaut $+16$ points pour Q5 (sans H1 ou sans H3), $+10$ pour Y9 (sans H2), $+7$ pour X7 (sans E, K2 ou K5) et $+4$ pour X2 (sans H4). Q6 devient fragile sans le lecteur H2 ($+18$ points, $[-3\,;\,+36]$). Y12 est fragile dans 8 des 16 coupes.

Sans les treize articles lus en partie par un autre modèle. Sans les treize articles finis par `claude-opus-5` (écart 29), l'échantillon de RL classique de 2026 passe à 48 articles et celui de 2018 reste à 50. Le code (Q5, $p_{\text{Holm}} = 0{,}0001$), la sensibilité (Y9, $0{,}008$) et le calcul (X7, $0{,}04$) passent toujours la correction. L'incertitude rapportée ne la passe plus : $+25$ points, $[+8\,;\,+41]$, $p_{\text{Holm}} = 0{,}07$. Q2 ne la passe plus non plus ($p_{\text{Holm}} = 0{,}07$). Les tendances de la trajectoire ne changent pas (section 4.7).

La recherche contre les « non ». Sur les sept critères vérifiés, les deux échantillons comptent 377 décisions finales no. Pour 94, les motifs fixés d'avance ne trouvent aucune occurrence dans le texte. Les 283 autres ont été soumises à l'arbitre. L'arbitre en retourne 10 : 2 en 2018 (C19 pour Q5, C120 pour Y9) et 8 en 2026 (K058, K063 et LR03 pour Q4, K002 pour Q7, ML06 pour Q8, K056, LR03 et LR09 pour X7). Il en juge 24 douteuses. Refaite avec les décisions retournées, la comparaison principale garde tous ses verdicts (`study/falsify_no.json`). L'écart du calcul chiffré grandit, à $+37$ points $[+17\,;\,+53]$. Celui des hyperparamètres des références tombe à -1 point. Si l'on retourne aussi les 24 décisions douteuses, l'incertitude rapportée et la sensibilité passent juste au-dessus du seuil ($p_{\text{Holm}} = 0{,}07$ et $0{,}054$). Cette seconde borne n'était pas annoncée dans l'écart 27.

Bilan de robustesse. Le code et le calcul passent le seuil dans tous nos contrôles. La sensibilité aux réglages le passe dans tous sauf la borne des décisions douteuses, où elle s'arrête à $p_{\text{Holm}} = 0{,}054$. L'incertitude rapportée échoue dans deux contrôles, sans les treize articles et avec la borne des décisions douteuses, avec $p_{\text{Holm}} = 0{,}07$ chaque fois. Le rapport la garde parmi les quatre changements, avec cette réserve.

Le coût et le risque de chaque pratique. Les trois évaluateurs s'accordent entre eux : leurs notes se corrèlent de 0,78 à 0,91 pour le coût, de 0,90 à 0,93 pour le risque (Spearman, par paire). Leurs notes ne prédisent pas les écarts observés. La corrélation de Spearman vaut $+0{,}16$ pour le coût ($p = 0{,}52$ par permutation) et $-0{,}04$ pour le risque ($p = 0{,}89$). Le balayage d'un hyperparamètre (Y9), noté coûteux (médiane 4), a progressé. Les versions des bancs d'essai (Y7) et des dépendances (Y10), notées presque gratuites (médiane 1), n'ont pas bougé. La lecture « écoutée là où cela coûtait peu » est retirée.

4.5 Notre lecture de 2018 contre les chiffres de Pineau

L'écart. La comparaison secondaire de l'écart 12 oppose notre lecture de 2018 aux chiffres de Pineau, sur les six critères Q3 à Q8 où elle en donne un comparable, avec une correction de Holm sur ces six critères (`extension_stats.json`, champ `ours2018_vs_pineau`).

- **Q3, hyperparamètres de la méthode** : 55 % (26 sur 47) chez nous, 90 % chez Pineau. Écart de -35 points, $[-50\,;\,-17]$, $p_{\text{Holm}} = 0{,}001$.
- **Q7, évaluation tenue à part** : 33 % (16 sur 48) chez nous, 10 % chez Pineau. Écart de $+23$ points, $[+7\,;\,+39]$, $p_{\text{Holm}} = 0{,}03$.
- **Q4, hyperparamètres des références** : 35 % contre 60 %. Écart de -25 points, $[-42\,;\,-5]$, $p_{\text{Holm}} = 0{,}06$.
- **Q5, code en lien** : 31 % contre 55 %. Écart de -25 points, $[-42\,;\,-5]$, $p_{\text{Holm}} = 0{,}06$.
- **Q6 et Q8** : 26 % contre 20 % et 6 % contre 5 %, sans écart détectable.

Les 12 premiers articles de la calibration donnaient déjà le même signe : Q3 à 6 sur 10, Q7 à 4 sur 11 (`results_final.json`). Notre grille est plus sévère que celle de Pineau sur ce que l'article donne, plus large sur ce qui compte comme tenu à part.

Pourquoi notre lecture diffère. Le tirage de Pineau et sa façon de trancher les cas douteux restent inconnus : la diapositive annonce 50 articles de RL de 2018 publiés à NeurIPS, ICML et ICLR, sans dire comment ils ont été choisis ni comment chaque critère a été décidé. Nous ne pouvons donc pas attribuer l'écart à une cause unique. Trois mécanismes le rendent plausible, sans qu'aucun soit démontré.

- **Des règles écrites plus strictes sur ce qui est donné.** Nos règles de Q3 et Q4 exigent les hyperparamètres de toutes les méthodes principales. Des « valeurs par défaut » sans liste ne comptent que si la version de la bibliothèque est donnée. Pour Q5, « sera publié » et une page de projet qui ne mène pas au code valent no. Une lectrice qui accepte un tableau partiel ou une promesse de publication trouve des parts plus hautes.
- **Une règle plus large sur ce qui est tenu à part.** Notre Q7 compte des graines d'environnement non vues à l'entraînement. Pineau pensait sans doute à une séparation plus nette, puisque sa feuille de route de NeurIPS demande de séparer les tâches et de passer à des simulateurs plus riches.
- **Un autre tirage.** Notre échantillon de 2018 est tiré au hasard parmi les 131 articles dont le titre correspond à notre expression. Celui de Pineau peut contenir des articles que notre filtre ne voit pas, ou en exclure.

Après correction, les écarts de Q4 et de Q5 restent compatibles avec le hasard ($p_{\text{Holm}} = 0{,}06$). Ceux de Q3 et de Q7 ne le sont pas.

La règle de verdict qui en découle. Comparer nos articles de 2026 aux chiffres de Pineau mêlerait le changement des pratiques et celui de la grille. L'écart 20 en tire la règle de verdict (section 3.8). Les chiffres de Pineau restent montrés comme contexte, à la section 4.6.

4.6 Le premier tirage et les modèles de langue

Le premier tirage contre Pineau. La figure 4.3 compare les 45 articles expérimentaux du premier tirage, grands modèles de langue compris, aux chiffres de Pineau. La figure 4.4 dessine les écarts avec leurs intervalles de Newcombe, la figure 4.5 les parts de 2026 avec leurs intervalles de Wilson. Trois intervalles s'écartent nettement de zéro. Les

Table 4.4. Les chiffres de Pineau (2018), comme contexte et jamais comme verdict : ses parts sur 50 articles de RL de 2018, pour les six critères qu'elle chiffre (pourcentage publié, `study/results_final.json` ; entre parenthèses le k/n des tests, son pourcentage multiplié par 50 et arrondi, soit 56 % pour Q5 et 4 % pour Q8), comparées à notre lecture de 50 articles de 2018 avec notre grille, puis aux 45 articles expérimentaux du premier tirage de 2026, grands modèles de langage compris. La première ligne de chaque critère mesure l'écart entre notre grille et la sienne sur la même année : notre lecture est plus sévère sur Q3 et plus large sur Q7, ce qui a conduit à ne fonder les verdicts que sur la comparaison avec la même grille (écart 20 de `study/PROTOCOL.md`, tableau 4.1). La seconde mêle le changement des pratiques, celui de la grille et celui de la population. Écart en points avec l'intervalle de Newcombe à 95 %, test exact de Fisher, correction de Holm sur Q3 à Q8 dans chaque famille de comparaisons. **En gras**, ce qui passe la correction de Holm. Le tirage de Pineau reste inconnu. Sources : `study/extension_stats.json`, `study/results_final.json`.

Code	Critère, puis comparaison	k/n	%	Pineau 2018	Écart [Newcombe]	p	p Holm
Q3	Tous les hyperparamètres de la méthode						
	notre lecture de 2018	26/47	55	90 (45/50)	**−35 [−50 ; −17]**	**0,0002**	**0,001**
	premier tirage 2026	30/45	67		**−23 [−39 ; −7]**	**0,006**	**0,02**
Q4	Tous les hyperparamètres des références						
	notre lecture de 2018	17/48	35	60 (30/50)	−25 [−42 ; −5]	0,02	0,06
	premier tirage 2026	8/44	18		**−42 [−57 ; −22]**	**5×10^{-5}**	**0,0002**
Q5	Code en lien						
	notre lecture de 2018	15/48	31	55 (28/50)	−25 [−42 ; −5]	0,02	0,06
	premier tirage 2026	30/45	67		+11 [−9 ; +29]	0,30	0,91
Q6	Choix des hyperparamètres décrit						
	notre lecture de 2018	12/47	26	20 (10/50)	+6 [−11 ; +22]	0,63	1
	premier tirage 2026	12/45	27		+7 [−10 ; +23]	0,47	0,95
Q7	Évaluation sur conditions tenues à part						
	notre lecture de 2018	16/48	33	10 (5/50)	**+23 [+7 ; +39]**	**0,006**	**0,03**
	premier tirage 2026	31/44	70		**+60 [+42 ; +73]**	**1×10^{-9}**	**6×10^{-9}**
Q8	Test statistique appliqué						
	notre lecture de 2018	3/48	6	5 (2/50)	+2 [−8 ; +13]	0,67	1
	premier tirage 2026	2/45	4		0 [−10 ; +11]	1	1

hyperparamètres de la méthode baissent, de 90 % à 67 % (-23 points, $[-39\,;\,-7]$, $p_{\text{Holm}} = 0{,}02$). Ceux des références baissent, de 60 % à 18 % (-42 points, $[-57\,;\,-22]$, $p_{\text{Holm}} = 0{,}0002$). L'évaluation tenue à part monte, de 10 % à 70 % ($+60$ points, $[+42\,;\,+73]$). Le code passe de 55 % à 67 % sans écart détectable. Le test statistique passe de 5 % à 4 %. Ces écarts mêlent deux changements, celui des pratiques et celui de la grille. Ils servent de contexte, jamais de verdict.

Les articles sur les modèles de langue. Le tableau 4.5 oppose, dans le premier tirage, les 27 articles de la famille `llm` aux 14 articles de RL classique. Les différences sont fortes. L'évaluation tenue à part est presque universelle dans la famille `llm` (26 sur 27) et minoritaire dans le RL classique du premier tirage (5 sur 13) : les recueils de problèmes de mathématiques séparent par construction les problèmes d'entraînement et de test. L'incertitude est rare dans la famille `llm` (4 sur 27) et universelle dans le RL classique du premier tirage (14 sur 14). Les hyperparamètres des références sont tous donnés dans 5 articles `llm` sur 27. Seize articles `llm` sur 27 n'essaient qu'une famille de modèles de base. La hausse apparente de l'évaluation tenue à part contre Pineau vient presque entière de ces articles. La baisse apparente de l'incertitude dans le premier tirage aussi.



FIGURE 4.3. La grille de Pineau, appliquée en 2018 et en 2026, à titre de contexte. Chaque ligne est un critère de lecture. Le point de 2018 donne la part des 50 articles lus par Pineau qui satisfont le critère. Le point de 2026 donne la part dans les 45 articles expérimentaux de notre premier tirage d'ICLR 2026, d'ICML 2026 et de NeurIPS 2025, grands modèles de langue compris. Les deux points ne viennent pas de la même grille : notre lecture de 2018 trouve 55 % pour les hyperparamètres de la méthode, là où Pineau trouvait 90 %. Le test statistique, absent du dessin, passe de 5 % à 4 %. Données : `study/audit_stats.json`.

Un changement de population. La famille `llm` n'existait pas en 2018. Un écart entre le premier tirage de 2026 et 2018 qui passe par ces articles mesure d'abord l'arrivée d'une population nouvelle, avec ses propres habitudes. Il ne dit rien du changement des pratiques au sein d'une population restée comparable. Les verdicts de ce rapport valent donc pour le RL classique seul.

4.7 La trajectoire de 2018 à 2026

Les parts année par année. Le tableau 4.6 et la figure 4.6 donnent, pour chaque pratique, la part des articles de RL classique qui la suivent en 2018 (48 articles), en 2019, 2020, 2022 et 2024 (20 articles par année, ICML et NeurIPS) et en 2026 (50 articles). Le code en lien passe de 31 % en 2018 à 60 %, 50 %, 70 % et 75 % de 2019 à 2024, puis 74 % en 2026. La sensibilité à un hyperparamètre vaut 44 %, puis 35 %, 45 %, 60 %, 85 % et 80 %. Le calcul chiffré vaut 27 %, puis 30 %, 40 %, 40 %, 55 % et 58 %. Une incertitude rapportée vaut 63 %, puis 80 %, 85 %, 95 %, 100 % et 88 %. Le test statistique vaut 6 %, puis 5 %, 10 %, 10 %, 25 % et 12 %. Avec 20 articles, une part a une incertitude d'environ vingt points de part et d'autre.

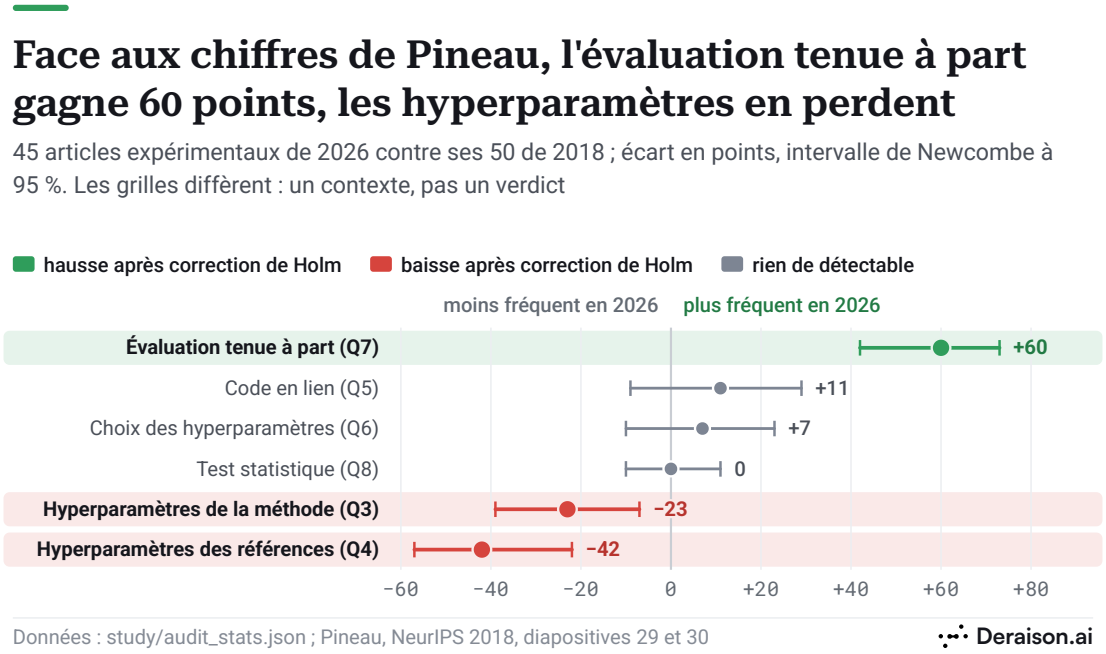


FIGURE 4.4. Écart entre la part des 45 articles expérimentaux de notre premier tirage de 2026 et celle des 50 articles de 2018 lus par Pineau, critère par critère, en points de pourcentage. Le point est l'écart observé, la barre son intervalle de Newcombe à 95 %, qui tient compte de l'incertitude des deux échantillons. Au-dessus de zéro, la pratique est plus fréquente en 2026. L'écart mêle le changement des pratiques et la différence entre notre grille et celle de Pineau : il sert de contexte, les verdicts viennent de la comparaison à grille égale. Données : `study/audit_stats.json`.

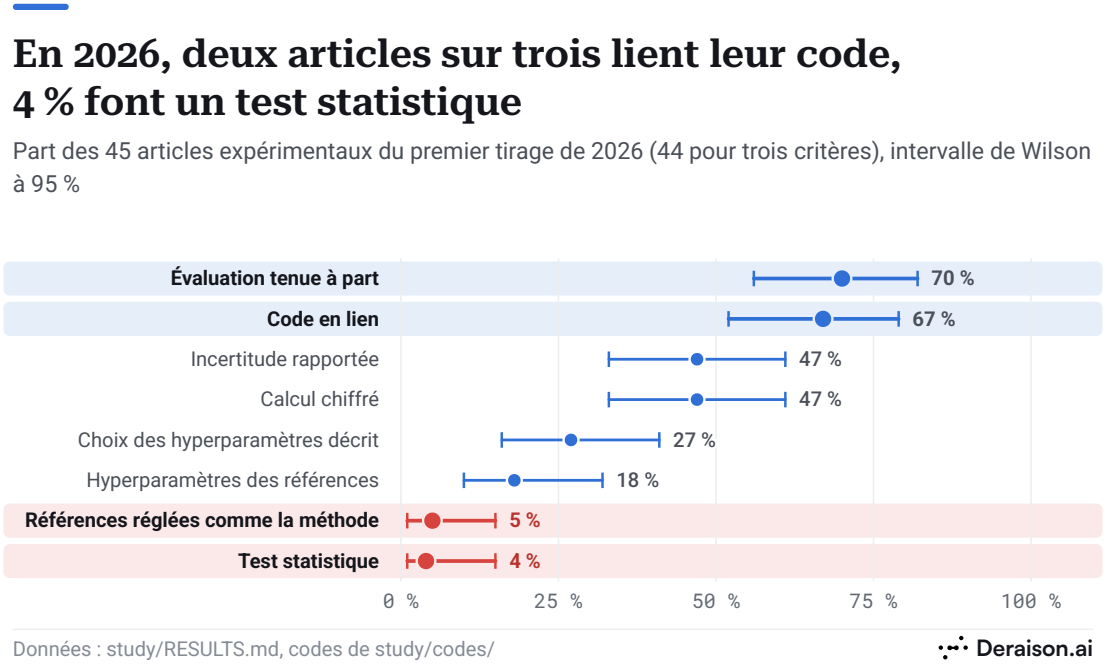


FIGURE 4.5. Part des 45 articles expérimentaux du premier tirage de 2026 qui satisfont chaque critère, avec son intervalle de Wilson à 95 %. Le tirage mêle RL classique et articles sur les grands modèles de langue. Avec un échantillon de cette taille, un écart de moins de quinze points avec 2018 se confond avec le hasard du tirage. Données : `study/results_final.json`.

TABLE 4.5. RL appliqué aux grands modèles de langage (famille `llm` du critère X5) contre RL classique, dans le premier tirage aléatoire de 50 articles de 2026 (ICLR 2026, ICML 2026, NeurIPS 2025) : 27 articles `llm`, 14 `classic`, 9 `theory` (non montrés, comptés dans « Tous »). k/n sur les articles à expériences où le critère s'applique, part en pourcentage entre parenthèses ; intervalle de Wilson à 95 % pour la colonne « Tous », seul calculé par le script. Seule la grille v1 a ce découpage par famille dans les résultats (`study/audit_stats.json`, harmonisation de `study/overrides.json`) : la grille v2 n'est pas ventilée par famille, et aucun test entre familles n'a été préregistré ; le tableau est descriptif. Les deux dernières colonnes rappellent l'échantillon classique élargi à 50 articles (tableau 4.1). Les 18 articles `llm` et 3 articles `theory` lus dans la file A3 restent à part : le tri sur le titre en avait déjà écarté beaucoup, ils ne forment pas un échantillon. Sources : `study/audit_stats.json`, `study/extension_stats.json`.

		Premier tirage 2026 (grille v1)				RL classique 2026 élargi	
Code	Critère	RL sur grands modèles	RL classique	Tous	% tous [Wilson]	k/n	%
Q3	Tous les hyperparamètres de la méthode	17/27 (63)	11/14 (79)	30/45	67 [52 ; 79]	36/50	72
Q4	Tous les hyperparamètres des références	5/27 (19)	2/14 (14)	8/44	18 [10 ; 32]	14/49	29
Q5	Code en lien	19/27 (70)	11/14 (79)	30/45	67 [52 ; 79]	37/50	74
Q6	Choix des hyperparamètres décrit	6/27 (22)	6/14 (43)	12/45	27 [16 ; 41]	24/50	48
Q7	Évaluation sur conditions tenues à part	26/27 (96)	5/13 (38)	31/44	70 [56 ; 82]	22/49	45
Q8	Test statistique appliqué	2/27 (7)	0/14 (0)	2/45	4 [1 ; 15]	6/50	12
X2	Incertitude rapportée	4/27 (15)	14/14 (100)	21/45	47 [33 ; 61]	44/50	88
X3	Seuls les meilleurs essais gardés	1/27 (4)	1/14 (7)	2/45	4 [1 ; 15]	7/50	14
X4	Références réglées avec le même soin	0/27 (0)	2/14 (14)	2/44	5 [1 ; 15]	8/49	16
X7	Calcul chiffré	12/27 (44)	8/14 (57)	21/45	47 [33 ; 61]	29/50	58

Les tendances. Le test de Cochran-Armitage, avec les années pour scores, donne une tendance nette pour les quatre pratiques après correction de Holm sur les cinq critères testés : $z = 4{,}29$ pour le code ($p_{\text{Holm}} = 7 \times 10^{-5}$), $z = 4{,}78$ pour la sensibilité (9×10^{-6}), $z = 3{,}44$ pour le calcul ($0{,}002$), $z = 3{,}39$ pour l'incertitude ($0{,}002$). Le test statistique ne montre aucune tendance ($z = 1{,}53$, $p = 0{,}13$).

Aucun saut juste après 2019. De 2018 à 2019-2020, aucun écart ne passe la correction. Le code monte de $+24$ points, $[+3\,;\,+42]$, avec $p = 0{,}03$ avant correction et $p_{\text{Holm}} = 0{,}16$ après. L'incertitude monte de $+20$ points, $[+1\,;\,+37]$, $p_{\text{Holm}} = 0{,}23$. La sensibilité ne monte pas (-4 points). De 2019-2020 à 2022-2024, seule la sensibilité monte nettement, de 40 % à 73 %, avec un intervalle de $+11$ à $+50$ points et $p_{\text{Holm}} = 0{,}03$. Ces séries décrivent une hausse lente, commencée avant la liste de contrôle ou avec elle. Elles ne disent pas ce qui l'a causée. Coakley et al. (2026) voient de même la part du code publié monter avant les listes de contrôle, sur cinq congrès d'intelligence artificielle.

Sans les treize articles. Sans les articles finis par `claude-opus-5`, les quatre tendances passent toujours la correction ($p_{\text{Holm}} < 0{,}004$) et le test statistique n'en montre toujours aucune. La montée tardive de la sensibilité garde $p_{\text{Holm}} = 0{,}02$ (`trajectory_sans_opus5.json`).

TABLE 4.6. Trajectoire du RL classique de 2018 à 2026 (écart 25 de `study/PROTOCOL.md`, écrit avant tout tirage) : part des articles qui remplissent chaque critère, en pourcentage, par année. 2018 : notre lecture des articles à expériences ; 2019, 2020, 2022 et 2024 : les 20 premiers articles expérimentaux de RL classique par année, tirés au hasard dans ICML et NeurIPS (ICLR absente, son site refusant les téléchargements automatiques) ; 2026 : l'échantillon classique de la comparaison principale. Le nombre entre parenthèses donne le dénominateur quand il diffère de l'effectif de la colonne (critère sans objet pour certains articles). Avec 20 articles, une part a une incertitude d'environ vingt points de part et d'autre. Pour les cinq critères testés (Q5, Y9, X7, X2, Q8) : test de tendance de Cochran-Armitage sur les six années (scores égaux aux années), p bilatéral et p corrigé par Holm sur les cinq ; les six autres lignes sont descriptives. Partie basse : écart immédiat (2019 et 2020 réunis contre 2018) et écart tardif (2022 et 2024 contre 2019 et 2020), en points, intervalle de Newcombe à 95 %, Holm sur les cinq critères dans chaque famille. **En gras**, ce qui passe la correction de Holm. Source : `study/trajectory.json` (`study/trajectory.py`).

Code	Critère	2018	2019	2020	2022	2024	2026	p tendance	p Holm
	n	48	20	20	20	20	50		
Q5	**Code en lien**	**31**	**60**	**50**	**70**	**75**	**74**	**2×10^{-5}**	**7×10^{-5}**
Y9	**Sensibilité à un hyperparamètre**	**44**	**35**	**45**	**60**	**85**	**80**	**2×10^{-6}**	**9×10^{-6}**
X7	**Calcul chiffré**	**27**	**30**	**40**	**40**	**55**	**58**	**0,0006**	**0,002**
X2	**Incertitude rapportée**	**63**	**80**	**85**	**95**	**100**	**88**	**0,0007**	**0,002**
Q8	Test statistique appliqué	6	5	10	10	25	12	0,13	0,13
Q3	Tous les hyperparamètres de la méthode	55 (47)	45	55	70	55	72		
Q4	Tous les hyperparamètres des références	35	16 (19)	45	25	40	29 (49)		
Q6	Choix des hyperparamètres décrit	26 (47)	30	40	40	35	48		
Q7	Évaluation sur conditions tenues à part	33	25	20	20	55	45 (49)		
X4	Références réglées avec le même soin	13	26 (19)	35	15	25	16 (49)		
Y7	Version des bancs d'essai	23	25	20	40	20	26		

Écarts entre périodes (Newcombe, Fisher, Holm sur les cinq critères testés)

Code	Critère	2019-2020 contre 2018 écart [IC]	p Holm	2022-2024 contre 2019-2020 écart [IC]	p Holm
Q5	Code en lien	+24 [+3 ; +42]	0,16	+17 [−3 ; +36]	0,49
Y9	Sensibilité à un hyperparamètre	−4 [−23 ; +16]	1	**+32 [+11 ; +50]**	**0,03**
X7	Calcul chiffré	+8 [−11 ; +27]	1	+13 [−9 ; +32]	0,62
X2	Incertitude rapportée	+20 [+1 ; +37]	0,23	+15 [+1 ; +30]	0,23
Q8	Test statistique appliqué	+1 [−10 ; +14]	1	+10 [−5 ; +25]	0,62

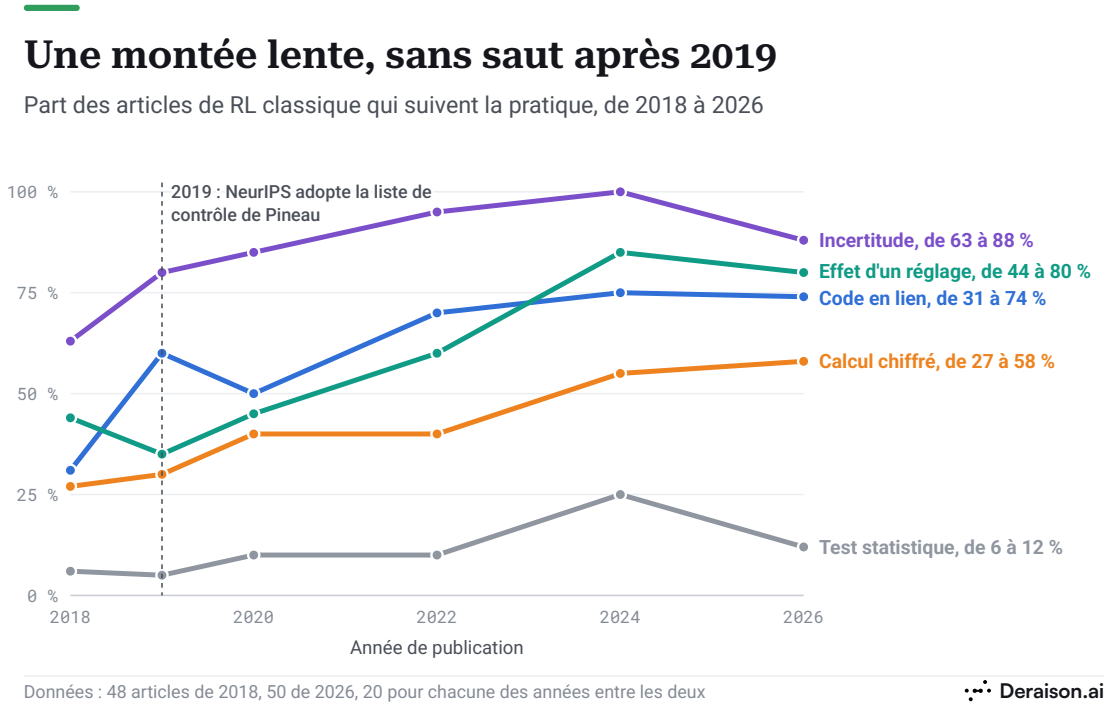


FIGURE 4.6. Part des articles de RL classique qui suivent chaque pratique, année par année : 48 articles de 2018 (ceux qui comportent des expériences), 20 par année pour 2019, 2020, 2022 et 2024, tirés d'ICML et de NeurIPS, 50 de 2026. La ligne verticale en pointillé marque la liste de contrôle de NeurIPS (2019). Le code en lien, la sensibilité à un hyperparamètre, le calcul chiffré et l'incertitude rapportée montent régulièrement. Le test statistique, en gris, ne montre aucune tendance. Avec 20 articles, une part a une incertitude d'environ vingt points de part et d'autre. Données : `study/trajectory.json`.

Le pic du test statistique en 2024. Le test statistique atteint 25 % des articles en 2024 (5 sur 20), puis retombe à 12 % en 2026 (6 sur 50). Avec 20 articles, ce pic reste dans la marge d'incertitude. Nous ne le lisons pas comme un fait.

4.8 L'accord entre lecteurs

Le tableau 4.7 et la figure 4.7 donnent l'accord des doubles lectures à l'aveugle, critère par critère, sur les lectures brutes avant harmonisation.

La grille v1 sur le premier tirage. Le second lecteur G et les lecteurs principaux s'accordent sur 127 décisions sur 130 (98 %), soit 13 critères sur 10 articles. Les trois désaccords touchent Q3, Q7 et X4, avec des kappas de 0,80, 0,78 et 0,74. Le kappa vaut 1 pour les autres critères où les deux lecteurs varient.

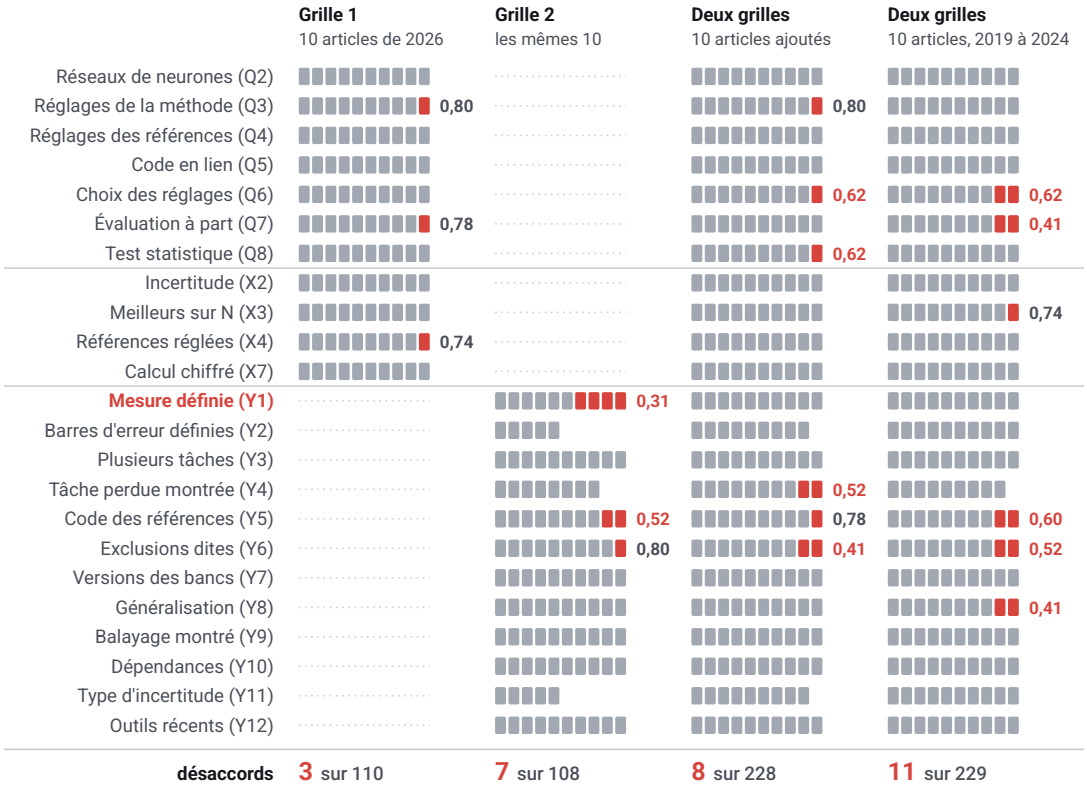
La grille v2 sur le premier tirage. Sur les mêmes dix articles, G2 et les lecteurs principaux s'accordent complètement sur 9 critères sur 12. L'accord tombe à 90 % sur les exclusions (Y6, kappa 0,80), à 80 % sur l'implémentation des références (Y5, kappa 0,52) et à 60 % sur la mesure de performance (Y1, kappa 0,31). Y1 est le seul critère fragile de toutes les doubles lectures. Ses règles ont été précisées ensuite par les écarts 17 et 19.

TABLE 4.7. Accord entre deux lectures à l'aveugle d'agents de langage, sur les lectures brutes, avant toute harmonisation. G : dix articles du premier tirage de 2026 relus avec la grille v1 (127 décisions identiques sur 130, `study/audit_stats.json`) ; G2 : les mêmes dix articles relus avec la grille v2 ; R2 : dix articles tirés au hasard parmi les articles ajoutés (`random.Random(7)`, écart 12), relus avec les deux grilles ; R3 : dix articles de la trajectoire 2019 à 2024 (`random.Random(2030)`, écart 26). Accord : décisions identiques sur les articles où les deux lecteurs ont répondu oui ou non (Y3 converti en oui à partir de deux tâches) ; κ : kappa de Cohen, vide quand un des deux lecteurs ne varie pas (accord trivial). Une case vide hors κ : critère absent de cette double lecture. **En gras**, le seul critère sous 70 % d'accord, signalé comme fragile par le script : Y1, mesure de performance, avant que ses règles soient précisées (écarts 17 et 19). Tous les lecteurs sont des copies du même modèle (écart 28) : l'accord mesure la constance de ce modèle, pas celle de deux lecteurs indépendants. Sources : `study/agree_v2.json`, `study/audit_stats.json`.

		G tirage 2026, v1		G2 tirage 2026, v2		R2 articles ajoutés		R3 trajectoire	
Code	Critère	accord	κ	accord	κ	accord	κ	accord	κ
Q1	L'article a des expériences	10/10							
Q2	Réseaux de neurones employés	10/10	1,00			10/10	1,00	10/10	
Q3	Tous les hyperparamètres de la méthode	9/10	0,80			9/10	0,80	10/10	1,00
Q4	Tous les hyperparamètres des références	10/10	1,00			10/10	1,00	10/10	1,00
Q5	Code en lien	10/10	1,00			10/10	1,00	10/10	1,00
Q6	Choix des hyperparamètres décrit	10/10	1,00			9/10	0,62	8/10	0,62
Q7	Évaluation sur conditions tenues à part	9/10	0,78			10/10	1,00	8/10	0,41
Q8	Test statistique appliqué	10/10	1,00			9/10	0,62	10/10	1,00
X2	Incertitude rapportée	10/10	1,00			10/10	1,00	10/10	
X3	Seuls les meilleurs essais gardés	10/10				10/10		9/10	0,74
X4	Références réglées avec le même soin	9/10	0,74			10/10		10/10	1,00
X5	Famille (llm, classic, theory)	10/10	1,00						
X7	Calcul chiffré	10/10	1,00			10/10	1,00	10/10	1,00
Y1	**Mesure de performance définie**			**6/10**	**0,31**	**10/10**	**1,00**	**10/10**	**1,00**
Y2	Barres d'erreur définies			5/5	1,00	9/9	1,00	10/10	1,00
Y3	Plusieurs tâches (au moins 2)			10/10	1,00	10/10		10/10	1,00
Y4	Une tâche perdue montrée			8/8	1,00	8/10	0,52	9/9	1,00
Y5	Origine du code des références			8/10	0,52	9/10	0,78	8/10	0,60
Y6	Exclusions dites			9/10	0,80	8/10	0,41	8/10	0,52
Y7	Version des bancs d'essai			10/10	1,00	10/10	1,00	10/10	1,00
Y8	Test de généralisation			10/10	1,00	10/10	1,00	8/10	0,41
Y9	Sensibilité à un hyperparamètre			10/10	1,00	10/10	1,00	10/10	1,00
Y10	Versions des bibliothèques			10/10		10/10	1,00	10/10	
Y11	Type d'incertitude nommé			5/5	1,00	9/9	1,00	10/10	1,00
Y12	Outil statistique postérieur à 2018			10/10		10/10	1,00	10/10	1,00

Lues deux fois à l'aveugle, 646 décisions sur 675 concordent

Quatre fois, un second agent relit dix articles sans voir la première lecture. Un carré par article lu deux fois : gris si les deux lectures concordent, rouge sinon. Le nombre est le kappa de Cohen.



Pointillé : critère absent de cette grille. Moins de dix carrés : le critère ne s'applique pas à tous les articles. Kappa en rouge : sous 0,70. Nom en rouge : la seule rangée sous 70 % d'accord, la mesure de performance (Y1) lue avec la grille 2.

Données : study/audit_stats.json (grille 1), study/agree_v2.json (grille 2), lectures brutes avant harmonisation ; intervalles du kappa non calculés

Deraison.ai

FIGURE 4.7. Accord entre les doubles lectures à l'aveugle, critère par critère, sur les lectures brutes, avant harmonisation : part de décisions identiques et kappa de Cohen quand les deux lecteurs varient. Quatre paires de lectures de dix articles chacune : G (grille v1) et G2 (grille v2) sur le premier tirage de 2026, R2 sur les articles ajoutés de 2018 et de 2026, R3 sur la trajectoire de 2019 à 2024. Le seuil de 70 % d'accord, sous lequel un critère est dit fragile, est marqué. Données : `study/agree_v2.json`, `study/audit_stats.json`.

Les articles ajoutés. R2 relit dix articles tirés parmi les ajouts de 2018 et de 2026 avec les deux grilles. L'accord atteint au moins 80 % sur chacun des 23 critères et il est complet sur 17 d'entre eux. Les accords les plus bas sont ceux de Y4 (80 %, kappa 0,52) et de Y6 (80 %, kappa 0,41).

La trajectoire. R3 relit dix articles de A4. L'accord atteint au moins 80 % sur chacun des 23 critères et il est complet sur 17 d'entre eux, dont les cinq critères testés de la trajectoire (Q5, Y9, X7, X2, Q8). Les kappas les plus bas sont ceux de Q7 et de Y8 (80 %, kappa 0,41).

La stabilité d'un seul modèle. Tous les lecteurs sont des copies du même modèle, à part les treize articles achevés par `claude-opus-5`. L'accord des doubles lectures mesure donc la stabilité de la lecture de ce modèle, pas l'accord de deux lecteurs indépendants. Un humain aurait pu lire autrement les mêmes passages. Les citations et les pages de `study/codes/` et `study/codes_v2/` permettent à chacun de vérifier chaque décision.

4.9 Ce que les articles déclarent

La signification statistique. La question de la liste de contrôle de NeurIPS sur la signification statistique se contente de barres d'erreur « ou d'une autre information appropriée ». Sur les 30 articles de NeurIPS 2025 lus, 18 y répondent « Yes ». Aucun des 18 n'applique de test : la liste ne demande pas le test et les articles ne le font pas. Un seul des 18 ne rapporte pas non plus d'incertitude (`study/checklist.json`).

Le code, les réglages et le calcul. Sur le code, 5 articles sur 22 répondent « Yes » sans lien vers leur code dans la version que nous avons lue, soit 23 % $[10\,;\,43]$. Quatre n'ont pas de lien même dans la version des actes. Trois écrivent dans leur propre justification que le code sera publié plus tard. Sur les réglages, 6 articles sur 22 répondent « Yes » sans donner tous les hyperparamètres de leur méthode selon notre règle. Sur le calcul, 6 articles sur 21 répondent « Yes » sans chiffrer de temps de calcul, que la question demande pourtant. Un article n'a pas de liste lisible dans les actes. Une case cochée ne garantit pas la pratique qu'elle nomme.

4.10 La sensibilité au filtre de titre

Combien d'articles de RL le filtre manque. Sur les 400 articles tirés hors du filtre de titre, 25 ont un résumé qui correspond à l'expression régulière : 6,2 %, avec un intervalle de Wilson de 4,3 % à 9,1 % (`study/title_filter.json`). Rapportée à la population hors filtre, cette part donne environ 1 037 articles de RL manqués, de 708 à 1 503, soit 45 % de l'ensemble des articles de RL selon ce critère (de 36 % à 55 %). C'est un majorant : la règle, mécanique, compte aussi des faux positifs, comme un article où « RL » désigne l'apprentissage de représentations. La plupart des articles trouvés traitent des grands modèles de langue.

Ce que ces articles changent. Lus avec les deux grilles, les 20 premiers articles de RL caché ressemblent au premier tirage de 2026 : le code en lien dans 60 % d'entre eux (67 % dans le tirage), une incertitude dans 50 % (47 %). Treize traitent des grands modèles

de langue. Les 7 articles de RL classique sont trop peu nombreux pour éprouver nos verdicts. Le filtre rate beaucoup d'articles, surtout sur les modèles de langue, sans signe que ces articles suivent d'autres pratiques.

4.11 Synthèse

À grille égale, dans le RL classique, quatre pratiques ont changé entre 2018 et 2026 : le lien vers le code, la sensibilité à un hyperparamètre montrée, le calcul chiffré, une incertitude rapportée. Le code et le calcul résistent à toutes les coupes, la sensibilité à presque toutes. L'incertitude rapportée échoue dans deux contrôles, avec $p_{\mathrm{Holm}} = 0{,}07$. Les quatre montent lentement de 2018 à 2024, sans saut juste après la liste de contrôle de 2019. Le test statistique, les références décrites et réglées avec soin, l'évaluation tenue à part, les versions des bancs d'essai et des bibliothèques n'ont pas bougé de façon détectable. Pineau a été écoutée sur ce qui se publie à côté du résultat, beaucoup moins sur la façon dont le résultat lui-même est jugé. Notre lecture de 2018 diffère de celle de Pineau sur les hyperparamètres de la méthode et sur l'évaluation tenue à part, d'où la règle de verdict à grille égale. Le chapitre 10 rassemble les limites de ces résultats. Le chapitre 11 les met en regard des expériences.

CHAPITRE 5

Les expériences : protocole et méthodes

Le sondage des chapitres 3 et 4 dit ce que les articles publient. Les expériences mettent les failles de 2018 à l'épreuve sur des entraînements réels, faits pour ce rapport. Elles reprennent le terrain de Henderson et al. (2018) : des corps simulés dans le simulateur physique MuJoCo, que des algorithmes d'apprentissage par renforcement apprennent à faire avancer. Elles le font avec les algorithmes de 2026 et avec les règles que Joelle Pineau demandait : assez de graines aléatoires, des intervalles de confiance, des références traitées avec le même soin, une évaluation tenue à part, du code et des données publiés.

Ce chapitre fixe tout ce qui précède un résultat : les outils et leurs versions, les programmes et leurs réglages, les tâches, les budgets, les graines aléatoires, le protocole d'évaluation, le format des fichiers, les statistiques et le plan de chaque expérience. Il décrit ce que le code fait, ligne à ligne quand c'est utile. Il signale aussi chaque endroit où le code s'écarte du protocole écrit. Les résultats viennent au chapitre 6. La chaîne logicielle qui relie les essais aux documents est dessinée au chapitre 8.

5.1 Un protocole écrit avant le premier entraînement

Le protocole des expériences est le fichier `sim/SIM_PROTOCOL.md`. Sa première version date du 8 novembre 2026. Elle a été écrite avant tout entraînement. Elle fixe les questions, les algorithmes, les tâches, les budgets, le nombre de graines aléatoires, l'évaluation et les analyses. Elle pose aussi trois règles de conduite :

- les graines aléatoires sont fixées avant de lancer, de 0 à 29 ;
- aucun essai n'est retiré ni relancé à cause de son score ;
- un essai qui plante est relancé une fois avec la même graine aléatoire, puis l'incident est noté.

Tout ce qui change ensuite s'inscrit dans une liste numérotée d'« écarts », datée, avec la mention de ce que nous avions déjà vu au moment de décider. Le protocole en compte vingt au 21 novembre 2026. Le chapitre 9 les reprend tous, avec ceux du sondage et du volet des modèles de langue. Ce chapitre cite un écart par son numéro quand il change une méthode. La section 5.12 classe les décisions selon qu'elles précèdent ou suivent les résultats qu'elles touchent.

Dans tout le rapport, un *essai* désigne un entraînement complet d'un programme sur une tâche avec une graine aléatoire donnée. Les résultats *principaux* sont ceux qu'annonçait le protocole : dix graines aléatoires par couple de programme et de tâche. Les ajouts de l'écart 12 (vingt graines aléatoires, budget égal, réplication) viennent à côté. Chaque divergence est dite.

5.2 Les outils et leurs versions

Quatre environnements Python figés. Quatre environnements Python 3.11.18 ont servi, tous sur un Mac à puce Apple M2 Max (macOS 27.2, arm64). Chacun est figé par un `pip freeze` pris le 15 novembre 2026, dans `envs/` (tableau D.2, chapitre D) :

- `sim/env` entraîne et réévalue (`sim/run.py`, `sim/dose.py`) : Stable-Baselines3 2.9.2 (Raffin et al. 2021), sb3-contrib 2.9.2, SBX 0.28.0 (Raffin 2023), JAX 0.10.2 (Bradbury et al. 2018), Flax 0.12.8, Optax 0.2.8, PyTorch 2.15.0 (Paszke et al. 2019), Gymnasium 1.1.1 (Towers et al. 2024), MuJoCo 4.0.0 (Todorov et al. 2012) ;
- `sim/env-an` calcule toutes les analyses : NumPy 2.4.6, SciPy 1.17.1, statsmodels 0.15.0, rliable 1.2.0 (Agarwal et al. 2021) ;
- `sim/env-zoo` fait tourner le script officiel de RL Baselines3 Zoo 2.9.2 (Raffin 2020) pour l'expérience E6, avec les mêmes bibliothèques que `sim/env` ;
- `sim/env-llm` sert au volet des modèles de langue (chapitre 7).

Ce que chaque outil apporte. Gymnasium, successeur d'OpenAI Gym maintenu par la fondation Farama, définit l'interface des tâches : `reset` rend une première observation, `step` applique une action et rend l'observation suivante, la récompense et deux drapeaux de fin d'épisode. MuJoCo calcule la physique. Il est gratuit et ouvert depuis 2022, alors qu'il demandait une licence payante en 2018. Stable-Baselines3 fournit PPO écrit en PyTorch. SBX (« Stable-Baselines Jax ») réécrit PPO, SAC et CrossQ en JAX avec la même interface. sb3-contrib fournit TRPO. rliable calcule les moyennes interquartiles, leurs intervalles bootstrap et les probabilités d'amélioration.

Un écart de version. Le protocole annonçait Stable-Baselines3 2.6.0. Tous les entraînements ont tourné avec la version 2.9.2, installée le 8 novembre à 10 h 11, avant le premier entraînement (écart 14, point c). Les versions de `envs/train.txt` sont celles qui ont réellement tourné.

Reproduire sur une autre machine. Un essai est déterminé par sa graine aléatoire sur une machine donnée : chaque entraînement tourne sur un seul fil de calcul (`OMP_NUM_THREADS=1`, `torch.set_num_threads(1)`, options XLA à un fil pour JAX). Sur un autre processeur, MuJoCo, JAX et PyTorch arrondissent autrement. La même graine aléatoire y donne une autre courbe. Une relance ailleurs met à l'épreuve les statistiques du rapport. Elle ne retrouve aucune courbe à l'identique.

5.3 Les tâches

Les quatre tâches sont celles de Henderson et al. (2018), dans leur version courante de Gymnasium : HalfCheetah-v5, Hopper-v5, Walker2d-v5 et Swimmer-v5. Dans chacune, un corps articulé est simulé par MuJoCo. L'agent observe à chaque pas un vecteur de positions et de vitesses, puis choisit une action continue : le couple appliqué à chaque articulation. La récompense paie surtout la vitesse vers l'avant, moins un coût qui croît avec l'effort des moteurs. Pour Hopper et Walker2d, elle ajoute une prime à chaque pas où le corps reste debout. Un épisode s'arrête au plus tard après 1 000 pas.

- HalfCheetah est un guépard plan à six articulations. Il ne tombe jamais : ses épisodes durent toujours 1 000 pas.
- Hopper est un sauteur à une jambe. L'épisode s'arrête quand il tombe, ce qui rend les scores très inégaux d'un essai à l'autre.
- Walker2d est un marcheur plan à deux jambes, arrêté lui aussi à la chute.
- Swimmer est un nageur à trois segments dans un fluide visqueux. Ses scores sont bien plus bas que ceux des trois autres tâches.

Le « score » d'un épisode est son retour : la somme des récompenses reçues de son premier à son dernier pas. Les scores des quatre tâches ne se comparent pas entre eux. Toutes les comparaisons de ce rapport se font à l'intérieur d'une tâche.

L'expérience E6 emploie aussi Hopper-v4, la version précédente. Les réglages de PPO que publie RL Baselines3 Zoo pour Hopper sont inscrits sous ce nom de tâche dans le fichier `hyperparams/ppo.yml` du recueil : ils ont été réglés sur cette version.

5.4 Les programmes

Nos expériences comparent quatre *programmes*, c'est-à-dire quatre codes prêts à l'emploi, qui réalisent trois *algorithmes*. Un cinquième programme, TRPO, sert à une seule expérience (B1). Le tableau 5.1 donne leurs réglages.

PPO, deux fois. PPO (*Proximal Policy Optimization*, Schulman et al. 2017) est l'algorithme de référence de 2018. Il sert encore de point de comparaison courant en 2026. C'est aussi l'ancêtre direct de GRPO, l'algorithme du volet des modèles de langue. PPO est un algorithme « sur politique » (*on-policy*) : il collecte un paquet de transitions avec la politique courante, s'en sert pour quelques passes de gradient, puis le jette. Soit $r_t(\theta) = \pi_\theta(a_t \mid s_t) / \pi_{\theta_{\text{old}}}(a_t \mid s_t)$ le rapport des probabilités de l'action sous la nouvelle et l'ancienne politique. Soit $\hat{A}_t$ l'avantage estimé par GAE (*generalized advantage estimation*) : combien l'action prise a rapporté de plus que ce que la fonction de valeur prévoyait pour cet état. PPO maximise l'objectif tronqué

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t\Big[\min\big(r_t(\theta)\,\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\,\hat{A}_t\big)\Big], \tag{5.1}$$

avec $\epsilon = 0{,}2$, plus un terme d'erreur de la fonction de valeur. Deux programmes réalisent PPO :

- PPO PyTorch, la classe `PPO` de Stable-Baselines3 (`ppo_sb3` dans le code) ;
- PPO JAX, la classe `PPO` de SBX (`ppo_sbx` dans le code).

Les deux ont les mêmes valeurs par défaut pour tous les hyperparamètres de l'algorithme. Ils diffèrent par l'initialisation des poids : Stable-Baselines3 initialise ses couches par une matrice orthogonale (`ortho_init=True`), SBX garde l'initialisation par défaut de Flax (`ortho_init=False`). Ils diffèrent aussi par la bibliothèque de calcul, ses générateurs aléatoires et ses arrondis. L'expérience E3 mesure ce que ces différences coûtent.

SAC. SAC (*Soft Actor-Critic*, Haarnoja et al. 2018) est le standard « hors politique » (*off-policy*) du contrôle continu depuis 2019. Il garde toutes les transitions vues dans

une mémoire de rejeu et apprend à chaque pas sur un lot tiré de cette mémoire. Il maximise le retour augmenté de l'entropie de la politique, $\sum_t \mathbb{E}\big[r_t + \alpha\,\mathcal{H}(\pi(\cdot \mid s_t))\big]$, avec un coefficient α réglé automatiquement vers une entropie cible. Deux réseaux critiques estiment la valeur des actions. Des copies lentes de ces réseaux, dites réseaux cibles, stabilisent l'apprentissage. Le programme est la classe `SAC` de SBX (`sac`).

CrossQ. CrossQ (Bhatt et al. 2024), paru à ICLR 2024, vise l'efficacité en données, le score atteint par pas d'interaction avec l'environnement. C'est un descendant de SAC. Il supprime les réseaux cibles, normalise les activations par lots (*batch normalization*) dans le critique et l'acteur. Il élargit aussi le critique. Le programme est la classe `CrossQ` de SBX (`crossq`). Son critique a deux couches de 1 024 neurones, au lieu des 2 048 de l'article : SBX fait ce choix pour calculer plus vite. Nous l'avons gardé.

TRPO, pour B1 seulement. TRPO (*Trust Region Policy Optimization*, Schulman et al. 2015) est l'algorithme de la figure 5 de Henderson et al. (2018), celle dont l'expérience B1 cherche à expliquer le t de 9 (section 5.10.2). Il maximise l'avantage pondéré par le rapport des probabilités sous une contrainte de divergence de Kullback-Leibler moyenne entre l'ancienne et la nouvelle politique, $\hat{\mathbb{E}}_t\big[\mathrm{KL}(\pi_{\theta_{\text{old}}} \| \pi_\theta)\big] \le \delta$, résolue par gradient conjugué puis recherche linéaire. Le programme est la classe `TRPO` de sb3-contrib (`trpo`), avec toutes ses valeurs par défaut.

TD3, prévu puis abandonné. Le protocole prévoyait TD3 si le budget le permettait. Il ne l'a pas permis : TD3 n'a pas été lancé (écart 1).

Pourquoi les valeurs par défaut. Les quatre programmes de la grille principale gardent les valeurs par défaut de leur bibliothèque. Aucun n'a été réglé. Ce choix applique à la lettre la demande de Pineau sur les références (faille F7, des références moins bien réglées que la méthode) : chaque programme reçoit le même soin de réglage, ici aucun. Il a un coût. Un programme bien servi par ses valeurs par défaut est avantagé. Le classement de E2 vaut pour ces réglages. L'expérience E6 mesure, sur un cas, ce que change un réglage publié. La seule commande qui construit un programme de la grille, dans `sim/run.py`, est `cls("MlpPolicy", TASK, seed=SEED, verbose=0)`, sans autre argument.

Les réglages publiés du Zoo, pour E6. L'expérience E6 oppose PPO PyTorch avec ses valeurs par défaut à PPO PyTorch avec les réglages que RL Baselines3 Zoo publie pour Hopper-v4. `sim/run.py` les recopie dans la constante `ZOO_PPO_HOPPER`, identiques à ceux de `rl_zoo3/hyperparams/ppo.yml` dans la version 2.9.2 (écart 5) :

- paquets de 512 pas, lots de 32, 5 passes, $\gamma = 0{,}999$, $\lambda = 0{,}99$ pour GAE ;
- taux d'apprentissage 9,808 28 × 10^{-5}, coefficient d'entropie 0,002 295 19, coefficient de valeur 0,835 671, norme de gradient bornée à 0,7, troncature 0,2 ;
- réseaux de deux couches de 256 neurones pour la politique et la valeur, activation ReLU, sans initialisation orthogonale, logarithme de l'écart type initial des actions à -2 ;
- normalisation des observations et des récompenses par une moyenne et un écart type glissants (`VecNormalize`).

Table 5.1. Les réglages des cinq programmes, lus dans le code des bibliothèques installées (Stable-Baselines3 2.9.2, sb3-contrib 2.9.2, SBX 0.28.0). Ce sont les valeurs par défaut : aucun programme n'a été réglé. Une case vide signifie que le réglage n'existe pas pour cet algorithme. Les réseaux sont des perceptrons multicouches, dont on donne les largeurs des couches cachées.

Réglage	PPO PyTorch	PPO JAX	SAC	CrossQ	TRPO
Taux d'apprentissage	3×10^{-4}	3×10^{-4}	3×10^{-4}	1×10^{-3}	1×10^{-3} (critique)
Facteur d'actualisation γ	0,99	0,99	0,99	0,99	0,99
Pas collectés par mise à jour	2 048	2 048	1	1	2 048
Taille des lots	64	64	256	256	128
Passes sur chaque paquet	10	10			
Paramètre λ de GAE	0,95	0,95			0,95
Troncature ϵ	0,2	0,2			
Contrainte de divergence δ					0,01
Coefficient d'entropie	0	0	automatique	automatique	
Mémoire de rejeu			1 000 000	1 000 000	
Pas avant d'apprendre			100	100	
Mise à jour des réseaux cibles τ			0,005	sans réseau cible	
Mises à jour du critique par mise à jour de l'acteur			1	3	10
Réseau de la politique	64, 64, tanh	64, 64, tanh	256, 256, ReLU	256, 256, ReLU	64, 64, tanh
Réseau de valeur ou critique	64, 64, tanh	64, 64, tanh	2 fois 256, 256	2 fois 1 024, 1 024, normalisé par lots	64, 64, tanh
Initialisation orthogonale	oui	non			oui

La normalisation des récompenses divise chaque récompense par l'écart type d'un retour actualisé glissant. Dans la recette du Zoo, son facteur d'actualisation est celui de l'algorithme, 0,999. Notre premier passage (`ppo_sb3_zoo`, option `--zoo`) a laissé la valeur par défaut de `VecNormalize`, 0,99. Le second passage (`ppo_sb3_zoofix`, option `--zoofix`) passe 0,999 (écart 2). Les deux sont conservés.

5.5 Les budgets

Ce que fixe le protocole. PPO PyTorch, PPO JAX et TRPO s'entraînent un million de pas. SAC et CrossQ s'entraînent 200 000 pas. Un pas est une transition de l'environnement : une observation, une action, une récompense.

Pourquoi deux budgets. Les deux familles d'algorithmes ne se comparent pas au même régime. Un algorithme sur politique jette ses données après chaque mise à jour : il lui faut beaucoup de pas. Un million est l'ordre de grandeur de 2018 pour ces tâches. Un algorithme hors politique réutilise ses données à chaque pas : les articles de 2024 à 2026 comparent SAC et CrossQ dans un régime « peu de données ». Le protocole fixait

300 000 pas. L'écart 1 les a ramenés à 200 000, avant le premier entraînement, pour tenir le calcul sur une machine partagée : nous mesurions alors 19 ms par pas pour ces deux algorithmes.

Le nombre exact de pas. `sim/run.py` entraîne par blocs de 50 000 pas et évalue après chaque bloc. L'appel est `model.learn(50_000, reset_num_timesteps=False)`, répété jusqu'au budget. PPO et TRPO collectent leurs transitions par paquets de 2 048 pas et ne s'arrêtent qu'à la fin d'un paquet. Chaque bloc fait donc 51 200 pas. Les modèles enregistrés de PPO PyTorch, PPO JAX et TRPO comptent 1 024 000 pas, alors que la colonne `step` des résultats écrit 1 000 000. Ceux de E6 réglés comme le Zoo, à paquets de 512 pas, en comptent 1 003 520. SAC et CrossQ font exactement 200 000 pas (écart 14, point d).

Le budget égal. La comparaison de E2 se fait à budgets inégaux. Le rapport le dit à chaque fois. L'écart 12, point iii, ajoute une lecture à budget égal : PPO est lu à 200 000 pas sur sa courbe d'évaluation, au même point que SAC et CrossQ. Aucun score de PPO à 200 000 pas n'avait été regardé avant d'écrire cette règle.

5.6 Les graines aléatoires

Ce que fixe une graine aléatoire. Une graine aléatoire est le nombre de départ dont l'ordinateur tire tout le hasard d'un entraînement : l'initialisation des réseaux, les positions initiales du corps simulé, le bruit des actions explorées, les lots tirés dans la mémoire de rejeu. `sim/run.py` la passe au constructeur du programme (`seed=SEED`), qui en dérive les générateurs de NumPy, de PyTorch ou de JAX et celui de l'environnement d'entraînement. Même programme, même graine aléatoire, même machine : même essai.

Graines aléatoires d'entraînement. Les graines aléatoires d'entraînement sont les entiers 0 à 29, fixés avant le premier entraînement :

- 0 à 9 pour chaque couple de programme et de tâche, les résultats principaux ;
- 10 à 19 pour la consolidation de l'écart 12, les vingt graines aléatoires ;
- 0 à 29 pour PPO PyTorch sur HalfCheetah (E1) et pour TRPO sur HalfCheetah (B1).

Les graines aléatoires 10 à 19 de PPO PyTorch sur HalfCheetah, entraînées pour E1 avec le même code, servent telles quelles à la consolidation.

Graines aléatoires d'évaluation. L'évaluation emploie les graines aléatoires 10 000 à 10 009, une par épisode d'évaluation. Elles ne servent jamais à l'entraînement, qui emploie les graines 0 à 29. Les mêmes dix graines aléatoires servent à chaque point d'évaluation, à chaque essai et à chaque physique : deux politiques sont toujours évaluées sur les mêmes dix positions de départ.

Graines aléatoires des analyses. Les tirages des analyses ont leurs propres graines aléatoires, fixées dans chaque script : 2018 pour les tirages de E1 et B1 (`np.random.default_rng(2018)`), 2026 pour E5 et pour tous les bootstraps de rliable (section 5.9.10). Deux exécutions d'une analyse rendent un fichier identique à l'octet près.

Chaque script nomme ses graines aléatoires. Les scripts d'analyse lisaient d'abord tous les fichiers présents. Avec les graines aléatoires 10 à 19 de l'écart 12, E6 serait passé de dix à vingt graines aléatoires sans le dire. Depuis l'écart 14, point f, chaque script nomme ses graines aléatoires et s'arrête s'il en manque une. Les résultats publiés n'ont pas bougé.

5.7 L'évaluation, tenue à part

Le protocole d'évaluation. Après chaque bloc d'entraînement, la fonction `evaluate` de `sim/run.py` joue dix épisodes. Pour l'épisode i, de 0 à 9 :

1. un environnement neuf est créé (`gym.make(TASK)`), distinct de celui de l'entraînement ;
2. il est remis à zéro avec la graine aléatoire $10\,000 + i$;
3. la politique choisit chaque action sans hasard (`deterministic=True`) : l'action moyenne pour PPO, SAC et CrossQ, au lieu d'un tirage dans la loi de la politique ;
4. l'épisode court jusqu'à la chute ou jusqu'à 1 000 pas, puis son retour est noté.

La fonction rend la moyenne et l'écart type des dix retours (`np.std`, sans correction de Bessel). Pour les programmes réglés comme le Zoo, chaque observation passe par la normalisation de l'entraînement, dont les statistiques ne bougent pas pendant l'évaluation. Une politique déterministe évaluée sur dix graines aléatoires fixes donne toujours le même score. La variation entre les dix épisodes ne vient que des positions de départ.

Le score d'un essai. Le score d'un essai est la moyenne des dix retours à la dernière évaluation, au budget de son programme : la colonne `eval_mean` de la dernière ligne de son fichier. C'est la mesure principale fixée par le protocole. L'expérience B3 en essaie deux autres.

Une physique modifiée, à la fin. À la fin de l'entraînement, la politique finale joue dix épisodes de plus, sur les mêmes graines aléatoires d'évaluation, dans une physique modifiée. Le code modifie le modèle MuJoCo de l'environnement avant la remise à zéro :

- la masse de chaque corps est multipliée par 1,2 (`body_mass[:] *= 1.2`) ;
- le coefficient de frottement de glissement de chaque géométrie, sol compris, est multiplié par 0,8 (`geom_friction[:, 0] *= 0.8`).

La moyenne des dix retours s'écrit dans la colonne `perturbed_mean` de la dernière ligne. Elle sert à l'expérience E7.

La physique modifiée par degrés. L'écart 12, point ii, réévalue chaque politique finale dans d'autres physiques, sans aucun entraînement (`sim/dose.py`). Une seule grandeur change à la fois :

- les masses, multipliées par 0,7, 0,8, 0,9, 1,0, 1,1, 1,2 et 1,3, soit de 30 % plus légères à 30 % plus lourdes ;

- les frottements de glissement, multipliés par 0,6, 0,7, 0,8, 0,9, 1,1 et 1,2, soit de 40 % plus faibles à 20 % plus forts.

Chaque physique est jouée sur les dix mêmes graines aléatoires d'évaluation, avec le même code d'évaluation. Le facteur 1,0 sert de contrôle : une réévaluation correcte y redonne le score final écrit par `sim/run.py`. Le résultat de chaque politique est un fichier `results/dose/<programme>/<tâche>/seed<k>.json`. Les politiques de PPO de la grille principale n'ont pas de normalisation des observations : `sim/dose.py` les évalue telles quelles.

5.8 Le format des essais

Un fichier CSV par essai. Chaque essai écrit un fichier `results/<programme>/<tâche>/seed<k>.csv`, réécrit après chaque évaluation. Ses colonnes sont `step`, `eval_mean`, `eval_std`, `perturbed_mean`, `wall` :

- `step`, le nombre de pas annoncé à cette évaluation, de 50 000 en 50 000 ;
- `eval_mean` et `eval_std`, la moyenne et l'écart type des retours des dix épisodes d'évaluation ;
- `perturbed_mean`, la moyenne des retours dans la physique modifiée, vide sauf à la dernière ligne ;
- `wall`, les secondes écoulées depuis le début de l'essai.

Une courbe de PPO ou de TRPO a vingt lignes, une courbe de SAC ou de CrossQ en a quatre. Le module `src/pineau_rl/runs.py` lit ce format. Sa fonction `read_run` vérifie les colonnes requises et la croissance stricte des pas. Elle tient pour inachevé un essai dont la dernière ligne n'est pas au budget du programme. Elle coupe aussi, sur demande, une courbe à un pas donné (lecture à budget égal). La fonction `write_run` écrit ce format depuis n'importe quelle bibliothèque.

Les politiques finales. Chaque essai enregistre sa politique finale dans `models/<programme>/<tâche>/seed<k>.zip`, avec les statistiques de normalisation à côté pour les programmes réglés comme le Zoo (`.vecnorm.pkl`). Le fichier `sim/models/MANIFEST.sha256` donne l'empreinte SHA-256 de 250 politiques, prises le 15 novembre 2026 sur la machine qui les a entraînées. Les politiques de PPO, de TRPO et de E6 sont dans le dépôt. Celles de SAC et de CrossQ, trop lourdes pour git, sont attachées à la publication `policies-v1` du dépôt sur GitHub. Le script `sim/fetch_models.py` les télécharge une à une, espacées d'une pause aléatoire de 1,5 à 4 secondes, vérifie chaque archive puis chaque politique contre le manifeste. L'option `--check` vérifie seulement les politiques présentes.

Les journaux. Chaque lancement écrit un journal. Depuis l'écart 14, chaque tentative garde le sien (`logs/<tâche>.attempt<n>.log`, jamais écrasé) et ajoute une ligne au registre `logs/attempts.jsonl` : tâche, numéro de tentative, début, fin, code de sortie, empreinte SHA-256 du résultat. Pour la période antérieure, le registre `logs/attempts_reconstructed.csv` reconstruit une ligne par tentative à partir des dates des journaux, des sorties des lanceurs et de l'historique git (section 5.11).

5.9 Les statistiques

Cette section définit chaque statistique employée, dans l'ordre où le chapitre 6 en a besoin. Les fonctions sont dans `src/pineau_rl/stats.py` et `src/pineau_rl/bootstrap.py`.

5.9.1 La moyenne interquartile

La moyenne interquartile (IQM, *interquartile mean*) écarte le quart le plus bas et le quart le plus haut des scores, puis fait la moyenne du reste (Agarwal et al. 2021). Elle résiste mieux qu'une moyenne à un essai hors norme et reste plus précise qu'une médiane. rliable la calcule par la moyenne tronquée de SciPy à 25 % de chaque côté (`trim_mean(x, 0.25)`), qui retire $\lfloor n/4 \rfloor$ scores à chaque extrémité. Avec les scores triés $x_{(1)} \le \cdots \le x_{(n)}$ et $g = \lfloor n/4 \rfloor$:

$$\mathrm{IQM}(x) = \frac{1}{n - 2g} \sum_{i=g+1}^{n-g} x_{(i)}. \tag{5.2}$$

Avec dix graines aléatoires, $g = 2$: l'IQM est la moyenne des six scores du milieu. Avec vingt, $g = 5$ et dix scores restent. Avec trente, $g = 7$ et seize restent.

5.9.2 L'intervalle bootstrap stratifié

L'incertitude d'une IQM se mesure par bootstrap (Efron 1979). On tire avec remise n scores parmi les n observés, on recalcule l'IQM, puis on répète B fois. L'intervalle à 95 % est formé des centiles 2,5 et 97,5 des B valeurs obtenues (méthode des centiles, la méthode par défaut de rliable) :

$$\left[\, q_{0{,}025}\big(\mathrm{IQM}(x^{*1}), \ldots, \mathrm{IQM}(x^{*B})\big),\ q_{0{,}975}\big(\mathrm{IQM}(x^{*1}), \ldots, \mathrm{IQM}(x^{*B})\big) \,\right], \qquad B = 2\,000. \tag{5.3}$$

rliable appelle ce bootstrap « stratifié » : avec plusieurs tâches, il rééchantillonne les essais à l'intérieur de chaque tâche, sans mêler les tâches. Chacun de nos appels traite une seule tâche. La stratification n'y change rien : le bootstrap revient à rééchantillonner les essais d'un programme.

5.9.3 La probabilité d'amélioration

La probabilité d'amélioration d'un programme X sur un programme Y est la probabilité qu'un essai de X, tiré au hasard, batte un essai de Y, tiré au hasard, les égalités comptant pour moitié (Agarwal et al. 2021). Avec n_x scores x_i et n_y scores y_j :

$$P(X > Y) = \frac{1}{n_x n_y} \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} \Big(\mathbf{1}[x_i > y_j] + \tfrac{1}{2}\, \mathbf{1}[x_i = y_j] \Big) = \frac{U}{n_x n_y}, \tag{5.4}$$

où U est la statistique de Mann-Whitney (Mann et Whitney 1947). rliable la calcule ainsi (`scipy.stats.mannwhitneyu`). Elle vaut 0,5 quand les deux programmes se valent. Son intervalle à 95 % vient du même bootstrap que l'IQM : les essais de X et ceux de Y sont rééchantillonnés chacun de leur côté, $B = 2\,000$ fois, puis l'on prend les centiles 2,5 et 97,5.

5.9.4 La règle du vainqueur

Le protocole fixe une règle unique pour déclarer qu'un programme l'emporte sur un autre. Nous l'appelons *la règle du vainqueur établi*. Un écart est établi quand l'une de deux conditions tient :

- les intervalles à 95 % des deux IQM ne se chevauchent pas ;
- l'intervalle à 95 % de la probabilité d'amélioration exclut 0,5.

Sinon, aucun vainqueur n'est déclaré, quel que soit l'écart des IQM. E3 applique les deux conditions (`e3.py`). E2, E2 avec Bonferroni et B3 emploient la seconde : une paire y est dite « tranchée » quand l'intervalle de sa probabilité d'amélioration exclut 0,5. E7 et la physique par degrés en emploient l'analogue pour un rapport : une perte est établie quand l'intervalle de l'IQM du rapport reste sous 1.

5.9.5 Les tests t de Student et de Welch

Le test t compare deux moyennes. Avec deux groupes de tailles n_a et n_b, de moyennes $\bar{a}$ et $\bar{b}$ et de variances empiriques s_a^2 et s_b^2 (corrigées de Bessel), le test de Student (Student 1908) suppose les deux variances égales et calcule

$$t = \frac{\bar{a} - \bar{b}}{s_p \sqrt{1/n_a + 1/n_b}}, \qquad s_p^2 = \frac{(n_a - 1)s_a^2 + (n_b - 1)s_b^2}{n_a + n_b - 2}, \tag{5.5}$$

avec $n_a + n_b - 2$ degrés de liberté. Le test de Welch (Welch 1947) ne suppose pas les variances égales :

$$t = \frac{\bar{a} - \bar{b}}{\sqrt{s_a^2/n_a + s_b^2/n_b}}, \qquad \nu = \frac{\left(s_a^2/n_a + s_b^2/n_b\right)^2}{\dfrac{(s_a^2/n_a)^2}{n_a - 1} + \dfrac{(s_b^2/n_b)^2}{n_b - 1}}. \tag{5.6}$$

La valeur p bilatérale est $2\left(1 - F_\nu(|t|)\right)$, où F_ν est la fonction de répartition de la loi de Student à ν degrés de liberté. Deux groupes de cinq donnent $\nu \geq 4$: ν vaut au moins le plus petit des $n - 1$. Tous nos tests t sont des tests de Welch (`scipy.stats.ttest_ind`, option `equal_var=False`). C'est le test que l'expérience E1 met en cause, comme le prévoyait le protocole.

5.9.6 Trois lectures du t moyen de 2018

La légende de la figure 5 de Henderson et al. (2018) parle du « test t moyen à deux échantillons sur toute la distribution d'entraînement ». Elle compare deux groupes de courbes d'apprentissage entières. Soit $C_{s,k}^a$ le score d'évaluation de l'essai s du groupe a au point d'évaluation k, pour $k = 1, \ldots, K$ ($K = 20$ pour PPO et TRPO). Nos analyses calculent trois lectures de cette légende (`welch_curve` et `welch_pooled`).

- *Le t moyen.* Un test de Welch à chaque point, sur les cinq scores de chaque groupe, donne t_k et ν_k. On rapporte $\bar{t} = \frac{1}{K}\sum_k t_k$, converti en p avec la moyenne des ν_k.
- *Le p moyen.* Le même calcul, résumé par $\bar{p} = \frac{1}{K}\sum_k p_k$.
- *Les points empilés.* Les $5 \times K$ points de chaque groupe sont pris comme autant d'observations indépendantes, puis comparés par un seul test de Welch, à environ 198 degrés de liberté.

5.9.7 La statistique du script de Henderson

Peter Henderson a publié le 17 juin 2017 le script `from_same_dist.py` (Henderson 2017), compagnon de Islam et al. (2017). Le script fait un autre calcul que les trois lectures précédentes. La fonction `henderson_script` le reproduit sur nos fichiers. Soit $M_{s,k}$ la colonne `eval_mean` et $S_{s,k}$ la colonne `eval_std` de l'essai s au point k. Soit A et B les deux groupes de cinq essais.

1. Le script fait la moyenne des courbes de chaque groupe, colonne par colonne : $\bar{M}_k^A = \frac{1}{5}\sum_{s\in A} M_{s,k}$ et $\bar{S}_k^A = \frac{1}{5}\sum_{s\in A} S_{s,k}$.
2. À chaque point k, il applique un test de Welch construit à partir de ces seules statistiques (`scipy.stats.ttest_ind_from_stats`), avec n épisodes par groupe :

$$t_k = \frac{\bar{M}_k^A - \bar{M}_k^B}{\sqrt{\left((\bar{S}_k^A)^2 + (\bar{S}_k^B)^2\right)/n}}. \tag{5.7}$$

3. Il rapporte la moyenne des t_k et la moyenne des p_k.

Le dénominateur de l'équation (5.7) contient l'écart type *entre les épisodes* d'un même essai. La variabilité *entre les graines aléatoires*, celle que la figure de 2018 devait mesurer, n'y entre jamais. Les cinq graines aléatoires d'un groupe se fondent en une seule courbe moyenne. Pour nos essais, $n = 10$, le nombre d'épisodes d'évaluation. Une analyse de sensibilité prend $n = 5$, le nombre de trajectoires par itération des essais de 2018 (5 000 pas par itération, épisodes de 1 000 pas). Notre $S_{s,k}$ est un écart type sans correction de Bessel, passé tel quel au test, comme le script passe ses propres colonnes.

5.9.8 Corrections pour comparaisons multiples

Quand on fait m tests, la probabilité qu'au moins un se trompe dépasse le seuil de chacun. Deux corrections servent.

- *Bonferroni* (Dunn 1961) élève chaque intervalle au niveau $1 - 0{,}05/m$. Les m intervalles tiennent alors ensemble au niveau de 95 % au moins. E2 compte $m = 24$ comparaisons (6 paires sur 4 tâches) : chaque intervalle passe au niveau $1-0{,}05/24$, soit 99,79 % (écart 18).
- *Holm* (Holm 1979) ajuste des valeurs p triées $p_{(1)} \le \cdots \le p_{(m)}$ en $\tilde{p}_{(i)} = \min\bigl(1, \max_{j\le i}(m-j+1)\,p_{(j)}\bigr)$. Elle sert à E6 sur ses deux ou trois comparaisons.

5.9.9 Autres tests

Le test de Shapiro-Wilk (Shapiro et Wilk 1965) mesure l'écart d'une distribution à la loi normale. E1 l'applique aux trente scores de PPO PyTorch. Le test de Mann-Whitney (Mann et Whitney 1947) compare deux groupes par leurs rangs. Le diagnostic de E6 le rapporte à côté du test de Welch.

5.9.10 Des intervalles reproductibles

rliable 1.2.0 accepte un argument `random_state` mais tire ses rééchantillons dans le générateur global de NumPy (`np.random.choice`). Six scripts ne redonnaient pas leurs

intervalles à l'identique d'une exécution à l'autre (écart 13). Chaque script fixe désormais ce générateur au départ (`np.random.seed(2026)`), puis appelle rliable dans un ordre fixe. Chaque appel consomme le générateur global : l'intervalle d'une comparaison dépend des appels qui la précèdent dans le même script. La correction n'a changé aucune estimation ponctuelle ni aucun verdict. Les bornes ont bougé de l'ordre du bruit de Monte-Carlo, au plus 16 points sur une IQM de 2 423 pour E6.

Une variante pour le diagnostic de E6. Les scripts `e6_diag.py` et `e6_official.py` calculent les intervalles des IQM avec $B = 5\,000$ rééchantillons. Leur probabilité d'amélioration a un bootstrap à part (`probability_of_improvement_bootstrap`) : 5 000 tirages, les deux groupes rééchantillonnés l'un après l'autre avec un générateur de graine aléatoire 2018, sans rliable. Le même score peut donc recevoir deux intervalles un peu différents selon le script qui le calcule : l'IQM de PPO PyTorch par défaut sur Hopper-v5 a un intervalle dans `e6.json` et un autre dans `e6_diag.json`. Le rapport cite celui du diagnostic, comme `docs/conclusions.md`.

5.10 Les expériences, une à une

Le tableau 5.2 résume les plans. Il ne contient que des faits de conception : la question, le plan, le nombre d'essais et le moment où l'analyse a été écrite.

Table 5.2. Les expériences : la question posée, le plan, les essais employés et le moment où l'analyse a été fixée. « Préenregistrée » signifie écrite avant tout résultat de l'expérience. Aucun chiffre de cette table n'est un résultat.

Expérience	Question	Plan	Essais	Analyse fixée
E1	Deux groupes d'essais du même programme passent-ils pour deux algorithmes ?	PPO PyTorch sur HalfCheetah, 30 graines aléatoires. 10 000 tirages de deux groupes, test de Welch sur les scores finaux, au hasard puis triés.	30 entraînements	préenregistrée (8 novembre)
E1, seconde statistique	Le t moyen sur la courbe de 2018 se retrouve-t-il ?	Les mêmes 30 courbes, test de Welch à chacun des 20 points, t moyen et p moyen.	les 30 de E1	après les résultats de E1 (écart 3)
B1, TRPO	La dispersion de TRPO explique-t-elle le t de 9 ?	TRPO sur HalfCheetah, 30 graines aléatoires, trois lectures du t moyen.	30 entraînements	avant tout entraînement TRPO (écart 4)
B1, script	Le calcul de Henderson explique-t-il le t de 9 ?	Le script de 2017 appliqué aux courbes de TRPO et de PPO, mêmes tirages.	les 30 de E1 et les 30 de TRPO	après B1 TRPO, avant ce calcul (écart 6)
E2	Le programme en tête change-t-il de tâche en tâche ?	4 programmes sur 4 tâches, IQM et intervalle, probabilité d'amélioration par paire.	160 entraînements	détails fixés avant tout score de CrossQ (écarts 7 et 14)

Expérience	Question	Plan	Essais	Analyse fixée
E2, Bonferroni	Le verdict de E2 résiste-t-il à 24 comparaisons ?	Les 24 intervalles de E2 au niveau simultané de 95 %.	ceux de E2	après E2, avant ce calcul (écart 18)
E3	Deux programmes du même algorithme donnent-ils le même résultat ?	PPO PyTorch contre PPO JAX, 4 tâches, mêmes graines aléatoires.	80 de ceux de E2	préenregistrée
E4	Que devient la moyenne quand on garde les 3 meilleurs essais sur 10 ?	Moyenne et écart type des 10 puis des 3 meilleurs, inversions parmi 24 comparaisons.	ceux de E2	avant tout score de CrossQ (écart 7)
E5	Combien de graines aléatoires retrouvent le bon classement ?	Pour chaque comparaison, 10 000 tirages avec remise de n essais, n de 3 à 20.	ceux de E2, les 30 de E1	avant tout score de CrossQ (écart 7)
E6	Que change une référence réglée selon les réglages publiés ?	PPO PyTorch sur Hopper, valeurs par défaut contre réglages du Zoo, puis diagnostic et script officiel.	40 entraînements de plus que E2, plus 10 officiels	préenregistrée, diagnostic après le premier résultat (écarts 2 et 5)
E7	Que coûte une physique modifiée ?	Rapport du score en physique modifiée au score en physique d'entraînement, par essai.	ceux de E2	préenregistrée, classement descriptif ajouté après (écart 11)
Physique par degrés	La perte de E7 tient-elle à une seule modification ?	Masses de 0,7 à 1,3 fois, frottements de 0,6 à 1,2 fois, une grandeur à la fois.	320 réévaluations (avec vingt graines aléatoires)	règle écrite avant tout calcul (écarts 12 et 14)
B3	La mesure choisie change-t-elle le classement ?	Score final, meilleur score, moyenne de la courbe, inversions entre mesures.	ceux de E2	avant tout score de CrossQ (écart 8)
Vingt graines aléatoires	Les résultats à dix graines aléatoires tiennent-ils ?	Graines aléatoires 10 à 19, mêmes règles, E2, E3, E5, E7, B3, physique par degrés.	150 entraînements de plus	avant tout nouvel entraînement (écarts 12 et 20)
Budget égal	Le classement de E2 tient-il au budget de PPO ?	PPO lu à 200 000 pas, E2 et B3 recalculés.	ceux de E2	avant tout regard sur ces scores (écart 12)
E4 répliquée	Le gonflement de E4 se refait-il ?	E4 à l'identique sur les graines aléatoires 10 à 19.	160 essais des graines aléatoires 10 à 19	avant tout nouvel entraînement (écart 12)

5.10.1 E1 : même code, autre graine aléatoire

E1 entraîne trente fois PPO PyTorch sur HalfCheetah-v5, un million de pas chacun, graines aléatoires 0 à 29, sans rien changer d'autre. Le script `sim/analysis/e1.py` lit les trente scores finaux, puis fait, avec un générateur de graine aléatoire 2018 :

- les statistiques de la distribution : moyenne, écart type, médiane, minimum, maximum, coefficient de variation (écart type rapporté à la moyenne), test de Shapiro-Wilk ;
- pour $k = 3$, 5 et 10, 10 000 tirages de deux groupes disjoints de k essais, obtenus par une permutation des trente puis les k premiers contre les k suivants. Chaque tirage reçoit un test de Welch et l'écart relatif des moyennes $|\bar{a} - \bar{b}| / \min(\bar{a}, \bar{b})$. Le script rapporte la part des tirages où $p < 0{,}05$, la médiane et le 95e centile de l'écart relatif.
- la coupe « triée » : 10 000 tirages de dix essais sans remise, triés par score, puis un test de Welch des cinq meilleurs contre les cinq pires. Cette coupe montre ce que produirait un tri des essais avant la comparaison.
- la sélection des trois meilleurs sur dix : sur 10 000 tirages de dix essais, la moyenne des trois meilleurs rapportée à la moyenne des dix.

Puisque les deux groupes sortent toujours du même programme, toute différence déclarée est un faux positif. Avec des groupes tirés au hasard et un test bien calibré, la part attendue est de 5 %.

La seconde statistique. Henderson et al. (2018) ne testent pas des scores finaux mais des courbes (section 5.9.6). `sim/analysis/e1_curve.py` refait leur statistique sur les trente courbes : 10 000 tirages de dix essais, coupés en deux groupes de cinq au hasard ou triés par score final, puis un test de Welch à chacun des vingt points d'évaluation. On rapporte la part des tirages déclarés différents par le t moyen, par le p moyen et par le test au dernier point, le 95e centile de $|\bar{t}|$ et la part des tirages où $|\bar{t}| \geq 9{,}09$. Cette statistique a été ajoutée le 9 novembre, après les résultats de `e1.py` (écart 3). Elle complète l'analyse préenregistrée sans la remplacer.

5.10.2 B1 : l'énigme du t de 9

Henderson et al. (2018) publient, pour TRPO sur HalfCheetah, $t = -9{,}09$ et $p = 0{,}0016$ entre deux groupes de cinq essais du même code. Pineau précise que ces groupes ne sont pas triés. B1 cherche d'où vient ce chiffre, en deux temps écrits chacun avant son calcul.

Premier temps : leur algorithme. L'écart 4, écrit le 9 novembre avant tout entraînement de TRPO, lance trente essais de TRPO sur HalfCheetah-v5, graines aléatoires 0 à 29, un million de pas, avec les valeurs par défaut de sb3-contrib 2.9.2 et la même évaluation que E1. `sim/analysis/b1_trpo.py` applique les trois lectures de la section 5.9.6 aux courbes de TRPO et de PPO, avec les mêmes tirages que `e1_curve.py`. La règle de lecture était fixée d'avance. Si le 95e centile de $|\bar{t}|$ au hasard approche 9 avec TRPO, la dispersion de l'algorithme explique leur chiffre. Si seule la lecture empilée l'approche, leur calcul l'explique. Si aucune lecture ne l'approche, aucune des deux pistes ne l'explique : le rapport le dit.

Second temps : leur script. L'écart 6, écrit le 9 novembre à 23 h 55 après les résultats de `b1_trpo.py` et avant tout calcul avec le script, relève deux faits. Le couple publié $(t, p) = (-9{,}09\,;\ 0{,}0016)$ impose environ 3,4 degrés de liberté en test bilatéral. Il exclut la lecture des points empilés, à environ 198 degrés de liberté, qui donnerait un p de l'ordre de 10^{-16}. Il exclut aussi le t moyen converti en p, puisqu'un test de Welch sur cinq contre cinq a au moins 4 degrés de liberté. Seule la moyenne des p reste compatible. Le second fait est le script de 2017 (section 5.9.7). `sim/analysis/b1_script.py` applique ce calcul aux trente courbes de TRPO et aux trente de PPO, avec les mêmes tirages (générateur de graine aléatoire 2018, 10 000 tirages de dix essais coupés en cinq contre cinq, au hasard puis triés), pour $n = 10$ puis $n = 5$. Il rapporte aussi, par programme, le rapport de l'écart type moyen entre épisodes au dernier point à l'écart type des scores finaux entre graines aléatoires.

Différences dites d'avance. Nos épisodes d'évaluation suivent une politique déterministe, alors que les fichiers de progression de 2017 enregistrent les épisodes d'entraînement, joués par la politique stochastique. Nous avons vingt points d'évaluation, quand leurs courbes comptent une ligne par itération. Nous ne pouvons pas prouver que ce script a produit leur figure 5. Son nom (« de la même distribution ») et sa sortie (« t=..., p=... ») le rendent probable.

5.10.3 E2 : le classement change-t-il de tâche en tâche ?

E2 entraîne les quatre programmes sur les quatre tâches, dix graines aléatoires chacun. `sim/analysis/e2.py` lit le score final de chaque essai, puis, pour chaque tâche :

- l'IQM de chaque programme avec son intervalle bootstrap à 95 % (équations (5.2) et (5.3)) ;
- le classement par IQM ;
- pour chacune des six paires, la probabilité d'amélioration du mieux classé sur l'autre, avec son intervalle (équation (5.4)), avec le drapeau « tranchée » si l'intervalle exclut 0,5 ;
- un profil de performance : pour 25 seuils τ de 0 à 1,2, la part des essais dont le score, divisé par la meilleure IQM de la tâche, dépasse τ.

Le programme *en tête* d'une tâche est celui de plus grande IQM. Le classement *change* si le programme en tête n'est pas le même sur toutes les tâches. Le changement est *établi* quand le programme en tête d'une tâche est battu, de façon tranchée, par un autre programme sur une autre tâche. Le script compte ces changements établis : chaque triplet (tâche où X mène, tâche où X perd, programme qui bat X) en compte un. Ces détails ont été écrits le 10 novembre, après les résultats de PPO et de SAC et avant d'avoir regardé le moindre résultat de CrossQ (écart 7, lu avec la correction de l'écart 14, point b).

Avec la correction de Bonferroni. `sim/analysis/e2_bonferroni.py` refait les mêmes appels, dans le même ordre, après la même graine aléatoire : les rééchantillons bootstrap sont identiques à ceux de `e2.py`, seul le niveau des centiles change, à 99,79 %. Il compte les changements établis avec la même définition (écart 18).

5.10.4 E3 : deux programmes du même algorithme

`sim/analysis/e3.py` oppose PPO PyTorch à PPO JAX sur chaque tâche, avec les mêmes graines aléatoires et les scores à un million de pas. Il rapporte les deux IQM avec leurs intervalles, le rapport des IQM, la probabilité qu'un essai de PPO JAX batte un essai de PPO PyTorch avec son intervalle, le chevauchement des intervalles, puis le vainqueur selon la règle de la section 5.9.4.

5.10.5 E4 : les trois meilleurs sur dix

`sim/analysis/e4.py` calcule, pour chaque programme et chaque tâche, la moyenne et l'écart type des dix essais, puis des trois meilleurs, enfin le gonflement $\bar{x}_{3\text{ meilleurs}}/\bar{x}_{10}-1$. Une comparaison est une paire de programmes sur une tâche : il y en a 24. Elle *s'inverse* si le signe de l'écart des moyennes change quand chaque programme ne garde que ses trois meilleurs essais. Le script rapporte le gonflement médian, son étendue, la médiane du rapport des écarts types et le nombre d'inversions.

5.10.6 E5 : combien de graines aléatoires ?

`sim/analysis/e5.py` part des 24 comparaisons. Pour chacune, le bon ordre est le signe de l'écart des moyennes de tous les essais disponibles. La distribution observée des essais tient lieu de la vraie distribution des scores de l'algorithme. Pour $n = 3$, 5, 10 et 20, on tire 10 000 fois n essais avec remise pour chacun des deux programmes, puis l'on compte la part des tirages qui retrouvent le bon ordre :

$$\hat{P}_n = \frac{1}{N}\sum_{r=1}^{N} \mathbf{1}\Big[\operatorname{sign}\big(\bar{x}_n^{*(r)} - \bar{y}_n^{*(r)}\big) = \operatorname{sign}(\bar{x} - \bar{y})\Big], \qquad N = 10\,000. \tag{5.8}$$

Pour PPO PyTorch sur HalfCheetah, l'analyse à dix graines aléatoires tire parmi les trente essais de E1. Le script rapporte la médiane de $\hat{P}_n$ sur les 24 comparaisons, puis sur les comparaisons *proches*, dont l'écart relatif des moyennes $|\bar{x} - \bar{y}|/\max(|\bar{x}|, |\bar{y}|)$ reste sous 20 %. L'écart 20 a corrigé un défaut avant le calcul à vingt graines aléatoires : le script prenait pour bon ordre la moyenne des dix premiers essais, même avec vingt. Il prend désormais la moyenne de toutes les graines choisies. L'analyse à dix graines aléatoires n'a pas changé d'un octet.

5.10.7 E6 : la référence réglée

E6 voulait illustrer la faille F7, la référence mal réglée : l'écart qu'une « nouvelle méthode » peut s'attribuer quand elle est seule à être réglée. Le plan initial oppose, sur Hopper-v5, dix essais de PPO PyTorch avec ses valeurs par défaut (ceux de E2) à dix essais avec les réglages du Zoo (`e6.py`).

Le diagnostic. Le premier résultat a déclenché un diagnostic, ajouté après l'avoir vu (écart 2). `e6_diag.py` compare cinq groupes de dix essais : valeurs par défaut sur v5, Zoo avec l'erreur de facteur d'actualisation sur v5, Zoo corrigé sur v5, valeurs par défaut sur v4, Zoo corrigé sur v4. Trois comparaisons sont corrigées par Holm : l'erreur (Zoo corrigé contre Zoo avec erreur, sur v5), le test loyal sur la version du Zoo (Zoo corrigé contre valeurs par défaut, sur v4), le même test sur v5. Chaque comparaison rapporte le p de Welch, celui de Mann-Whitney et la probabilité d'amélioration avec son intervalle.

Le pipeline officiel. L'écart 5, écrit après le diagnostic et avant tout entraînement officiel, demande si le plafond observé au premier résultat, un score qui cesse de monter dès les premiers blocs d'entraînement, vient de la recette publiée ou de notre façon de l'appliquer. Le script `sim/zoo_official.sh` lance dix fois le script officiel du recueil, `python -m rl_zoo3.train --algo ppo --env Hopper-v4 --seed i`, pour i de 0 à 9, dans l'environnement séparé `sim/env-zoo`, sur un fil, sans évaluation pendant l'entraînement (`--eval-freq -1`). `e6_official.py` évalue ensuite chaque politique officielle avec notre protocole : dix épisodes déterministes, graines aléatoires d'évaluation 10 000 à 10 009, normalisation des observations chargée puis figée, sans normalisation de la récompense. Deux comparaisons, fixées d'avance et corrigées par Holm, opposent les politiques officielles à notre Zoo corrigé sur v4 puis à nos valeurs par défaut sur v4. La règle de lecture était écrite. Si l'officiel dépasse nettement 1 000, notre application était fautive. S'il plafonne aussi, la recette publiée fait moins bien que les valeurs par défaut dans cette version des bibliothèques. Le rapport le dit alors sans parler de référence mal réglée par nous.

5.10.8 E7 : la physique modifiée

`sim/analysis/e7.py` calcule, pour chaque essai s, le rapport des deux scores finaux

$$\rho_s = \frac{\texttt{perturbed_mean}_s}{\texttt{eval_mean}_s}, \tag{5.9}$$

puis, pour chaque programme et chaque tâche, l'IQM des ρ_s avec son intervalle bootstrap. Le verdict est une *perte* si l'intervalle reste sous 1, un *gain* s'il reste au-dessus, « aucun changement détectable » sinon. L'IQM des rapports diffère du rapport des IQM : la part de score gardée ne se lit pas sur deux classements de scores.

Le classement descriptif. `e7_rank.py` classe les quatre programmes sur chaque tâche par IQM du score, dans la physique d'entraînement puis dans la physique modifiée. Il dit si le programme en tête change. Ce complément, décidé après les résultats de E7, n'a pas de test (écart 11).

La physique par degrés. `sim/analysis/dose.py` lit les 160 fichiers de réévaluation (320 avec vingt graines aléatoires). Pour chaque programme, tâche, grandeur et facteur, il calcule l'IQM du rapport au score à 1,0 fois, avec son intervalle bootstrap, puis le même verdict que E7. Pour chaque tâche et chaque physique, il classe les quatre programmes par IQM du score, sans test. Il vérifie enfin que le score à 1,0 fois retrouve le score final de `run.py`. Il rapporte le plus grand écart relatif. La règle date de l'écart 12, point ii, écrit avant tout calcul. Le script lui-même a été écrit alors que 30 des 160 réévaluations existaient déjà, dont une avait été affichée : le commit qui le disait « écrit avant les résultats » se trompait (écart 14, point a). Le script n'a aucun paramètre libre.

5.10.9 B3 : la mesure change-t-elle le classement ?

`sim/analysis/b3.py` calcule trois mesures par essai sur les courbes enregistrées, sans aucun entraînement :

- le score final, au dernier point ;
- le meilleur score atteint le long de la courbe ;

- la moyenne de la courbe, l'aire sous la courbe divisée par sa longueur.

Pour chaque tâche et chaque mesure, il calcule l'IQM de chaque programme et le classement. Une paire de programmes *s'inverse* entre deux mesures si l'ordre de leurs IQM change. L'inversion est *établie* si, sous chacune des deux mesures, l'intervalle de la probabilité d'amélioration exclut 0,5, dans des sens opposés. Les courbes de PPO ont vingt points sur un million de pas, celles de SAC et de CrossQ quatre points sur 200 000 : la moyenne de la courbe ne couvre pas le même budget (écart 8).

5.10.10 La consolidation : vingt graines aléatoires et budget égal

L'écart 12, écrit le 15 novembre avant tout nouvel entraînement, ajoute trois volets avec le même code et les mêmes réglages :

- les graines aléatoires 10 à 19 pour les quatre programmes sur les quatre tâches, soit 150 nouveaux entraînements, puisque PPO PyTorch sur HalfCheetah avait déjà les siens ;
- la physique par degrés de la section 5.10.8 ;
- la lecture à budget égal de la section 5.5.

Le module `sim/analysis/runs.py` lit trois variables d'environnement, toutes inactives par défaut : `SEEDS` (par exemple `0:20`), `BUDGET=equal` et `SUFFIX`, ajouté au nom du fichier de sortie. L'écart 20, écrit le 21 novembre après la fin de tous les entraînements et avant tout calcul sur les graines aléatoires 10 à 19, fixe la liste des calculs, tous lancés par `make analysis` :

- sur les vingt graines aléatoires (suffixe `_s20`) : E2, E3, E5, E7, B3, la physique par degrés, le classement descriptif de E7 et la correction de Bonferroni de E2 ;
- E4 sur les seules graines aléatoires 10 à 19 (suffixe `_r10`), comme réplication indépendante ;
- E2 et B3 à budget égal, sur dix graines aléatoires (`_equal`) puis sur vingt (`_equal_s20`).

Aucun score des graines aléatoires 10 à 19 n'avait été lu avant l'écart 20, sauf le contrôle de la physique par degrés à 1,0 fois, qui compare chaque fichier à lui-même. Les deux derniers ajouts de la première liste ne figuraient pas dans l'écart 12 : ils appliquent aux vingt graines aléatoires des règles déjà écrites. Ils sont publiés quels qu'ils soient. Les résultats à dix graines aléatoires restent les résultats principaux.

5.11 La file d'attente et le calcul

La machine. Tous les entraînements ont tourné sur processeur, sur le Mac de la section 5.2, partagé avec d'autres projets. Un entraînement ou une réévaluation occupe un cœur (un seul fil de calcul). Le tableau 5.3 donne, par programme et par tâche, le nombre d'essais, les pas, la durée médiane et les heures de cœur.

Les lanceurs, dans l'ordre où ils ont servi. Plusieurs scripts ont lancé les essais. Chacun saute un essai dont la politique finale existe déjà.

Table 5.3. Calcul des expériences, relevé dans les fichiers de résultats : pour chaque programme et chaque tâche, nombre d'essais finis, pas d'entraînement par essai (colonne `step` de la dernière ligne), durée murale médiane d'un essai et somme des durées, en heures. La durée murale est la colonne `wall` de `sim/results/<programme>/<tâche>/seed<g>.csv` à la dernière évaluation (secondes depuis le lancement, évaluations comprises), et pour GRPO le dernier `wall` des pas d'entraînement de `sim/llm/results/<modèle>/mul2/<récompense>/seed<g>.jsonl`. Ce ne sont pas des heures-cœurs : chaque entraînement tournait sur un fil, PPO JAX sur deux au début (écart 4), GRPO sur le GPU Apple ou sur deux fils CPU, sur une machine partagée dont la charge variait ; les réévaluations de E7 à plusieurs intensités et les dix agents officiels du Zoo (E6) n'enregistrent pas leur durée et manquent ici. Les modèles PPO et TRPO comptent en vérité 1 024 000 pas (blocs de 51 200, écart 14 (d)). Le registre reconstruit `sim/logs/attempts_reconstructed.csv` compte 292 lancements pour les entraînements de la grille, de B1 et du diagnostic de E6, dont 250 aboutis (écart 14 (e)).

Programme	Tâche	essais	pas par essai	durée médiane (h)	total (h)
PPO PyTorch	HalfCheetah-v5	30	1 000 000	0,47	16,6
	Hopper-v4	10	1 000 000	0,58	7,0
	Hopper-v5	20	1 000 000	0,48	12,0
	Swimmer-v5	20	1 000 000	0,56	18,5
	Walker2d-v5	20	1 000 000	0,46	9,3
PPO JAX	HalfCheetah-v5	20	1 000 000	0,81	15,9
	Hopper-v5	20	1 000 000	0,85	17,2
	Swimmer-v5	20	1 000 000	0,75	14,5
	Walker2d-v5	20	1 000 000	0,85	17,6
SAC	HalfCheetah-v5	20	200 000	1,66	31,3
	Hopper-v5	20	200 000	0,81	17,0
	Swimmer-v5	20	200 000	0,62	13,1
	Walker2d-v5	20	200 000	0,85	17,0
CrossQ	HalfCheetah-v5	20	200 000	3,39	85,3
	Hopper-v5	20	200 000	3,75	87,1
	Swimmer-v5	20	200 000	3,15	68,9
	Walker2d-v5	20	200 000	2,62	52,3
TRPO	HalfCheetah-v5	30	1 000 000	0,09	2,6
PPO PyTorch, Zoo (gamma 0,99)	Hopper-v5	10	1 000 000	0,83	8,1
PPO PyTorch, Zoo corrigé	Hopper-v4	10	1 000 000	0,64	6,3
	Hopper-v5	10	1 000 000	0,73	7,7
Total des entraînements MuJoCo		400			525,4
GRPO, Qwen2.5 0,5B	mul2, récompense vraie	5	100	4,32	19,0
	mul2, récompense au hasard	5	100	5,61	29,4
GRPO, Llama 3.2 1B	mul2, récompense vraie	5	100	1,21	6,1
	mul2, récompense au hasard	4	100	1,58	7,1
Total GRPO (essais finis)		19			61,7

- `sim/drive.sh` lit une liste de tâches (`jobs.txt` par défaut) et en lance P à la fois par `xargs -P`. Un essai qui échoue est relancé une fois, dans un journal `.retry.log`. `drive2.sh` en est une copie pour `jobs2.txt`, les essais de E6 corrigés.
- `sim/relay.sh` change de file sans tuer d'essai : il attend la fin des essais en cours, puis relance la file principale à un entraînement JAX (environ deux cœurs) et la file TRPO de B1 à deux entraînements, pour rester à quatre cœurs.
- `sim/zoo_official.sh` attend la fin de la file TRPO, puis lance les dix entraînements officiels de E6 sur deux voies de cinq.
- `sim/queue.sh` lit `jobs_consolidation.txt`, la liste de l'écart 12 : 160 lignes `train` et 320 lignes `dose`. Il réserve chaque tâche par un `mkdir` atomique, ce qui permet à deux copies de tourner sans se gêner. Il tient le registre `logs/attempts.jsonl`. Un arrêt par signal, code de sortie de 128 ou plus, n'est pas relancé (écart 14, point g).
- `sim/worker.sh` est `queue.sh` avec un plafond : il lit `workers.max` avant chaque tâche et s'en va si trop de copies tournent (écart 15).
- `sim/scale.sh` décide ce plafond toutes les heures. Il somme le calcul pris par les autres programmes de la machine (`%cpu` de `ps`), en déduit les cœurs libres moins un de marge. Il accorde au projet au plus `MAX_CORES` cœurs, l'essai GRPO sur processeur en prenant deux. Il lance de nouveaux `worker.sh` si le plafond monte. Il n'en lance aucun sans le surveillant. Ses décisions sont dans `logs/scale.log`.
- `sim/watchdog.sh`, le surveillant, mesure toutes les 20 minutes l'espace disque libre et la croissance de la mémoire d'échange (*swap*) depuis son lancement. Il arrête tous les lanceurs et tous les essais si le disque passe sous `MIN_FREE_GB` gigaoctets (10 par défaut) ou si la mémoire d'échange a grossi de plus de 8 Go. Il inscrit chaque tâche arrêtée au registre et libère sa réservation.

Le nombre de cœurs au fil des jours. Jusqu'au 15 novembre, le nombre d'essais simultanés était réglé à la main : quatre cœurs pour le projet à partir du 9 novembre, puis deux copies de `queue.sh` pour la consolidation. L'écart 15 (15 novembre) a relevé le plafond à huit cœurs, décidé chaque heure par `scale.sh`. L'écart 19 (19 novembre) l'a ramené à deux cœurs pour tout le projet, essai GRPO sur processeur compris. Le nombre de tâches parallèles ne change aucun résultat : chaque essai est déterminé par sa graine aléatoire et tourne sur un seul fil.

Les arrêts et les relances. Le protocole voulait qu'un plantage soit relancé une fois et noté. Le registre reconstruit `logs/attempts_reconstructed.csv` compte 292 lancements pour les 250 entraînements antérieurs à la consolidation : 42 tentatives arrêtées (2 seulement déclarées), aucune plantée (écart 14, point e). Les arrêts ont trois causes : un arrêt général le 8 novembre, probablement celui de l'incident mémoire du pilote GRPO, un arrêt demandé le 9 novembre pour une machine surchargée, puis quatre arrêts pour rester à quatre cœurs. Chaque arrêt a frappé tous les essais en cours à la fois. Aucun essai n'a été relancé ni écarté après que son score final était connu, ni pour une raison liée à son score. Pour PPO PyTorch et PPO JAX, les relances retrouvent à l'identique les points d'évaluation des tentatives arrêtées. Pour SAC et CrossQ, les courbes arrêtées ont été écrasées : cette vérification est impossible. Les arrêts suivants du surveillant, les 12, 15 et 16 novembre, sont décrits aux écarts 10, 14 et 16 (chapitre 9).

La chaîne jusqu'aux documents. Les essais écrivent leurs fichiers CSV et JSON. Les scripts de `sim/analysis/` en tirent les fichiers JSON de résultats. `make verify` recalcule chaque analyse et compare les fichiers à l'octet près, puis `docs/check_numbers.py` et `article/check_article.py` vérifient chaque nombre des documents contre ces fichiers. Le chapitre 8 décrit cette chaîne et la dessine (figure 8.1).

5.12 Ce qui a été écrit avant les résultats, ce qui l'a été après

Un protocole écrit d'avance protège contre une analyse choisie après avoir vu les données. Il ne protège que s'il dit honnêtement ce qui a été décidé après. Voici le classement, écart par écart, des décisions qui touchent une analyse.

Écrit avant tout résultat concerné.

- le protocole lui-même (8 novembre) : E1, E2, E3, E4, E5, E6, E7, leurs statistiques, la règle du vainqueur établi ;
- le budget de SAC et CrossQ ramené à 200 000 pas (écart 1), avant le premier entraînement ;
- B1 avec TRPO et ses trois lectures (écart 4), avant tout entraînement TRPO ;
- le pipeline officiel de E6 et sa règle de lecture (écart 5), avant tout entraînement officiel ;
- le calcul du script de Henderson (écart 6), avant ce calcul ;
- les détails de E2, E4 et E5 (écart 7) et le plan de B3 (écart 8), avant d'avoir regardé le moindre score de CrossQ, alors que 5 à 7 essais CrossQ sur 40 étaient finis, au lieu des 3 qu'écrivait l'écart 7 (écart 14, point b) ;
- la consolidation (écart 12), avant tout nouvel entraînement ;
- la correction de Bonferroni (écart 18), après E2 mais avant ce calcul ;
- la liste des calculs à vingt graines aléatoires (écart 20), avant tout calcul sur les graines 10 à 19.

Écrit après avoir vu un résultat.

- le diagnostic de E6 (écart 2), déclenché par le premier résultat de E6 ;
- la seconde statistique de E1 (écart 3), après les résultats de `e1.py` ;
- le classement descriptif de E7 (écart 11), après les résultats de E7, sans test ;
- la graine aléatoire du générateur global des bootstraps (écart 13), correction technique sans effet sur les verdicts ;
- les sept corrections de l'écart 14 : la date du script de la physique par degrés, les comptes des écarts 7 et 8, la version de Stable-Baselines3, le nombre exact de pas, les relances non déclarées, les graines aléatoires nommées dans chaque script, le registre des tentatives.

Aucune de ces décisions n'a retiré un essai, changé un réglage ou changé la règle du vainqueur établi. Les écarts sans effet sur l'analyse (organisation, interruptions, nombre de cœurs, batterie) sont au chapitre 9.

CHAPITRE 6

Les expériences : résultats

Ce chapitre donne le résultat de chaque expérience décrite au chapitre 5. Chaque section suit le même ordre : la question, le plan en une phrase, le résultat avec ses intervalles, puis ce qu'il signifie et ce qu'il ne dit pas.

Conventions de lecture. Les résultats principaux sont ceux à dix graines aléatoires (0 à 9), annoncés par le protocole. Les résultats à vingt graines aléatoires (0 à 19) et à budget égal viennent ensuite. Chaque divergence est dite. Un score est le retour moyen des dix épisodes d'évaluation finaux d'un essai (section 5.7). Un score de programme est une moyenne interquartile (IQM) de ces scores sur les graines aléatoires. Un intervalle écrit entre crochets, comme $[2\,268, 3\,369]$, est un intervalle bootstrap à 95 %. Un écart est *établi* quand il satisfait la règle du vainqueur établi (section 5.9.4) : intervalles des IQM disjoints, ou intervalle de la probabilité d'amélioration qui exclut 0,5. Les tableaux sont produits par le script `report/tools/tables.py` à partir des fichiers JSON de `sim/analysis/`. Les nombres du texte viennent des mêmes fichiers, avec l'arrondi de `docs/conclusions.md`.

6.1 E1 : même code, autre graine aléatoire

Question et plan. Deux groupes d'essais du même programme, qui ne diffèrent que par leurs graines aléatoires, peuvent-ils passer pour deux algorithmes différents ? E1 entraîne trente fois PPO PyTorch sur HalfCheetah-v5, un million de pas chacun, graines aléatoires 0 à 29, puis compare par un test de Welch des groupes tirés parmi ces trente essais (section 5.10.1).

Une dispersion large, qui n'a pas la forme d'une cloche. La moyenne des trente scores vaut 1 688, l'écart type 568, soit 34 % de la moyenne. La médiane vaut 1 580. Vingt-sept essais finissent entre 1 138 et 1 999. Trois décollent au-delà de 3 000, jusqu'à 3 430 (figure 6.1). Ces trois essais tirent la moyenne au-dessus de la médiane. Le test de Shapiro-Wilk rejette nettement la loi normale ($p = 4 \times 10^{-7}$). Un test t suppose des moyennes de groupe à peu près normales : avec cinq essais tirés dans une telle distribution, l'hypothèse est fragile.

Le hasard trompe rarement le test, le tri le trompe souvent. Le tableau 6.1 et la figure 6.2 donnent la part des 10 000 tirages où le test de Welch déclare différents, au seuil de 5 %, deux groupes issus du même programme. Chacune de ces déclarations est un faux positif.

- Trois essais contre trois, tirés au hasard : 1,6 % des tirages.
- Cinq contre cinq : 1,2 %.
- Dix contre dix : 1,7 %.

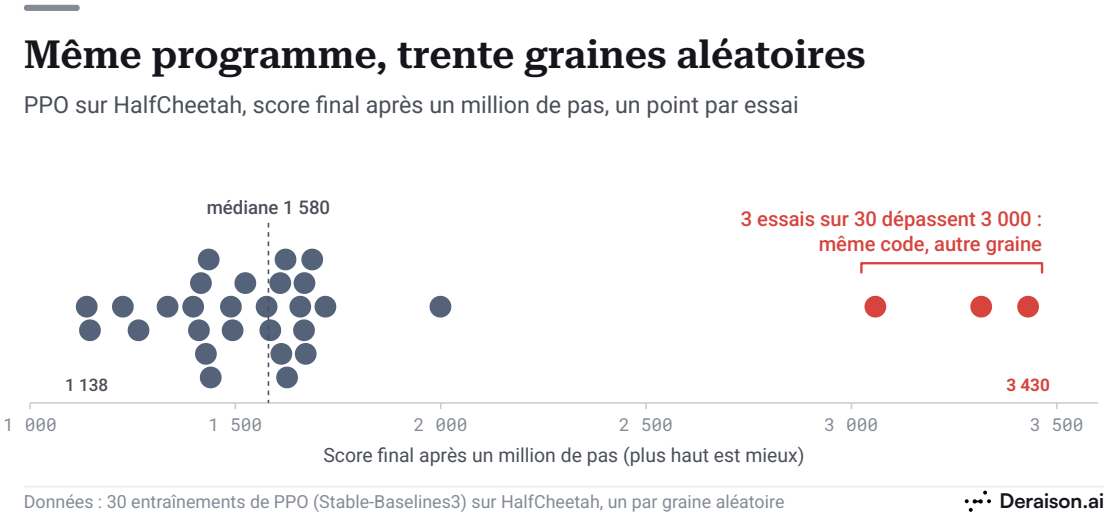


FIGURE 6.1. Trente entraînements du même programme sur la même tâche : PPO écrit en PyTorch (Stable-Baselines3) sur HalfCheetah-v5, un million de pas chacun. Chaque point est un essai, placé à son score d'évaluation final, la moyenne des retours de dix épisodes joués sans hasard sur des graines aléatoires jamais vues à l'entraînement. Seule la graine aléatoire d'entraînement change d'un point à l'autre. Moyenne 1 688, médiane 1 580 (ligne pointillée). Vingt-sept essais finissent entre 1 138 et 1 999, trois au-delà de 3 000.

- Les cinq meilleurs contre les cinq pires d'un tirage de dix essais : 31,0 %.

Tirés au hasard, les groupes trompent le test moins souvent que le seuil nominal de 5 %. Le test est conservateur sur cette distribution asymétrique. L'explication la plus probable : un essai hors norme gonfle la variance du groupe qui le reçoit en même temps que sa moyenne. Triés, les groupes le trompent près d'une fois sur trois. La valeur p médiane de la coupe triée vaut 0,09.

Le hasard trompe l'œil. Les groupes tirés au hasard trompent rarement le test, mais leurs moyennes diffèrent beaucoup. L'écart relatif des moyennes, rapporté à la plus petite des deux, vaut en médiane 17 % pour trois contre trois, 17 % pour cinq contre cinq et 11 % pour dix contre dix. Une fois sur vingt, il dépasse 62 %, 47 % et 33 %. Deux groupes de cinq essais tirés au hasard dessinent souvent deux courbes bien séparées. Dans la coupe triée, l'écart médian atteint 42 %.

Garder les trois meilleurs essais sur dix. Sur 10 000 tirages de dix essais, la moyenne des trois meilleurs vaut 2 249 en moyenne, contre 1 692 pour les dix : un gonflement de 33 %. L'expérience E4 étend cette mesure aux quatre programmes (section 6.5).

La statistique de 2018, hors de portée de PPO. Henderson et al. (2018) testent des courbes d'apprentissage entières. La seconde statistique de E1, ajoutée après les résultats précédents (écart 3), refait leur t moyen sur nos trente courbes : un test de Welch à chacun des vingt points d'évaluation, puis la moyenne des t (section 5.9.6). Une fois sur vingt, la valeur absolue du t moyen dépasse 1,4 quand les groupes de cinq sont tirés au hasard, 2,5 quand ils sont triés par score final. Aucun des 20 000 tirages n'atteint 9. Le test au seul dernier point retrouve le résultat des scores finaux : environ 31 % de faux positifs pour les groupes triés.

Table 6.1. Expérience E1 : trente entraînements de PPO PyTorch (Stable-Baselines3) sur HalfCheetah-v5, même code et mêmes hyperparamètres, graines aléatoires 0 à 29 ; score final : moyenne de 10 épisodes d'évaluation sur des graines jamais vues à l'entraînement. Partie haute : statistiques des 30 scores ; le test de Shapiro-Wilk rejette la normalité (trois essais décollent au-delà de 3 000). Partie médiane : 10 000 tirages par ligne (générateur `default_rng(2018)`), test de Welch sur les scores finaux de deux groupes du même programme : toute différence déclarée est un faux positif. L'écart apparent divise la différence des deux moyennes par la plus petite des deux. Les groupes tirés au hasard trompent le test moins souvent que le seuil de 5 % ; **en gras**, les groupes triés (5 meilleurs contre 5 pires sur 10 essais tirés), qui le trompent près d'une fois sur trois. Partie basse : moyennes sur 10 000 tirages de 10 essais. Source : `sim/analysis/e1.json` (`sim/analysis/e1.py`).

PPO PyTorch, HalfCheetah-v5, $n = 30$			
Scores finaux des 30 essais (évaluation à 1 million de pas)			
Moyenne ; écart type	1 688 ; 568	Coef. de variation	34 %
Médiane	1 580	Minimum ; maximum	1 138 ; 3 430
Shapiro-Wilk, p	4×10^{-7}	IQM	voir tableau 6.3

Groupes comparés	Faux positifs ($p < 0{,}05$)	Écart apparent médian	Écart dépassé 1 fois sur 20
3 contre 3, au hasard	1,6 %	17 %	62 %
5 contre 5, au hasard	1,2 %	17 %	47 %
10 contre 10, au hasard	1,7 %	11 %	33 %
5 meilleurs contre 5 pires, sur 10	**31,0 %**	42 %	non calculé (p médian 0,09)
Garder les 3 meilleurs sur 10	moyenne des 3 2 249	moyenne des 10 1 692	gonflement 33 %

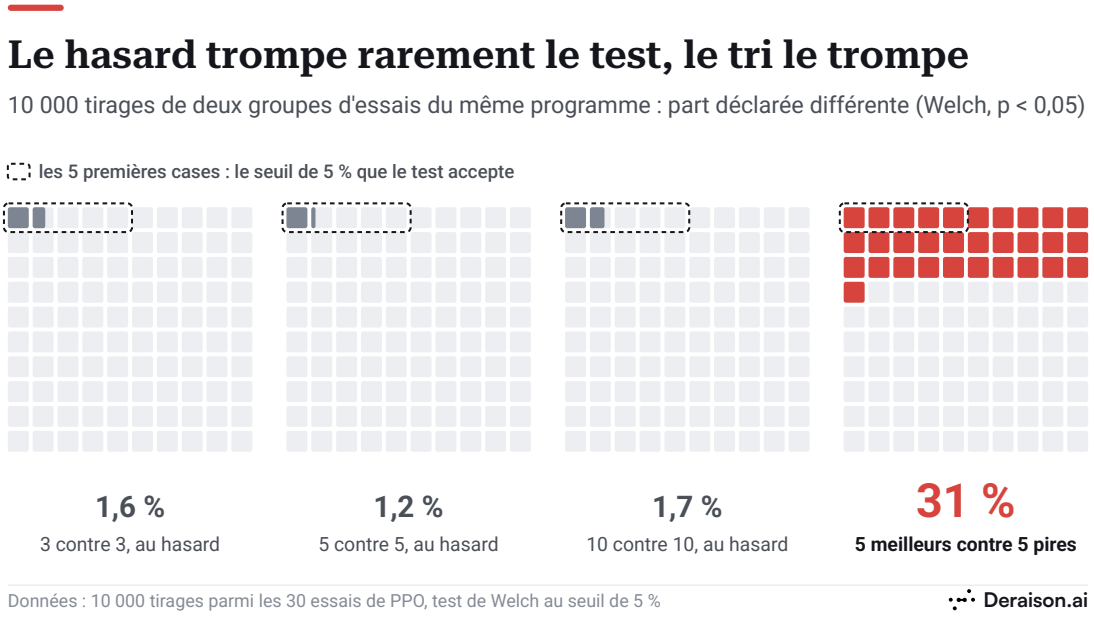


Figure 6.2. Part des 10 000 tirages où le test de Welch déclare différents, au seuil de 5 %, deux groupes d'essais du même programme (PPO PyTorch sur HalfCheetah, trente essais qui ne diffèrent que par leur graine aléatoire). Chaque grille compte 100 cases, une case valant un centième des tirages. Les cases encadrées en pointillé figurent le seuil de 5 %. Tirés au hasard, trois contre trois, cinq contre cinq ou dix contre dix, les groupes trompent le test moins souvent que ce seuil. Les cinq meilleurs opposés aux cinq pires d'un tirage de dix le trompent près d'une fois sur trois.

Ce que E1 établit, ses limites. Sur ces trente essais, le hasard des graines aléatoires trompe l'œil bien plus souvent que le test. Un test de Welch sur des groupes tirés au hasard reste prudent, même avec trois essais par groupe. Le tri des essais change ce hasard en faux positifs. La sélection des meilleurs gonfle la moyenne d'un tiers. E1 se limite à un programme et une tâche. La forme de la distribution, avec quelques essais qui décollent, dépend des deux. Sur une distribution plus symétrique, le test au hasard serait plus proche de son seuil nominal.

6.2 B1 : l'énigme du t de 9

Question. La figure 5 de Henderson et al. (2018) oppose deux groupes de cinq essais de TRPO sur HalfCheetah, avec le même code et les mêmes réglages. Elle rapporte $t = -9{,}09$ et $p = 0{,}0016$. Pineau précise en 2018 que les groupes ne sont pas triés. Or E1 montre qu'avec PPO, des groupes tirés au hasard n'approchent jamais un t moyen de 9. D'où vient ce chiffre ? Trois pistes restaient ouvertes : la dispersion propre à TRPO, une autre façon de lire leur légende, ou un autre calcul (section 5.10.2).

Première piste : TRPO disperse davantage, sans approcher 9. Les trente essais de TRPO, entraînés avec les valeurs par défaut de sb3-contrib, dispersent bien plus que ceux de PPO. Leur moyenne finale vaut 1 951, leur écart type 930, soit 48 % de la moyenne, contre 34 % pour PPO. Pourtant, avec des groupes de cinq tirés au hasard, le t moyen sur la courbe ne dépasse 1,4 qu'une fois sur vingt, comme pour PPO. Avec des groupes triés, il dépasse 1,9 une fois sur vingt. La règle fixée d'avance (écart 4) écarte la dispersion de l'algorithme comme explication.

Deuxième piste : les points empilés, un calcul fautif qui reste court. La troisième lecture empile les cent points des cinq courbes de chaque groupe, comme s'ils étaient des observations indépendantes. Une fois sur vingt, son $|t|$ dépasse 5,0 avec des groupes tirés au hasard, 6,3 avec des groupes triés. Aucun tirage n'atteint 9. Cette lecture a un défaut plus grave que son échec : elle déclare une différence dans 53 % des tirages au hasard avec TRPO, 23 % avec PPO, alors que les deux groupes font tourner le même code. Les vingt points d'une courbe ne sont pas indépendants : un essai qui part bien reste souvent devant.

Le p publié trahit le calcul. Un test de Welch bilatéral qui donne $t = 9{,}09$ et $p = 0{,}0016$ a environ 3,4 degrés de liberté. Ce couple exclut les points empilés, qui ont environ 198 degrés de liberté et donneraient un p de l'ordre de 10^{-16}. Il exclut aussi le t moyen converti en p : deux groupes de cinq donnent au moins 4 degrés de liberté (section 5.9.5). Seule une moyenne de valeurs p reste compatible avec le couple publié.

Le script de 2017. Peter Henderson a publié en juin 2017 un script au nom parlant, `from_same_dist.py`, « de la même distribution » (Henderson 2017). La section 5.9.7 en donne le calcul exact. En résumé :

1. il fait la moyenne des courbes des cinq graines aléatoires de chaque groupe, point par point ;
2. à chaque point, il compare les deux courbes moyennes par un test de Welch dont la variance est celle *des épisodes* d'une même itération, divisée par le nombre d'épisodes ;

3. il rapporte la moyenne des t et la moyenne des p.

La variabilité entre graines aléatoires, celle que la figure devait mesurer, n'entre jamais dans le test. Le script compte les épisodes au lieu des entraînements indépendants.

Le script retrouve le t de 9 sur nos essais de TRPO. Appliqué à nos trente courbes de TRPO avec $n = 10$ épisodes, selon un plan écrit avant le calcul (écart 6), le script donne les résultats du tableau 6.2 :

- Avec des groupes tirés au hasard, la valeur absolue du t moyen atteint 9,09 dans 38 % des tirages. Sa médiane vaut 7,1. Le test déclare une différence dans 43 % des tirages.
- Avec des groupes triés, 70 % des tirages atteignent 9,09.
- Avec $n = 5$, l'analyse de sensibilité, 23 % des tirages au hasard atteignent 9,09. Le test y déclare une différence dans 25 % des tirages.

Appliqué à nos trente courbes de PPO, le même calcul n'atteint jamais 9,09. Une fois sur vingt, son $|t|$ moyen dépasse 3,6 au hasard. La figure 6.3 montre ce contraste, la figure 6.4 montre les distributions complètes.

Table 6.2. Expérience B1 : la statistique de la figure de 2018 ($t = -9{,}09$, Henderson et al., TRPO sur HalfCheetah, deux groupes de cinq essais non triés) recalculée sur nos trente essais de PPO (ceux de E1) et nos trente essais de TRPO (sb3-contrib, valeurs par défaut), HalfCheetah-v5, un million de pas, 20 points d'évaluation. Chaque ligne : 10 000 tirages de 10 essais parmi 30, coupés en deux groupes de 5, au hasard ou triés par score final (générateur `default_rng(2018)`). Trois lectures de « t moyen sur toute la courbe » (écart 4 de `sim/SIM_PROTOCOL.md`, écrit avant tout entraînement TRPO) : t de Welch à chaque point puis moyenne, convertie en p avec les degrés de liberté moyens ; moyenne des p ; les 5 essais $\times$ 20 points de chaque groupe empilés comme 100 observations indépendantes. Puis le script `from_same_dist.py` publié par Henderson en 2017 (écart 6) : test de Welch à chaque itération sur l'écart type entre épisodes d'un même point, moyenne des t et des p ; la variation entre graines n'y entre pas. Colonnes : part des tirages déclarés différents ($p < 0{,}05$, faux positifs puisque les deux groupes exécutent le même code), 95e centile de $|t|$, part des tirages où $|t|$ atteint 9,09. Parts en pourcentage arrondies à l'unité, à un chiffre significatif sous 1 %. **En gras**, les seules lignes où $|t|$ atteint 9,09 : le script de 2017 appliqué à TRPO, dont l'écart type entre épisodes ne vaut que 7 % de l'écart type entre graines (PPO : 30 %). Coefficient de variation des scores finaux : TRPO 48 %, PPO 34 %. Sources : `sim/analysis/b1_trpo.json`, `sim/analysis/b1_script.json` (la ligne PPO t moyen reprend le calcul de `sim/analysis/e1_curve.json`).

		Groupes tirés au hasard			Groupes triés		
Algorithme	Calcul du t	faux pos. (%)	$\lvert t\rvert$, 95e c.	$\lvert t\rvert \geq 9{,}09$ (%)	faux pos. (%)	$\lvert t\rvert$, 95e c.	$\lvert t\rvert \geq 9{,}09$ (%)
PPO	t moyen sur la courbe (graines)	0,2	1,4	0	6	2,5	0
	p moyen sur la courbe (graines)	0	1,4	0	0,2	2,5	0
	points empilés (100 par groupe)	23	3,0	0	86	4,4	0
	script de Henderson, $n = 10$ épisodes	2	3,6	0	7	5,3	0
	script de Henderson, $n = 5$	0,1	2,5	0	0,9	3,8	0
TRPO	t moyen sur la courbe (graines)	0,04	1,4	0	1	1,9	0
	p moyen sur la courbe (graines)	0	1,4	0	0,06	1,9	0
	points empilés (100 par groupe)	53	5,0	0	84	6,3	0
	script de Henderson, $n = 10$ épisodes	**43**	**22,3**	**38**	**51**	**28,2**	**70**
	script de Henderson, $n = 5$	**25**	**15,8**	**23**	**31**	**19,9**	**49**

Pourquoi TRPO seul y arrive. Le dénominateur du script est l'écart type entre épisodes. Le bon dénominateur est l'écart type entre graines aléatoires. Notons ρ le rapport du premier au second, mesuré au dernier point d'évaluation. Plus ρ est petit, plus le t du script dépasse celui d'un test sur les graines aléatoires, d'un facteur proche de $1/\rho$. Chez notre TRPO, l'écart type entre les épisodes d'un même agent ne vaut que 7 % de l'écart type entre agents. Chez notre PPO, il en vaut 30 %. Un agent TRPO joue presque le même score à chaque épisode, alors que deux agents TRPO diffèrent beaucoup. C'est le cas où compter les épisodes au lieu des graines aléatoires fabrique des t énormes.

Ce qui reste incertain. Nous ne pouvons pas prouver que ce script a produit la figure 5. Son nom, sa sortie (« t=..., p=... ») et sa compatibilité avec le couple (t, p) publié le rendent probable. Nos conditions diffèrent des leurs. Nous l'avions dit avant le calcul. Nos épisodes d'évaluation sont joués par une politique déterministe, alors que leurs fichiers de progression enregistrent les épisodes d'entraînement d'une politique stochastique, qui varient sans doute davantage. Nous avons vingt points d'évaluation au lieu de leurs itérations. Notre TRPO est celui de sb3-contrib. Leur code était différent. L'ordre de grandeur se retrouve malgré ces différences.

Ce que B1 établit. Le t de 9 de 2018 n'exige aucun tri des essais pour apparaître : un test qui compte les épisodes au lieu des graines aléatoires suffit à le produire sur nos propres essais de TRPO. La faille F11, même code, autre graine aléatoire, est plus profonde que la figure ne le montrait : le hasard des graines aléatoires a trompé le test même qui devait le mesurer. La figure 6.5 rassemble E1 et B1 face au chiffre publié.



FIGURE 6.3. Deux groupes de cinq essais tirés au hasard parmi trente, 10 000 fois, pour chaque algorithme et chaque façon de calculer le t moyen le long de la courbe d'apprentissage. Chaque grille compte 100 cases, une case valant un centième des tirages. Elle colore la part des tirages où le t moyen, signe oublié, atteint 9,09, la valeur publiée par Henderson et ses coauteurs en 2018 pour TRPO sur HalfCheetah. Seul le script qu'ils ont publié en 2017, qui teste la dispersion entre épisodes au lieu de la dispersion entre graines aléatoires, y parvient, appliqué à TRPO : 38 % des tirages. PPO avec ce script, PPO et TRPO avec notre test sur les graines aléatoires, TRPO avec les points empilés : aucun tirage.

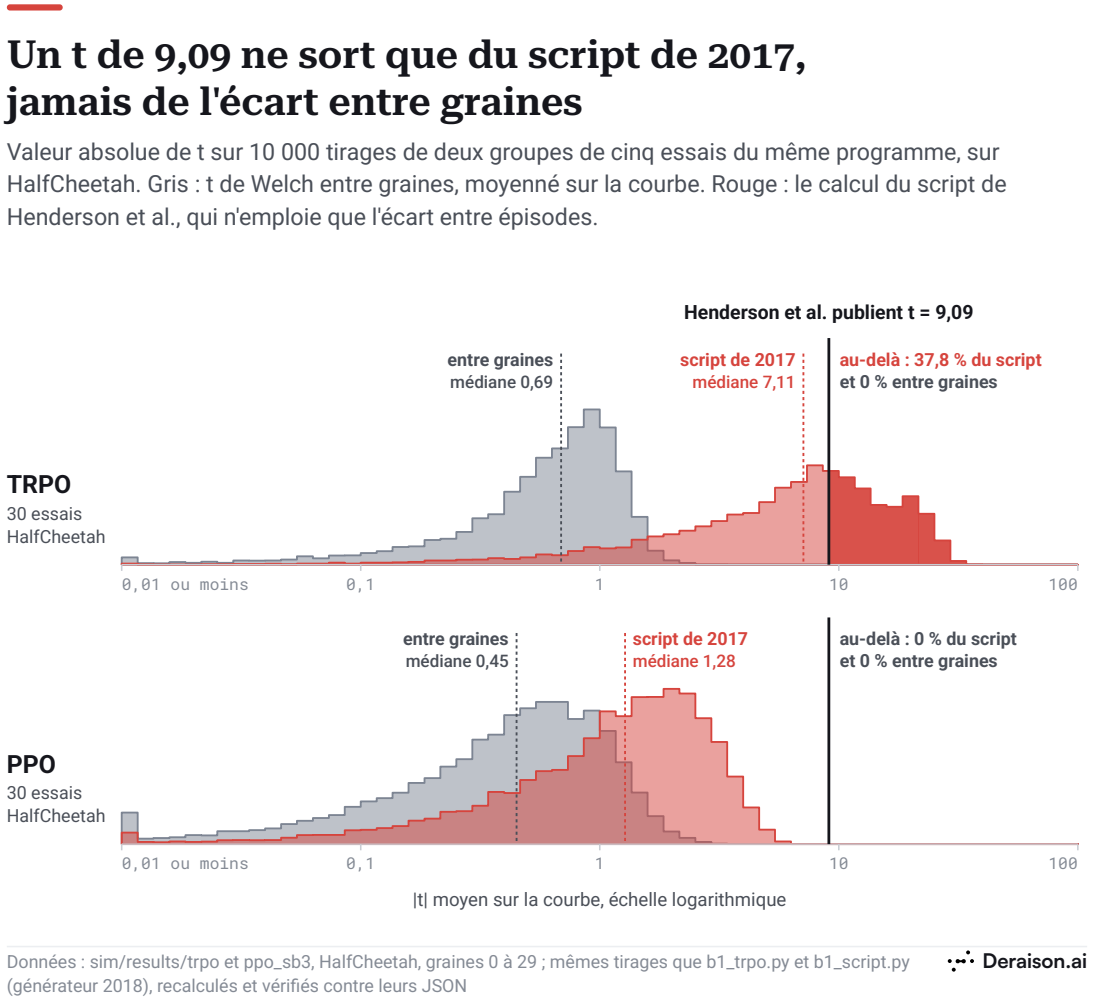


FIGURE 6.4. Distribution de la valeur absolue du t moyen sur 10 000 tirages de deux groupes de cinq essais pris au hasard parmi trente, sur HalfCheetah, pour TRPO (en haut) et pour PPO PyTorch (en bas), en échelle logarithmique. En gris, le test de Welch fait entre graines aléatoires à chaque point de la courbe, puis moyenné. En rouge, le calcul du script de Henderson (2017), qui fait la moyenne des courbes de chaque groupe et teste la dispersion entre les dix épisodes d'évaluation. La ligne verticale marque 9,09, la valeur publiée en 2018. Seul le script appliqué à TRPO la dépasse, dans 38 % des tirages.

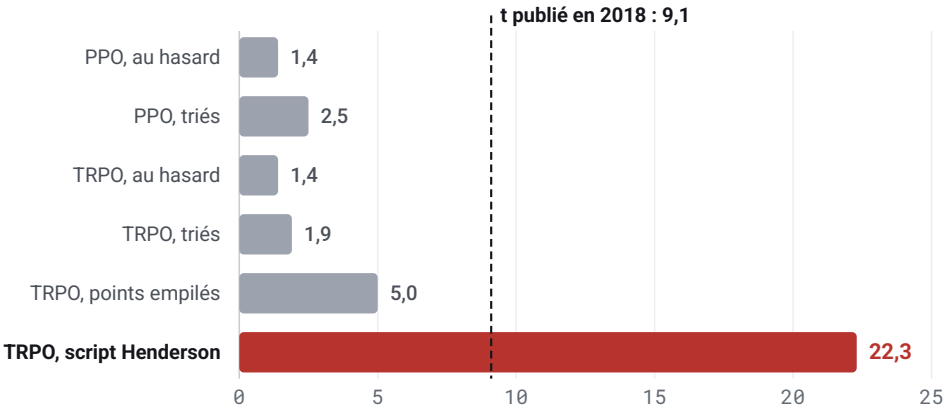
6.3 E2 : le programme en tête change de tâche en tâche

Question et plan. Henderson et al. (2018) montraient que l'algorithme en tête sur une tâche cessait de l'être sur une autre. E2 refait la comparaison avec quatre programmes de 2026 (PPO PyTorch, PPO JAX, SAC, CrossQ) sur les quatre tâches, dix graines aléatoires chacun, au budget de chaque programme : un million de pas pour PPO, 200 000 pour SAC et CrossQ (section 5.10.3).

Dix graines aléatoires : quatre tâches, trois programmes en tête. Le tableau 6.3 donne l'IQM de chaque programme sur chaque tâche avec son intervalle. La figure 6.6 rapporte chaque IQM à la meilleure de sa tâche.

Seul le script de Henderson dépasse le t de 2018 : la dispersion des graines n'y suffit pas

HalfCheetah, deux groupes de cinq essais parmi trente : 95e centile, sur 10 000 tirages, de la valeur absolue du t moyen le long de la courbe



Données : sim/analysis/e1_curve.json, b1_trpo.json, b1_script.json (docs/figures/data/e1_henderson_fr.csv)

Deraison.ai

FIGURE 6.5. Nos trente essais de PPO et nos trente essais de TRPO sur HalfCheetah, comparés au chiffre publié en 2018. Pour chaque façon de répartir dix essais en deux groupes de cinq, au hasard ou triés par score final, la barre donne la valeur absolue du t moyen le long de la courbe d'apprentissage dépassée une fois sur vingt, sur 10 000 tirages. La barre des points empilés traite les cent points des courbes de chaque groupe comme des observations indépendantes. La barre du script de Henderson applique son calcul de 2017, qui teste la dispersion entre épisodes, à nos essais de TRPO. La barre de Henderson et ses coauteurs est la valeur qu'ils publient sur leurs propres essais.

- HalfCheetah : SAC mène avec 5 191 [4 431, 5 601], devant CrossQ, 4 801 [2 427, 7 815]. Les deux PPO suivent loin derrière, 1 407 pour PPO PyTorch et 1 254 pour PPO JAX.
- Hopper : PPO JAX mène avec 3 064, devant PPO PyTorch (2 877), CrossQ (2 302) et SAC (1 308).
- Walker2d : CrossQ mène avec 5 414 [4 994, 5 627], loin devant PPO PyTorch (2 158), PPO JAX (1 272) et SAC (1 236).
- Swimmer : PPO JAX mène avec 112, devant CrossQ (110) et PPO PyTorch (107). SAC reste à 47.

Un changement établi, une tête rarement nette. Le programme en tête n'est pas le même partout. Le changement est établi selon la règle du protocole : SAC, en tête sur HalfCheetah, est battu de façon tranchée sur Hopper par les deux PPO, sur Walker2d par CrossQ et PPO PyTorch, sur Swimmer par les trois autres. `e2.py` compte 14 changements établis. En revanche, le programme en tête ne bat de façon tranchée tous les autres que sur Walker2d. Sur les trois autres tâches, dix graines aléatoires ne le séparent pas de son suivant : SAC de CrossQ sur HalfCheetah, PPO JAX de PPO PyTorch sur Hopper, PPO JAX de CrossQ sur Swimmer. L'intervalle de CrossQ sur HalfCheetah, de 2 427 à 7 815, montre combien ses essais varient sur cette tâche.

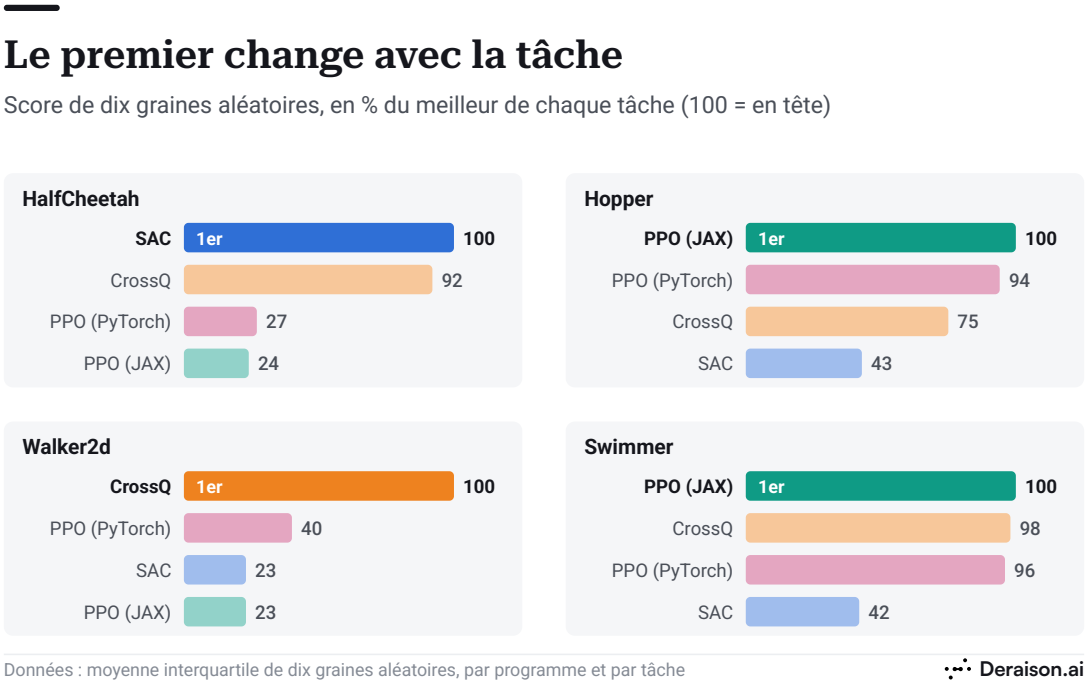


FIGURE 6.6. Score de chaque programme rapporté au meilleur de sa tâche, sur dix graines aléatoires : la barre à 100 désigne le programme en tête. Le score d'un programme est la moyenne interquartile (la moyenne des six scores du milieu sur dix) de ses scores d'évaluation finaux. PPO s'entraîne un million de pas, SAC et CrossQ 200 000. Le programme en tête change : SAC sur HalfCheetah, PPO écrit en JAX sur Hopper et Swimmer, CrossQ sur Walker2d.

Avec la correction de Bonferroni. E2 calcule 24 intervalles de probabilité d'amélioration sans correction. Portés au niveau simultané de 95 %, soit 99,79 % chacun (écart 18), ils laissent 12 changements établis au lieu de 14 (tableau 6.4). Deux paires perdent leur statut : PPO JAX contre CrossQ sur Hopper, PPO PyTorch contre SAC sur Walker2d. Le verdict de E2 résiste à la correction.

Les courbes d'apprentissage. La figure 6.7 montre les courbes d'évaluation des seize couples. Elle rappelle ce qu'un classement final cache : les courbes de SAC et de CrossQ s'arrêtent à 200 000 pas, celles de PPO continuent jusqu'à un million. La lecture à budget égal, plus bas, compare les quatre programmes au même point de ces courbes.

Vingt graines aléatoires. Avec les graines aléatoires 0 à 19, le programme en tête change toujours avec la tâche : CrossQ mène sur HalfCheetah et Walker2d, PPO JAX sur Hopper et Swimmer. Il reste 10 changements établis, 6 avec la correction de Bonferroni. Trois divergences avec les dix graines aléatoires sont à dire.

- Sur HalfCheetah, SAC cède la tête à CrossQ : son IQM passe de 5 191 à 4 214, celle de CrossQ de 4 801 à 4 694. L'écart entre les deux n'est établi ni à dix ni à vingt graines aléatoires.
- Sur Hopper, CrossQ monte de la troisième à la deuxième place (2 302 puis 2 792). L'avance de PPO JAX sur lui n'est plus établie.
- Le programme en tête bat désormais de façon tranchée tous les autres sur deux tâches, Walker2d et Swimmer.



FIGURE 6.7. Courbes d'évaluation des quatre programmes (PPO PyTorch, PPO JAX, SAC, CrossQ) sur les quatre tâches (une tâche par colonne). En abscisse, les pas d'entraînement, un point tous les 50 000 pas : jusqu'à un million pour PPO, 200 000 pour SAC et CrossQ. En ordonnée, le retour moyen de dix épisodes d'évaluation joués sans hasard. Le trait plein est la moyenne interquartile des dix graines aléatoires, la bande son intervalle bootstrap à 95 %, les traits pâles les essais individuels.

TABLE 6.3. Expérience E2 : moyenne interquartile (IQM) du score final de chaque programme sur chaque tâche MuJoCo (Gymnasium 1.1.1), avec son intervalle de confiance à 95 % par bootstrap stratifié (rliable, 2 000 rééchantillons, `np.random.seed(2026)`). Score final : moyenne de 10 épisodes d'évaluation sur des graines jamais vues à l'entraînement, à 1 million de pas pour PPO et 200 000 pour SAC et CrossQ (écart 1). 10 graines (0 à 9) : résultat principal préregistré ; 20 graines (0 à 19) et budget égal (PPO lu à 200 000 pas sur sa courbe d'évaluation) : consolidation de l'écart 12, calculée le 21 novembre 2026 (écart 20). Les colonnes SAC et CrossQ sont identiques à budget égal et à budget propre. **En gras**, le chef de chaque tâche (plus forte IQM, sans test). Dernière ligne : changements de chef tranchés, quand le chef d'une tâche est battu par un autre programme sur une autre tâche avec un intervalle de probabilité d'amélioration qui exclut 0,5 ; entre parenthèses, avec les 24 intervalles rendus simultanés par Bonferroni (écart 18, tableau 6.4). Sources : `sim/analysis/e2.json`, `e2_s20.json`, `e2_equal.json`, `e2_equal_s20.json`, `e2_bonferroni.json`, `e2_bonferroni_s20.json`.

Tâche, programme	10 graines	20 graines	Budget égal, 10	Budget égal, 20
HalfCheetah				
PPO PyTorch	1 407 [1 251 ; 1 776]	1 515 [1 404 ; 1 665]	764 [667 ; 829]	733 [647 ; 801]
PPO JAX	1 254 [1 171 ; 1 830]	1 341 [1 228 ; 1 800]	576 [433 ; 622]	590 [449 ; 660]
SAC	**5 191 [4 431 ; 5 601]**	4 214 [2 822 ; 5 053]	**5 191 [4 431 ; 5 601]**	4 214 [2 822 ; 5 053]
CrossQ	4 801 [2 427 ; 7 815]	**4 694 [2 837 ; 6 868]**	4 801 [2 427 ; 7 815]	**4 694 [2 837 ; 6 868]**
Hopper				
PPO PyTorch	2 877 [2 245 ; 3 365]	2 668 [2 112 ; 3 209]	1 716 [1 104 ; 2 425]	1 430 [1 090 ; 1 982]
PPO JAX	**3 064 [2 504 ; 3 526]**	**2 864 [2 436 ; 3 232]**	1 715 [1 112 ; 2 360]	1 783 [1 229 ; 2 319]
SAC	1 308 [903 ; 2 113]	1 338 [988 ; 1 983]	1 308 [903 ; 2 113]	1 338 [988 ; 1 983]
CrossQ	2 302 [1 404 ; 3 013]	2 792 [2 175 ; 3 138]	**2 302 [1 404 ; 3 013]**	**2 792 [2 175 ; 3 138]**
Walker2d				
PPO PyTorch	2 158 [1 534 ; 3 081]	2 204 [1 714 ; 2 847]	489 [428 ; 543]	477 [432 ; 523]
PPO JAX	1 272 [969 ; 2 229]	1 435 [991 ; 2 062]	411 [371 ; 476]	420 [398 ; 453]
SAC	1 236 [959 ; 1 832]	1 165 [1 045 ; 1 529]	1 236 [959 ; 1 832]	1 165 [1 045 ; 1 529]
CrossQ	**5 414 [4 994 ; 5 627]**	**5 156 [4 764 ; 5 435]**	**5 414 [4 994 ; 5 627]**	**5 156 [4 764 ; 5 435]**
Swimmer				
PPO PyTorch	107 [88 ; 113]	107 [94 ; 111]	48 [45 ; 54]	49 [47 ; 50]
PPO JAX	**112 [109 ; 115]**	**115 [112 ; 119]**	93 [75 ; 97]	86 [73 ; 95]
SAC	47 [44 ; 49]	48 [47 ; 50]	47 [44 ; 49]	48 [47 ; 50]
CrossQ	110 [101 ; 114]	108 [87 ; 113]	**110 [101 ; 114]**	**108 [87 ; 113]**
Changements de chef tranchés	14 (12)	10 (6)	3	0

À budget égal. PPO reçoit cinq fois plus de pas que SAC et CrossQ. Lu à 200 000 pas sur sa courbe (écart 12, point iii), il ne mène plus sur aucune tâche.

- À dix graines aléatoires, SAC mène sur HalfCheetah, CrossQ sur les trois autres tâches. Il reste 3 changements établis : SAC, en tête sur HalfCheetah, est battu de façon tranchée par CrossQ sur Walker2d et sur Swimmer, puis par PPO JAX sur Swimmer. Sur Hopper, aucune paire n'est tranchée.
- À vingt graines aléatoires, CrossQ mène sur les quatre tâches et aucun changement n'est établi. Sur HalfCheetah, son avance sur SAC (4 694 contre 4 214) n'est pas établie non plus.

Ce que E2 établit, ses limites. Le verdict préenregistré tient : à dix graines aléatoires et aux budgets du protocole, le programme en tête change de tâche en tâche, de façon établie, même après correction de Bonferroni. Deux nuances pèsent. Le programme en tête bat rarement tous les autres de façon nette. Avec vingt graines aléatoires, le

Table 6.4. Expérience E2, comparaisons deux à deux : probabilité d'amélioration $P(A > B)$, la probabilité qu'un essai de A batte un essai de B sur la tâche, avec son intervalle par bootstrap stratifié (rliable, 2 000 rééchantillons). IC 95 % : intervalle de chaque comparaison seule, celui de la règle préregistrée (écart 7) ; IC Bonferroni : les 24 intervalles (6 paires sur 4 tâches) rendus simultanés au niveau de 95 %, chacun au niveau $1 - 0{,}05/24$, soit 99,79 % (écart 18 de `sim/SIM_PROTOCOL.md`, écrit après les résultats de E2 et avant ce calcul). Les paires sont orientées dans l'ordre du classement à 10 graines ; † : à 20 graines, le classement s'inverse et le fichier donne $P(B > A)$, reporté tel quel pour la paire inversée. **En gras**, les intervalles qui excluent 0,5 : comparaison tranchée. Sources : `sim/analysis/e2.json`, `e2_s20.json`, `e2_bonferroni.json`, `e2_bonferroni_s20.json`.

	10 graines			20 graines		
Tâche, paire $A > B$	$P(A > B)$	IC 95 %	IC Bonferroni	P	IC 95 %	IC Bonferroni
HalfCheetah						
SAC > CrossQ	0,45	[0,16 ; 0,72]	[0,06 ; 1,00]	0,68†	[0,48 ; 0,84]	[0,32 ; 0,90]
SAC > PPO PyTorch	0,99	**[0,94 ; 1,00]**	**[0,88 ; 1,00]**	0,81	**[0,65 ; 0,93]**	**[0,54 ; 0,98]**
SAC > PPO JAX	0,98	**[0,91 ; 1,00]**	**[0,82 ; 1,00]**	0,87	**[0,74 ; 0,96]**	**[0,64 ; 0,99]**
CrossQ > PPO PyTorch	0,95	**[0,82 ; 1,00]**	**[0,68 ; 1,00]**	0,95	**[0,86 ; 1,00]**	**[0,79 ; 1,00]**
CrossQ > PPO JAX	0,95	**[0,82 ; 1,00]**	**[0,72 ; 1,00]**	0,90	**[0,78 ; 0,98]**	**[0,70 ; 1,00]**
PPO PyTorch > PPO JAX	0,64	[0,36 ; 0,89]	[0,19 ; 1,00]	0,63	[0,44 ; 0,80]	[0,33 ; 0,88]
Hopper						
PPO JAX > PPO PyTorch	0,54	[0,26 ; 0,79]	[0,13 ; 0,90]	0,52	[0,33 ; 0,71]	[0,23 ; 0,78]
PPO JAX > CrossQ	0,77	**[0,52 ; 0,94]**	[0,37 ; 1,00]	0,56	[0,37 ; 0,73]	[0,26 ; 0,82]
PPO JAX > SAC	0,89	**[0,71 ; 1,00]**	**[0,57 ; 1,00]**	0,84	**[0,71 ; 0,95]**	**[0,62 ; 0,98]**
PPO PyTorch > CrossQ	0,66	[0,40 ; 0,88]	[0,24 ; 0,96]	0,45†	[0,27 ; 0,63]	[0,20 ; 0,74]
PPO PyTorch > SAC	0,89	**[0,72 ; 1,00]**	**[0,59 ; 1,00]**	0,82	**[0,69 ; 0,94]**	**[0,56 ; 0,97]**
CrossQ > SAC	0,74	[0,49 ; 0,94]	[0,36 ; 1,00]	0,81	**[0,65 ; 0,93]**	**[0,57 ; 0,97]**
Walker2d						
CrossQ > PPO PyTorch	0,99	**[0,94 ; 1,00]**	**[0,88 ; 1,00]**	0,94	**[0,83 ; 1,00]**	**[0,73 ; 1,00]**
CrossQ > PPO JAX	1,00	**[1,00 ; 1,00]**	**[1,00 ; 1,00]**	0,96	**[0,87 ; 1,00]**	**[0,81 ; 1,00]**
CrossQ > SAC	1,00	**[1,00 ; 1,00]**	**[1,00 ; 1,00]**	0,96	**[0,86 ; 1,00]**	**[0,79 ; 1,00]**
PPO PyTorch > PPO JAX	0,73	[0,48 ; 0,94]	[0,32 ; 1,00]	0,70	**[0,52 ; 0,86]**	[0,41 ; 0,93]
PPO PyTorch > SAC	0,79	**[0,56 ; 0,97]**	[0,42 ; 1,00]	0,83	**[0,67 ; 0,95]**	**[0,59 ; 0,99]**
PPO JAX > SAC	0,56	[0,30 ; 0,80]	[0,12 ; 0,91]	0,55	[0,37 ; 0,73]	[0,29 ; 0,85]
Swimmer						
PPO JAX > CrossQ	0,62	[0,36 ; 0,85]	[0,22 ; 0,95]	0,71	**[0,55 ; 0,87]**	[0,43 ; 0,93]
PPO JAX > PPO PyTorch	0,66	[0,38 ; 0,89]	[0,20 ; 1,00]	0,79	**[0,64 ; 0,91]**	**[0,54 ; 0,98]**
PPO JAX > SAC	1,00	**[1,00 ; 1,00]**	**[1,00 ; 1,00]**	1,00	**[1,00 ; 1,00]**	**[1,00 ; 1,00]**
CrossQ > PPO PyTorch	0,54	[0,28 ; 0,80]	[0,16 ; 0,89]	0,53	[0,34 ; 0,71]	[0,26 ; 0,80]
CrossQ > SAC	1,00	**[1,00 ; 1,00]**	**[1,00 ; 1,00]**	0,86	**[0,71 ; 0,98]**	**[0,64 ; 1,00]**
PPO PyTorch > SAC	0,91	**[0,70 ; 1,00]**	**[0,60 ; 1,00]**	0,87	**[0,71 ; 1,00]**	**[0,61 ; 1,00]**

changement de tête d'une tâche à l'autre vient en bonne part du budget cinq fois plus grand de PPO : à budget égal, CrossQ mène partout. Un article qui compare deux méthodes à budgets différents compare aussi ces budgets. Enfin, le classement vaut pour les valeurs par défaut des bibliothèques. Un réglage par tâche le changerait peut-être, dans un sens que nous ignorons : l'expérience E6 montre, sur un cas, qu'un réglage publié fait moins bien que les valeurs par défaut.

6.4 E3 : deux programmes du même algorithme

Question et plan. Deux programmes qui réalisent le même algorithme donnent-ils le même résultat ? E3 oppose PPO PyTorch à PPO JAX sur les quatre tâches, avec les mêmes graines aléatoires, les mêmes valeurs par défaut et un million de pas (section 5.10.4). Les deux programmes diffèrent par la bibliothèque de calcul et par l'initialisation des poids (section 5.4).

Dix graines aléatoires : aucun vainqueur, des écarts larges. Le tableau 6.5 donne les deux IQM, leur rapport et la probabilité qu'un essai de PPO JAX batte un essai de PPO PyTorch. Sur aucune tâche, l'une des deux versions ne l'emporte selon la règle du vainqueur établi. Les écarts apparents sont pourtant larges. Sur Walker2d, l'IQM vaut 2 158 avec PyTorch et 1 272 avec JAX, 41 % de moins. La probabilité que JAX fasse mieux vaut 0,27, de 0,06 à 0,52 : l'intervalle touche encore 0,5. Les deux intervalles des IQM se chevauchent. Sur les trois autres tâches, les probabilités d'amélioration restent proches de 0,5 : 0,36 sur HalfCheetah, 0,54 sur Hopper, 0,66 sur Swimmer. Les intervalles des IQM de `e3.json` diffèrent de quelques points de ceux de `e2.json` pour les mêmes essais, parce que chaque script consomme le générateur aléatoire dans son propre ordre (section 5.9.10).

Table 6.5. Expérience E3 : deux implémentations du même algorithme, PPO de Stable-Baselines3 (PyTorch) et PPO de SBX (JAX), mêmes graines, score final à 1 million de pas. IQM du score final avec intervalle à 95 % par bootstrap stratifié (rliable, 2 000 rééchantillons) ; écart relatif entre les deux IQM ; probabilité qu'un essai JAX batte un essai PyTorch avec son intervalle bootstrap. Règle préregistrée : un vainqueur n'est déclaré que si les deux intervalles d'IQM sont disjoints, ou si l'intervalle de la probabilité d'amélioration exclut 0,5. **En gras**, les tâches où un vainqueur est déclaré : aucune à 10 graines, deux à 20 graines, où les écarts ressemblent à ceux de 10 graines avec des intervalles plus étroits. Sources : `sim/analysis/e3.json`, `sim/analysis/e3_s20.json`.

Tâche	PPO PyTorch IQM [IC 95 %]	PPO JAX IQM [IC 95 %]	JAX/PyTorch − 1	$P(\text{JAX} > \text{PyTorch})$ [IC 95 %]	Vainqueur
10 graines (0 à 9), résultat principal					
HalfCheetah	1 407 [1 251 ; 1 776]	1 254 [1 171 ; 1 830]	−11 %	0,36 [0,11 ; 0,63]	aucun
Hopper	2 877 [2 232 ; 3 339]	3 064 [2 489 ; 3 536]	+6 %	0,54 [0,28 ; 0,80]	aucun
Walker2d	2 158 [1 543 ; 3 005]	1 272 [973 ; 2 180]	−41 %	0,27 [0,06 ; 0,52]	aucun
Swimmer	107 [88 ; 113]	112 [109 ; 115]	+4 %	0,66 [0,39 ; 0,89]	aucun
20 graines (0 à 19), écart 12					
HalfCheetah	1 515 [1 404 ; 1 665]	1 341 [1 228 ; 1 800]	−11 %	0,38 [0,20 ; 0,56]	aucun
Hopper	2 668 [2 107 ; 3 227]	2 864 [2 411 ; 3 257]	+7 %	0,52 [0,33 ; 0,70]	aucun
Walker2d	**2 204 [1 703 ; 2 860]**	**1 435 [987 ; 2 072]**	**−35 %**	**0,30 [0,15 ; 0,48]**	**PyTorch**
Swimmer	**107 [95 ; 111]**	**115 [112 ; 119]**	**+7 %**	**0,79 [0,63 ; 0,91]**	**JAX**

Vingt graines aléatoires : les deux versions se séparent. C'est le résultat qui bouge le plus avec les graines aléatoires ajoutées. Avec vingt graines aléatoires, deux tâches ont un vainqueur établi :

- Sur Walker2d, PyTorch l'emporte, 2 204 contre 1 435. La probabilité que JAX fasse mieux vaut 0,30, de 0,15 à 0,48.
- Sur Swimmer, JAX l'emporte, 115 contre 107. La probabilité vaut 0,79, de 0,63 à 0,91, avec des intervalles d'IQM disjoints.

Les écarts ressemblent à ceux de dix graines aléatoires. Les dix ajoutées resserrent les intervalles.

Ce que E3 établit, ses limites. Avec dix graines aléatoires, nous ne pouvons pas dire que les deux versions de PPO diffèrent. Avec vingt, elles diffèrent sur deux tâches, dans des sens opposés. Une méthode comparée à « PPO » sur Walker2d gagnerait ou perdrait selon la version choisie. La faille F14, l'implémentation qui change le résultat, tient en 2026 entre deux bibliothèques du même auteur, aux valeurs par défaut identiques. E3 ne dit pas laquelle des différences de code en est la cause : l'initialisation des poids, les générateurs aléatoires ou les arrondis de la bibliothèque de calcul.

6.5 E4 : les trois meilleurs sur dix

Question et plan. Que devient un résultat quand on ne publie que les trois meilleurs de dix essais, une pratique que Henderson et al. (2018) relevaient ? E4 applique cette sélection à chacun des seize couples de E2, puis compte les comparaisons qui changent de sens parmi les 24 (section 5.10.5).

Une moyenne toujours gonflée. Le tableau 6.6 et la figure 6.8 donnent le gonflement pour chaque couple. Les seize gonflements sont positifs, par construction. La moyenne gonfle de 32 % en médiane, de 6 % à 84 % selon le programme et la tâche. Les plus forts touchent PPO JAX sur Walker2d (84 %), SAC sur Hopper (78 %) et CrossQ sur HalfCheetah (74 %). L'écart type des trois meilleurs vaut en médiane 35 % de celui des dix : la sélection gonfle la moyenne et resserre la dispersion apparente, ce qui fait paraître le résultat plus sûr.

Trois comparaisons inversées. Sur 24 comparaisons, 3 changent de sens quand chaque programme ne garde que ses trois meilleurs essais : PPO PyTorch contre PPO JAX sur HalfCheetah et sur Hopper, CrossQ contre PPO PyTorch sur Swimmer. Deux des trois opposent les deux versions de PPO. Quand l'écart est faible, garder ses meilleurs essais suffit à changer de vainqueur.

Réplication sur les graines aléatoires 10 à 19. Refaite à l'identique sur dix autres graines aléatoires, la sélection gonfle la moyenne de 43 % en médiane, de 7 % à 85 %. Elle inverse encore 3 comparaisons sur 24 : CrossQ contre PPO PyTorch sur Hopper, PPO PyTorch contre PPO JAX sur Hopper, CrossQ contre PPO PyTorch sur Swimmer. Deux des trois inversions sont les mêmes que sur les graines aléatoires 0 à 9.

Ce que E4 établit. La sélection des meilleurs essais gonfle toujours la moyenne, d'un tiers environ en médiane. Elle inverse les comparaisons serrées. Le résultat se réplique

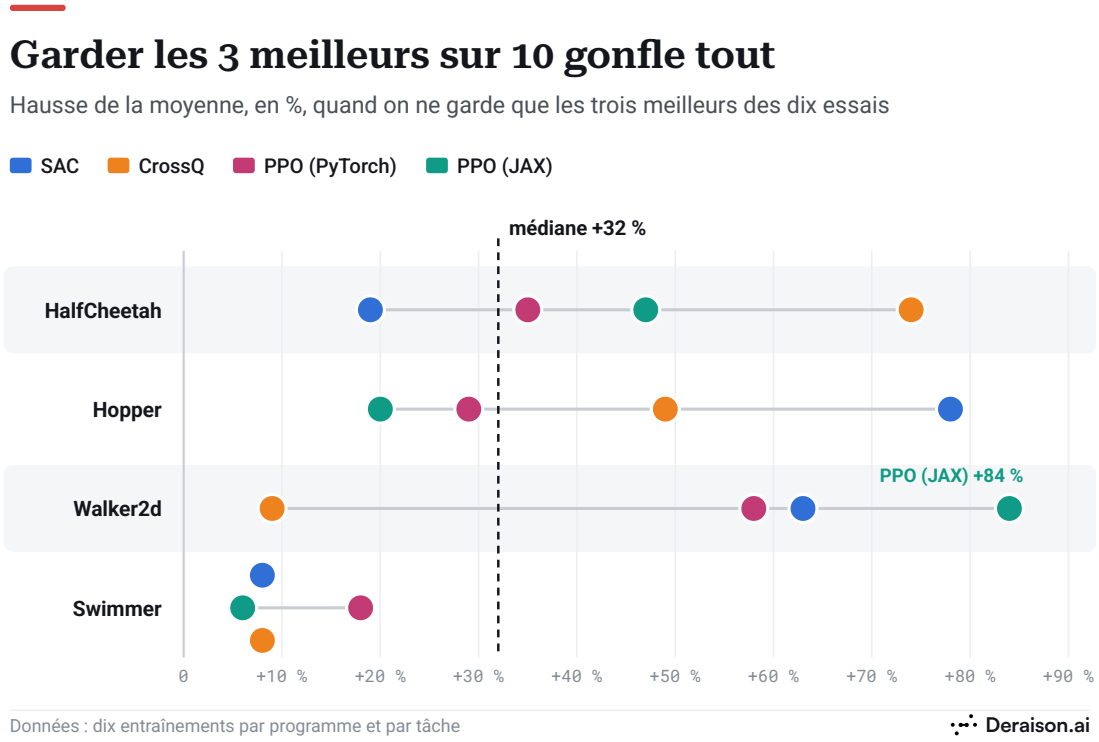


FIGURE 6.8. Hausse de la moyenne quand on ne publie que les trois meilleurs des dix essais, pour chaque programme (une couleur) et chaque tâche (une ligne), en pourcentage de la moyenne des dix. La ligne pointillée marque la médiane, +32 %. Les seize hausses sont positives : le tri sert toujours celui qui le pratique. Données : dix graines aléatoires par couple, scores d'évaluation finaux.

sur des graines aléatoires indépendantes. E4 mesure l'effet d'une pratique. Il ne dit pas combien d'articles la suivent : le sondage mesure cette part (critère X3, chapitre 4).

6.6 E5 : combien de graines aléatoires ?

Question et plan. Combien de graines aléatoires faut-il pour retrouver le bon ordre de deux programmes ? E5 tire avec remise n essais par programme parmi ceux observés, 10 000 fois, puis compte la part des tirages qui retrouvent l'ordre des moyennes de tous les essais (équation (5.8), section 5.10.6).

Dix graines aléatoires. Sur les 24 comparaisons, la part médiane des tirages qui retrouvent le bon ordre vaut 94 % avec trois graines aléatoires, 97 % avec cinq et 99 % avec dix (tableau 6.7). Ce chiffre rassurant tient aux écarts larges entre nos programmes. Sept comparaisons sont proches, avec des moyennes qui diffèrent de moins de 20 %. Pour elles, la médiane tombe à 65 % avec trois graines aléatoires, 71 % avec cinq, 77 % avec dix et 85 % avec vingt. La comparaison la plus difficile, CrossQ contre SAC sur HalfCheetah, ne retrouve le bon ordre que dans 60 % des tirages avec cinq graines aléatoires.

Vingt graines aléatoires. Le bon ordre devient celui des moyennes de vingt essais. Les tirages se font parmi les vingt. Sur les 24 comparaisons, la médiane ne bouge presque pas : 93 % avec trois graines aléatoires, 97 % avec cinq. Les comparaisons proches sont

Table 6.6. Expérience E4 : garder les 3 meilleurs essais sur 10. Pour chaque programme et chaque tâche, moyenne et écart type des scores finaux des 10 essais, puis des 3 meilleurs, et gonflement de la moyenne (moyenne des 3 meilleurs divisée par celle des 10, moins 1). Une comparaison (paire de programmes sur une tâche, 24 en tout) s'inverse quand le signe de l'écart entre moyennes change une fois que chaque programme ne garde que ses 3 meilleurs essais (écart 7 de `sim/SIM_PROTOCOL.md`). À droite, la même analyse refaite sur les graines 10 à 19 (écart 12). Comparaisons inversées, graines 0 à 9 : HalfCheetah : PPO PyTorch / PPO JAX ; Hopper : PPO PyTorch / PPO JAX ; Swimmer : CrossQ / PPO PyTorch ; graines 10 à 19 : Hopper : CrossQ / PPO PyTorch ; Hopper : PPO PyTorch / PPO JAX ; Swimmer : CrossQ / PPO PyTorch. L'écart type des 3 meilleurs tombe en médiane à 35 % de celui des 10. Sources : `sim/analysis/e4.json`, `sim/analysis/e4_r10.json`.

		Graines 0 à 9 (résultat principal)					Graines 10 à 19 (réplication)		
Tâche	Programme	moy. 10	é.-t. 10	moy. 3	é.-t. 3	gonfl.	moy. 10	moy. 3	gonfl.
HalfCheetah	PPO PyTorch	1 550	560	2 101	835	+35 %	1 752	2 329	+33 %
	PPO JAX	1 496	690	2 201	1 031	+47 %	1 930	3 577	+85 %
	SAC	4 982	1 214	5 931	422	+19 %	2 775	4 894	+76 %
	CrossQ	5 391	3 328	9 360	1 404	+74 %	5 027	8 125	+62 %
Hopper	PPO PyTorch	2 776	752	3 586	19	+29 %	2 471	3 572	+45 %
	PPO JAX	2 986	628	3 581	26	+20 %	2 483	3 272	+32 %
	SAC	1 488	820	2 653	150	+78 %	1 637	2 885	+76 %
	CrossQ	2 184	1 047	3 245	71	+49 %	2 871	3 346	+17 %
Walker2d	PPO PyTorch	2 294	1 107	3 633	663	+58 %	2 440	3 600	+48 %
	PPO JAX	1 624	1 104	2 988	1 148	+84 %	1 755	3 056	+74 %
	SAC	1 401	700	2 287	562	+63 %	1 379	2 205	+60 %
	CrossQ	5 313	495	5 784	148	+9 %	4 451	5 467	+23 %
Swimmer	PPO PyTorch	100	24	118	5	+18 %	96	113	+18 %
	PPO JAX	112	6	118	5	+6 %	117	125	+7 %
	SAC	47	4	50	2	+8 %	51	57	+12 %
	CrossQ	107	10	115	1	+8 %	88	124	+41 %
Gonflement médian (étendue)				+32 % (+6 % à +84 %)				+43 % (+7 % à +85 %)	
Comparaisons inversées						3 / 24			3 / 24

désormais huit, plus difficiles que dix graines aléatoires ne le laissaient voir : la médiane vaut 59 %, 62 %, 67 % et 74 % avec trois, cinq, dix et vingt graines aléatoires. CrossQ et PPO PyTorch sur Swimmer ont des moyennes qui diffèrent de moins de 1 % : même vingt graines aléatoires ne les ordonnent que dans 53 % des tirages, autant qu'une pièce de monnaie.

Ce que E5 établit, ses limites. Cinq graines aléatoires, la valeur médiane que le sondage relève dans les articles de 2026 (chapitre 4), suffisent pour des écarts larges. Elles ne suffisent pas pour des écarts de moins de 20 % : avec vingt graines aléatoires comme référence, un tel écart se lit à l'envers plus d'une fois sur trois. E5 fait tenir aux essais observés le rôle de la vraie distribution. Avec dix ou vingt essais, cette distribution est mal connue : pour les comparaisons les plus proches, le bon ordre l'est aussi.

6.7 E6 : une méthode de référence un peu trop belle

Question et plan. E6 voulait illustrer la faille F7 : l'écart qu'une nouvelle méthode s'attribue quand elle est seule à être réglée. Elle oppose, sur Hopper-v5, dix essais de PPO

TABLE 6.7. Expérience E5 : combien de graines retrouvent le bon classement ? Pour chaque comparaison de E2, 10 000 tirages avec remise de n essais par programme parmi ses essais (la distribution observée tient lieu de programme), puis la part des tirages dont les moyennes donnent le bon ordre, en pourcentage. Bon ordre : celui des moyennes de tous les essais choisis (10, ou 20 avec l'écart 20) ; à 10 graines, PPO PyTorch sur HalfCheetah tire dans les 30 essais de E1, comme préregistré. Écart : différence relative des deux moyennes, divisée par la plus grande en valeur absolue. **En gras**, les comparaisons serrées (écart sous 20 %) ; la dernière ligne donne leur nombre et la médiane de leurs parts. Sources : `sim/analysis/e5.json`, `sim/analysis/e5_s20.json`.

		10 graines (résultat principal)					20 graines (écarts 12 et 20)				
Tâche	Paire	écart	$n = 3$	5	10	20	écart	$n = 3$	5	10	20
HalfCheetah	CrossQ / PPO PyTorch	71	100	100	100	100	68	100	100	100	100
	CrossQ / PPO JAX	72	100	100	100	100	67	98	100	100	100
	CrossQ / SAC	**8**	**56**	**60**	**64**	**70**	26	73	79	89	96
	PPO PyTorch / PPO JAX	**3**	**73**	**71**	**77**	**84**	**4**	**47**	**51**	**56**	**58**
	PPO PyTorch / SAC	69	100	100	100	100	57	97	99	100	100
	PPO JAX / SAC	70	100	100	100	100	56	96	99	100	100
Hopper	CrossQ / PPO PyTorch	21	79	87	94	99	**4**	**55**	**57**	**60**	**63**
	CrossQ / PPO JAX	27	87	94	99	100	**8**	**61**	**65**	**71**	**80**
	CrossQ / SAC	32	83	89	96	99	38	91	96	99	100
	PPO PyTorch / PPO JAX	**7**	**65**	**69**	**76**	**85**	**4**	**58**	**59**	**63**	**68**
	PPO PyTorch / SAC	46	98	99	100	100	40	94	97	100	100
	PPO JAX / SAC	50	99	100	100	100	43	95	99	100	100
Walker2d	CrossQ / PPO PyTorch	57	100	100	100	100	52	99	100	100	100
	CrossQ / PPO JAX	69	100	100	100	100	65	100	100	100	100
	CrossQ / SAC	74	100	100	100	100	72	100	100	100	100
	PPO PyTorch / PPO JAX	29	79	84	91	98	29	79	85	93	98
	PPO PyTorch / SAC	39	89	95	99	100	41	93	97	100	100
	PPO JAX / SAC	**14**	**58**	**64**	**70**	**78**	**18**	**65**	**70**	**77**	**86**
Swimmer	CrossQ / PPO PyTorch	**7**	**65**	**72**	**83**	**91**	**1**	**50**	**51**	**53**	**53**
	CrossQ / PPO JAX	**4**	**74**	**82**	**91**	**97**	**15**	**84**	**91**	**98**	**100**
	CrossQ / SAC	57	100	100	100	100	50	99	100	100	100
	PPO PyTorch / PPO JAX	**11**	**81**	**88**	**96**	**100**	**14**	**89**	**94**	**99**	**100**
	PPO PyTorch / SAC	53	100	100	100	100	50	100	100	100	100
	PPO JAX / SAC	58	100	100	100	100	57	100	100	100	100
Médiane, 24 comparaisons			94	97	99	100		93	97	100	100
Médiane, comparaisons serrées		7	65	71	77	85	8	59	62	67	74

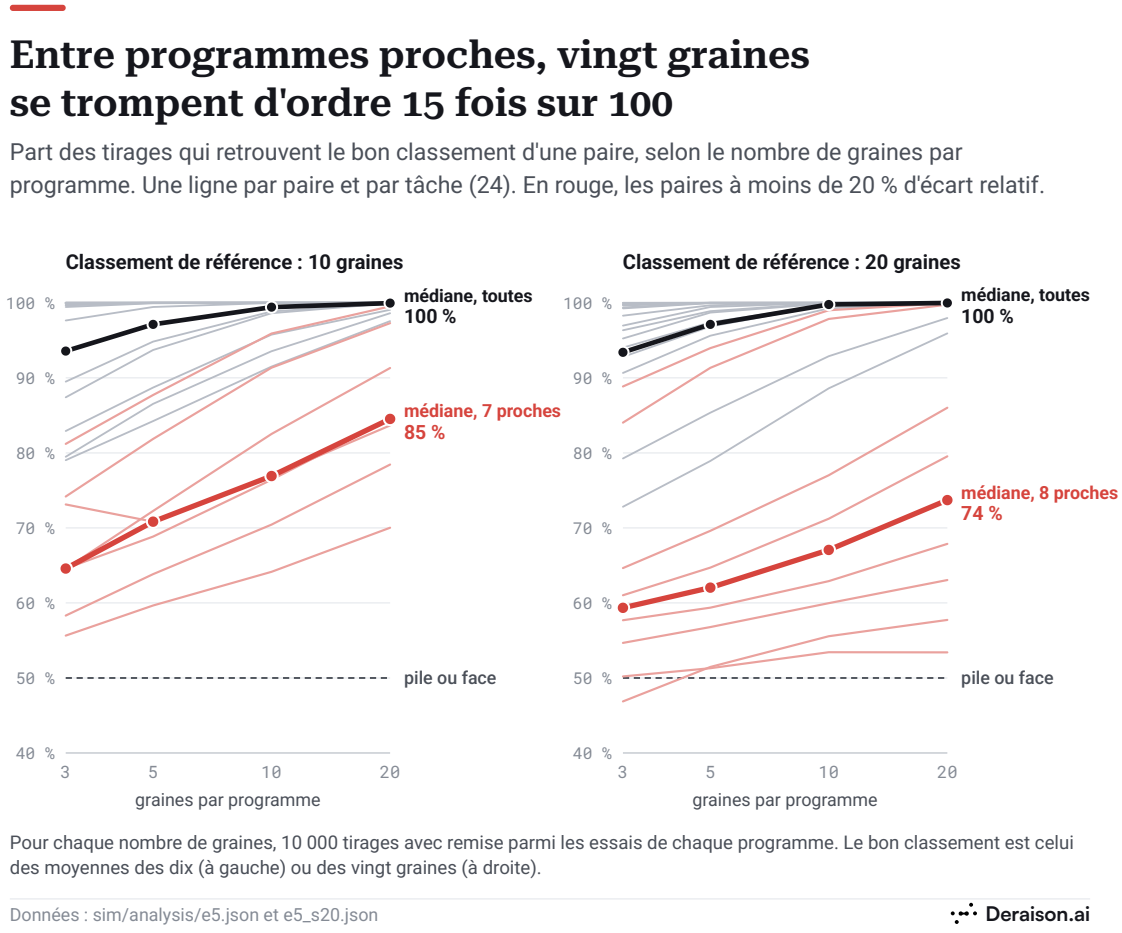


FIGURE 6.9. Part des 10 000 tirages qui retrouvent le bon ordre de deux programmes, en fonction du nombre de graines aléatoires tirées pour chacun (3, 5, 10, 20). Le bon ordre est celui des moyennes de tous les essais observés. Une courbe par comparaison entre deux programmes sur une tâche, 24 en tout. En rouge, les comparaisons proches, dont les moyennes diffèrent de moins de 20 %. Deux panneaux : le bon ordre et les tirages viennent de dix graines aléatoires, puis de vingt. Avec vingt graines aléatoires comme référence, cinq ne retrouvent le bon ordre des comparaisons proches que dans 62 % des tirages en médiane.

PyTorch avec les valeurs par défaut de Stable-Baselines3 à dix essais avec les réglages publiés par RL Baselines3 Zoo, qui font foi pour cette bibliothèque (section 5.10.7). Le tableau 6.8 rassemble tous les groupes.

Premier résultat : les réglages publiés feraient trois fois moins bien. La version par défaut atteint une IQM de 2 877 [2 268, 3 369]. La version réglée atteint 1 003 [856, 1 146]. En moyenne, 2 776 contre 994, avec un plafond près de 1 000 atteint dès 50 000 pas. Les réglages publiés feraient près de trois fois moins bien que les valeurs par défaut.

Une erreur de notre côté, qui n'explique rien. Le résultat était trop beau pour être cru sans vérification. Nous avons trouvé une erreur dans notre code. La normalisation des récompenses du Zoo emploie le facteur d'actualisation de l'algorithme, 0,999. Notre premier passage lui laissait la valeur par défaut, 0,99 (écart 2). Le diagnostic, décidé après ce premier résultat, relance la recette corrigée sur Hopper-v5 et sur Hopper-v4, plus les valeurs par défaut sur v4, dix graines aléatoires chacun.

- La recette corrigée obtient 1 012 $[902, 1\,135]$ sur v5, 984 $[773, 1\,002]$ sur v4, la version pour laquelle le recueil a été réglé.
- Les valeurs par défaut obtiennent 2 423 $[1\,657, 3\,058]$ sur v4.
- La probabilité qu'un essai corrigé batte un essai avec l'erreur vaut 0,47, de 0,21 à 0,73 : l'erreur ne change rien de détectable.
- Sur v4, un essai du Zoo corrigé ne bat un essai par défaut qu'avec une probabilité de 0,07, de 0 à 0,24. L'écart résiste à la correction de Holm ($p = 0{,}001$). Sur v5, la même probabilité vaut 0,01.

Notre erreur était réelle, sans expliquer l'écart. Elle n'explique pas non plus la différence de version : sur v4 comme sur v5, la recette plafonne près de 1 000.

Le script officiel plafonne aussi. Deux explications restaient possibles : la recette publiée fait vraiment moins bien que les valeurs par défaut avec les bibliothèques actuelles, ou notre façon de l'appliquer diffère encore de celle du recueil. L'écart 5, écrit avant tout entraînement officiel, tranche par le script du recueil lui-même. Les dix politiques entraînées par `rl_zoo3 . train` sur Hopper-v4, évaluées avec notre protocole, obtiennent 995 $[821, 1\,006]$. C'est autant que notre recette corrigée, 984 $[777, 1\,002]$ dans cette analyse : la probabilité qu'une politique officielle batte la nôtre vaut 0,58, de 0,32 à 0,82. Les valeurs par défaut obtiennent 2 423 $[1\,687, 3\,058]$. Une politique officielle ne bat une politique par défaut qu'avec une probabilité de 0,09, de 0 à 0,30. L'écart résiste à la correction de Holm ($p = 0{,}001$). Les intervalles de 984 et de 2 423 diffèrent de quelques points entre les deux analyses, parce que chacune tire ses propres rééchantillons.

Ce que E6 établit. Selon la règle de lecture écrite d'avance, notre application de la recette n'était pas en cause. Avec Stable-Baselines3 2.9.2, Gymnasium 1.1.1 et MuJoCo 4.0.0, la recette réglée et publiée par le recueil fait moins bien que les valeurs par défaut sur Hopper, sur sa version 4 comme sur la version 5. E6 ne montre donc pas une référence mal réglée par ses auteurs. Il montre qu'un réglage publié fait moins bien, aujourd'hui, que les valeurs par défaut de la même bibliothèque. L'explication la plus probable est que ce réglage a vieilli avec les logiciels qui l'entourent. E6 rappelle aussi qu'un résultat trop beau se vérifie jusqu'au script officiel.

Limites. Nous ne savons pas quel changement de bibliothèque a fait vieillir la recette. Nous n'avons essayé qu'une tâche et qu'un algorithme du recueil. Le plafond près de 1 000 correspond à peu près à ce que rapporte, sur 1 000 pas, la prime donnée à un sauteur qui reste debout : il évoque un agent qui tient debout sans avancer. Nous ne l'avons pas vérifié sur les vidéos de chaque politique.

6.8 E7 : ce que coûte une physique modifiée

Question et plan. Pineau demandait de séparer l'entraînement de l'évaluation. Nos agents sont déjà évalués sur des graines aléatoires jamais vues à l'entraînement. E7 va plus loin : chaque politique finale joue aussi dans une physique modifiée, avec des masses alourdies de 20 % et des frottements de glissement réduits de 20 %. Pour chaque essai, on calcule la part du score gardée, le rapport du score en physique modifiée au score en physique d'entraînement, puis son IQM sur les dix graines aléatoires avec son intervalle (section 5.10.8).

Table 6.8. Expérience E6, la méthode de référence : PPO PyTorch sur Hopper, un million de pas, 10 graines par groupe, valeurs par défaut de Stable-Baselines3 contre les réglages publiés par RL Baselines3 Zoo. La première passe (préregistrée, `sim/analysis/e6.json`, bootstrap de 2 000 rééchantillons) normalisait les observations avec un gamma de 0,99 au lieu de 0,999 ; le diagnostic décidé après ce résultat (écart 2) relance la recette corrigée sur Hopper v5 et v4, la version pour laquelle le Zoo a été réglé (`e6_diag.json`, 5 000 rééchantillons, Holm sur trois comparaisons) ; puis dix agents entraînés avec le script officiel du Zoo, évalués avec notre protocole (écart 5, écrit avant tout entraînement officiel ; `e6_official.json`, 2 000 rééchantillons, Holm sur deux comparaisons). Les bornes d'un même groupe varient ainsi d'un fichier à l'autre au niveau Monte-Carlo. IQM avec intervalle à 95 % par bootstrap (rliable) ; $P(A > B)$: probabilité qu'un essai de A batte un essai de B, intervalle bootstrap ; test de Welch, correction de Holm, test de Mann-Whitney quand le script le calcule. **En gras**, les comparaisons qui passent la correction de Holm : la recette publiée fait moins bien que les valeurs par défaut, avec notre code comme avec le script officiel. Moyenne et écart type des agents officiels calculés ici à partir de leurs dix scores (`e6_official.json`).

Groupe (PPO PyTorch, 10 graines)	moyenne	écart type	IQM [IC 95 %]	source
Valeurs par défaut, Hopper-v5	2 776	752	2 877 [2 268 ; 3 369]	e6_diag
Zoo, première passe (gamma 0,99), v5	994	285	1 003 [856 ; 1 146]	e6_diag
Zoo corrigé (gamma 0,999), v5	1 021	300	1 012 [902 ; 1 135]	e6_diag
Valeurs par défaut, Hopper-v4	2 360	911	2 423 [1 657 ; 3 058]	e6_diag
Zoo corrigé, v4	877	309	984 [773 ; 1 002]	e6_diag
Script officiel du Zoo, v4	898	310	995 [821 ; 1 006]	e6_official
Zoo corrigé, v4			984 [777 ; 1 002]	e6_official
Valeurs par défaut, Hopper-v4			2 423 [1 687 ; 3 058]	e6_official
Valeurs par défaut, Hopper-v5, préregistré			2 877 [2 278 ; 3 368]	e6
Zoo, première passe (gamma 0,99), v5, préregistré			1 003 [855 ; 1 149]	e6

Comparaison A contre B	$P(A > B)$ [IC]	p Welch	p Holm	p Mann-Whitney
Zoo corrigé (gamma 0,999), v5 / Zoo, première passe (gamma 0,99), v5	0,47 [0,21 ; 0,73]	0,84	0,84	0,85
Zoo corrigé, v4 / Valeurs par défaut v4	**0,07 [0,00 ; 0,24]**	**0,0005**	**0,001**	**0,001**
Zoo corrigé (gamma 0,999), v5 / Valeurs par défaut v5	**0,01 [0,00 ; 0,06]**	**2×10^{-5}**	**6×10^{-5}**	**0,0002**
Script officiel du Zoo, v4 / Zoo corrigé, v4	0,58 [0,32 ; 0,82]	0,88	0,88	
Script officiel du Zoo, v4 / Valeurs par défaut v4	**0,09 [0,00 ; 0,30]**	**0,0005**	**0,001**	

Douze couples sur seize perdent. Le tableau 6.9 et la figure 6.10 donnent la part gardée pour chaque couple. Douze couples sur seize perdent une part établie de leur score, c'est-à-dire avec un intervalle qui reste sous 1. Trois ne changent pas de façon détectable. Un gagne : SAC sur Swimmer, à 1,07 $[1{,}02, 1{,}12]$. La perte dépend du programme et de la tâche.

- Sur Hopper, PPO ne garde que 22 % de son score dans sa version PyTorch et 18 % dans sa version JAX, quand SAC en garde 64 % et CrossQ 65 %.
- Sur Walker2d, PPO PyTorch garde 34 %, CrossQ 46 %, PPO JAX 52 %, SAC 84 %.
- Sur HalfCheetah, les pertes sont plus modestes : de 85 % gardés pour PPO PyTorch à 92 % pour PPO JAX.
- Sur Swimmer, aucun programme ne perd.

Le classement bouleversé sur Hopper. Le complément descriptif de l'écart 11, sans test, classe les programmes dans chaque physique par IQM du score. Sur Hopper, l'ordre se renverse. PPO JAX, premier dans la physique d'entraînement avec 3 064, finit dernier dans la physique modifiée avec 525. CrossQ, troisième avec 2 302, prend la tête avec 1 302, devant SAC (751) et PPO PyTorch (609). Sur Walker2d, SAC passe de la dernière place à la deuxième. Sur HalfCheetah et Swimmer, l'ordre ne change pas. La figure 6.11 montre le renversement de Hopper.

Vingt graines aléatoires. Les verdicts restent les mêmes pour 14 couples sur 16. Deux changent. SAC sur Walker2d ne perd plus de façon détectable : 0,91, de 0,79 à 1,01. CrossQ sur Swimmer gagne un peu : 1,03, de 1,01 à 1,04. Il y a ainsi 11 pertes et 2 gains. Le classement descriptif se renverse toujours sur Hopper, seule tâche où le programme en tête change.

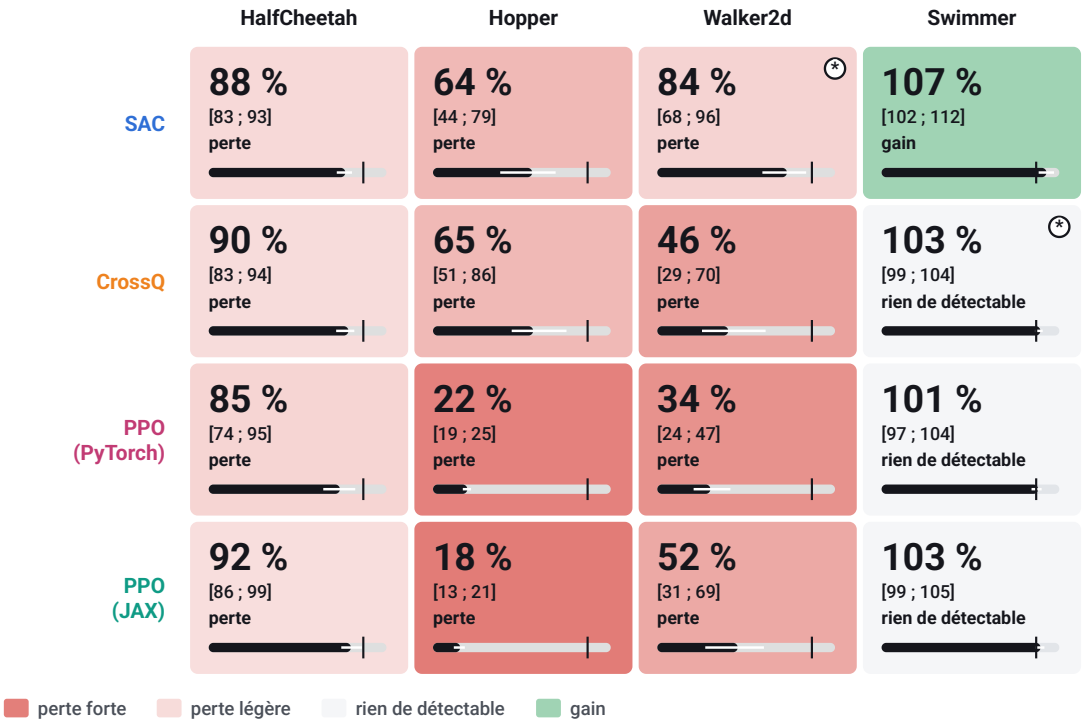
Ce que E7 établit, ses limites. Un programme en tête dans la physique d'entraînement finit dernier dans une physique voisine : PPO JAX sur Hopper. Le classement établi dans le monde de l'entraînement dit peu de chose de la robustesse. E7 ne teste qu'une modification, qui change deux grandeurs à la fois. La section suivante sépare les deux grandeurs et fait varier l'ampleur du changement. Les classements dans chaque

Table 6.9. Expérience E7, physique modifiée : chaque politique finale évaluée sur les mêmes graines d'évaluation tenues à part, dans la physique d'entraînement puis avec les masses multipliées par 1,2 et le frottement par 0,8. Ratio : score en physique modifiée divisé par le score en physique d'entraînement, par essai ; IQM des ratios sur les graines avec intervalle à 95 % par bootstrap stratifié (rliable, 2 000 rééchantillons). Verdict préregistré : perte quand l'intervalle reste sous 1, gain quand il reste au-dessus, ns (aucun changement détectable) sinon ; **en gras**, les pertes et les gains. Rang : place du programme sur la tâche par IQM du score, dans la physique d'entraînement puis dans la physique modifiée (complément descriptif de l'écart 11, décidé après les résultats, sans test). Le chef change sur Hopper à 10 graines, sur Hopper à 20 graines. Sources : `sim/analysis/e7.json`, `e7_s20.json`, `e7_rank.json`, `e7_rank_s20.json`.

		10 graines (résultat principal)			20 graines (écart 12)		
Tâche	Programme	ratio [IC 95 %]	verdict	rang	ratio [IC 95 %]	verdict	rang
HalfCheetah	PPO PyTorch	**0,85 [0,74 ; 0,95]**	**perte**	3→3	**0,89 [0,80 ; 0,97]**	**perte**	3→3
	PPO JAX	**0,92 [0,86 ; 0,99]**	**perte**	4→4	**0,89 [0,85 ; 0,94]**	**perte**	4→4
	SAC	**0,88 [0,83 ; 0,93]**	**perte**	1→1	**0,91 [0,88 ; 0,93]**	**perte**	2→2
	CrossQ	**0,90 [0,83 ; 0,94]**	**perte**	2→2	**0,91 [0,86 ; 0,93]**	**perte**	1→1
Hopper	PPO PyTorch	**0,22 [0,19 ; 0,25]**	**perte**	2→3	**0,23 [0,20 ; 0,27]**	**perte**	3→3
	PPO JAX	**0,18 [0,13 ; 0,21]**	**perte**	1→4	**0,19 [0,16 ; 0,23]**	**perte**	1→4
	SAC	**0,64 [0,44 ; 0,79]**	**perte**	4→2	**0,64 [0,49 ; 0,77]**	**perte**	4→2
	CrossQ	**0,65 [0,51 ; 0,86]**	**perte**	3→1	**0,55 [0,41 ; 0,73]**	**perte**	2→1
Walker2d	PPO PyTorch	**0,34 [0,24 ; 0,47]**	**perte**	2→3	**0,36 [0,25 ; 0,46]**	**perte**	2→3
	PPO JAX	**0,52 [0,31 ; 0,69]**	**perte**	3→4	**0,48 [0,34 ; 0,64]**	**perte**	3→4
	SAC	**0,84 [0,68 ; 0,96]**	**perte**	4→2	0,91 [0,79 ; 1,01]	ns	4→2
	CrossQ	**0,46 [0,29 ; 0,70]**	**perte**	1→1	**0,47 [0,32 ; 0,71]**	**perte**	1→1
Swimmer	PPO PyTorch	1,01 [0,97 ; 1,04]	ns	3→3	1,02 [0,99 ; 1,03]	ns	3→3
	PPO JAX	1,03 [0,99 ; 1,05]	ns	1→1	1,02 [0,98 ; 1,04]	ns	1→1
	SAC	**1,07 [1,02 ; 1,12]**	**gain**	4→4	**1,06 [1,04 ; 1,08]**	**gain**	4→4
	CrossQ	1,03 [0,99 ; 1,04]	ns	2→2	**1,03 [1,01 ; 1,04]**	**gain**	2→2
Pertes, gains (sur 16)		12, 1			11, 2		

Sous une physique modifiée, douze combinaisons sur seize perdent du score

Part du score gardée quand les masses passent à 120 % et le frottement à 80 %, moyenne interquartile sur dix graines avec son intervalle à 95 %. Perte déclarée seulement si l'intervalle reste sous 100 %.



Jauge : part gardée de 0 à 115 %, son intervalle en trait blanc, un repère à 100 %. Astérisque : le verdict change avec vingt graines. SAC sur Walker2d : rien de détectable (91 %) ; CrossQ sur Swimmer : gain (103 %).

Données : sim/analysis/e7.json et e7_s20.json, intervalles bootstrap stratifiés (rliable, 2 000 tirages) Deraison.ai

FIGURE 6.10. Part du score gardée dans une physique modifiée (masses multipliées par 1,2, frottements de glissement par 0,8), pour chaque programme et chaque tâche. Chaque case donne la moyenne interquartile, sur dix graines aléatoires, du rapport entre le score dans la physique modifiée et le score dans la physique d'entraînement, les deux mesurés sur les mêmes graines aléatoires d'évaluation. Les tâches sont en colonnes, les programmes en lignes. Une case marquée d'une perte a un intervalle à 95 % qui reste sous 1, une case marquée d'un gain un intervalle au-dessus de 1.

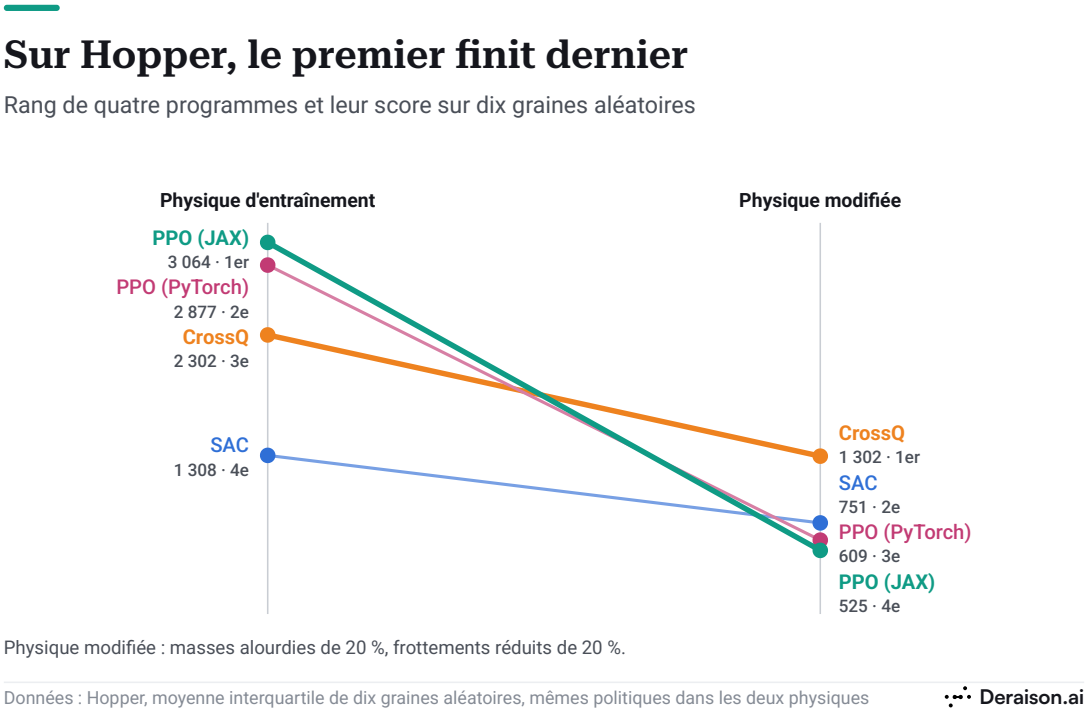


FIGURE 6.11. Hopper, quatre programmes, dix graines aléatoires chacun. À gauche, le score dans la physique de l'entraînement. À droite, le score avec des masses alourdies de 20 % et des frottements réduits de 20 %. Chaque ligne relie le score et le rang d'un programme dans les deux physiques. PPO écrit en JAX passe de la première à la dernière place, CrossQ de la troisième à la première. Les scores sont des moyennes interquartiles des scores d'évaluation finaux. Les rangs les comparent sans test.

physique comparent des IQM sans test. La part gardée se calcule essai par essai : elle ne se lit pas comme le rapport des deux IQM du classement.

6.9 La physique modifiée par degrés

Question et plan. Le renversement de E7 tient-il à une seule modification, choisie d'avance ? L'écart 12, point ii, réévalue chaque politique finale en ne changeant qu'une grandeur à la fois : les masses de 30 % plus légères à 30 % plus lourdes, les frottements de 40 % plus faibles à 20 % plus forts. Un couple perd quand l'intervalle de l'IQM de son rapport au score d'origine reste sous 1, comme dans E7 (section 5.10.8).

Le contrôle. À 1,0 fois, les réévaluations retrouvent le score final écrit par `run.py` : le plus grand écart relatif vaut zéro, pour les 160 politiques à dix graines aléatoires comme pour les 320 à vingt.

La perte croît avec l'ampleur du changement. Le tableau 6.10, la figure 6.12 et la figure 6.13 comptent les couples qui perdent, sur 16, pour chaque physique. À dix graines aléatoires :

- masses multipliées par 0,7, 0,8, 0,9 : 8, 6 et 3 couples perdent ;
- masses multipliées par 1,1, 1,2, 1,3 : 9, 12 et 11 couples perdent ;

- frottements multipliés par 0,6, 0,7, 0,8, 0,9 : 7, 6, 4 et 3 couples perdent ;
- frottements multipliés par 1,1 et 1,2 : 1 et 9 couples perdent.

Les pertes dessinent un V autour de la physique d'origine, plus raide du côté des masses alourdies. Alourdir les masses de 20 % fait perdre à lui seul 12 couples, autant que la modification combinée de E7. Réduire les frottements de 20 % n'en fait perdre que 4. Dans E7, la perte semble venir surtout des masses, même si les deux expériences ne testent pas cette décomposition. Quelques couples gagnent, surtout quand les frottements baissent : 3 gains à 0,6 fois et à 0,7 fois.

Le programme en tête de Hopper selon la physique. Sur Hopper, PPO JAX, premier dans la physique d'origine, n'est plus premier dans 10 des 12 physiques modifiées. Il ne garde la tête qu'avec des frottements à 0,9 et 1,1 fois. La tête passe à PPO PyTorch ou à CrossQ selon la grandeur et le facteur. Sur les trois autres tâches, le programme en tête change peu : SAC dans les douze physiques de HalfCheetah, CrossQ dans les douze de Walker2d, PPO JAX dans dix des douze de Swimmer.

Vingt graines aléatoires. Le V garde sa forme. Avec des masses à 0,9, 1,1, 1,2 et 1,3 fois, 4, 11, 12 et 12 couples perdent. Avec des frottements à 1,2 fois, 10. Sur Hopper, PPO JAX perd toujours la tête dans 10 physiques sur 12. Sur HalfCheetah, CrossQ prend la tête dans 9 des 12 physiques, comme il la prend dans la physique d'origine à vingt graines aléatoires.

Ce que la physique par degrés établit. Le renversement de Hopper ne tient pas à une modification choisie. Il apparaît dans presque toutes les physiques voisines, à dix

Table 6.10. Expérience E7 à plusieurs intensités (écart 12 (ii) de `sim/SIM_PROTOCOL.md`, règle écrite avant tout calcul ; écart 14 (a) sur la date du script) : chaque politique finale réévaluée sur les mêmes dix graines d'évaluation en changeant une seule grandeur à la fois. Pour chaque intensité, nombre de paires programme et tâche, sur 16, qui perdent (IQM du ratio au score en physique d'origine, intervalle à 95 % par bootstrap stratifié rliable de 2 000 rééchantillons, entièrement sous 1) ou qui gagnent (intervalle au-dessus de 1). Chef sur Hopper : programme de plus forte IQM du score, sans test ; dans la physique d'origine, PPO JAX y mène. Il perd la première place à 10 intensités sur 12 à 10 graines, 10 sur 12 à 20 graines. Contrôle : à 1,0, les réévaluations retrouvent le score final de `run.py` (plus grand écart relatif 0,0 à 10 graines, 0,0 à 20). Sources : `sim/analysis/dose.json`, `sim/analysis/dose_s20.json`.

		10 graines			20 graines		
Grandeur	Facteur	pertes	gains	chef sur Hopper	pertes	gains	chef sur Hopper
masses	0,7	8	0	PPO PyTorch	9	0	PPO PyTorch
	0,8	6	1	PPO PyTorch	7	1	PPO PyTorch
	0,9	3	1	PPO PyTorch	4	5	PPO JAX
	1,1	9	2	CrossQ	11	4	CrossQ
	1,2	12	1	PPO PyTorch	12	2	CrossQ
	1,3	11	1	CrossQ	12	2	CrossQ
frottement	0,6	7	3	CrossQ	7	3	CrossQ
	0,7	6	3	CrossQ	6	3	CrossQ
	0,8	4	2	CrossQ	5	3	CrossQ
	0,9	3	1	PPO JAX	4	4	CrossQ
	1,1	1	0	PPO JAX	3	0	PPO JAX
	1,2	9	0	PPO PyTorch	10	0	PPO PyTorch

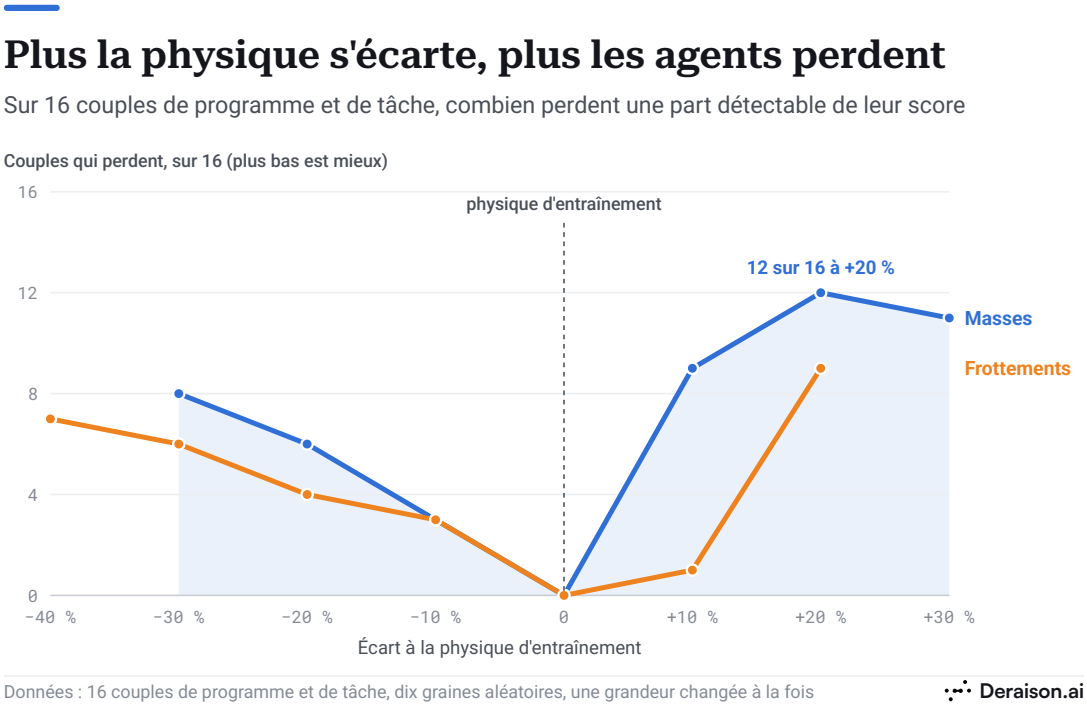


FIGURE 6.12. Quatre programmes sur quatre tâches, dix graines aléatoires chacun, soit 16 couples de programme et de tâche. Chaque point compte les couples qui perdent une part détectable de leur score quand une seule grandeur de la physique change : les masses (de 30 % plus légères à 30 % plus lourdes) ou les frottements de glissement (de 40 % plus faibles à 20 % plus forts). Un couple perd quand l'intervalle à 95 % de son score, rapporté à celui de la physique d'entraînement, reste sous 1. À 0 %, la physique d'entraînement sert de référence : aucune perte n'y est possible.

comme à vingt graines aléatoires. Les classements par physique restent sans test. Les modifications restent petites et uniformes : chaque masse ou chaque frottement change du même facteur. Elles ne remplacent pas un vrai test de généralisation, sur d'autres corps ou d'autres tâches.

6.10 B3 : la mesure change-t-elle le classement ?

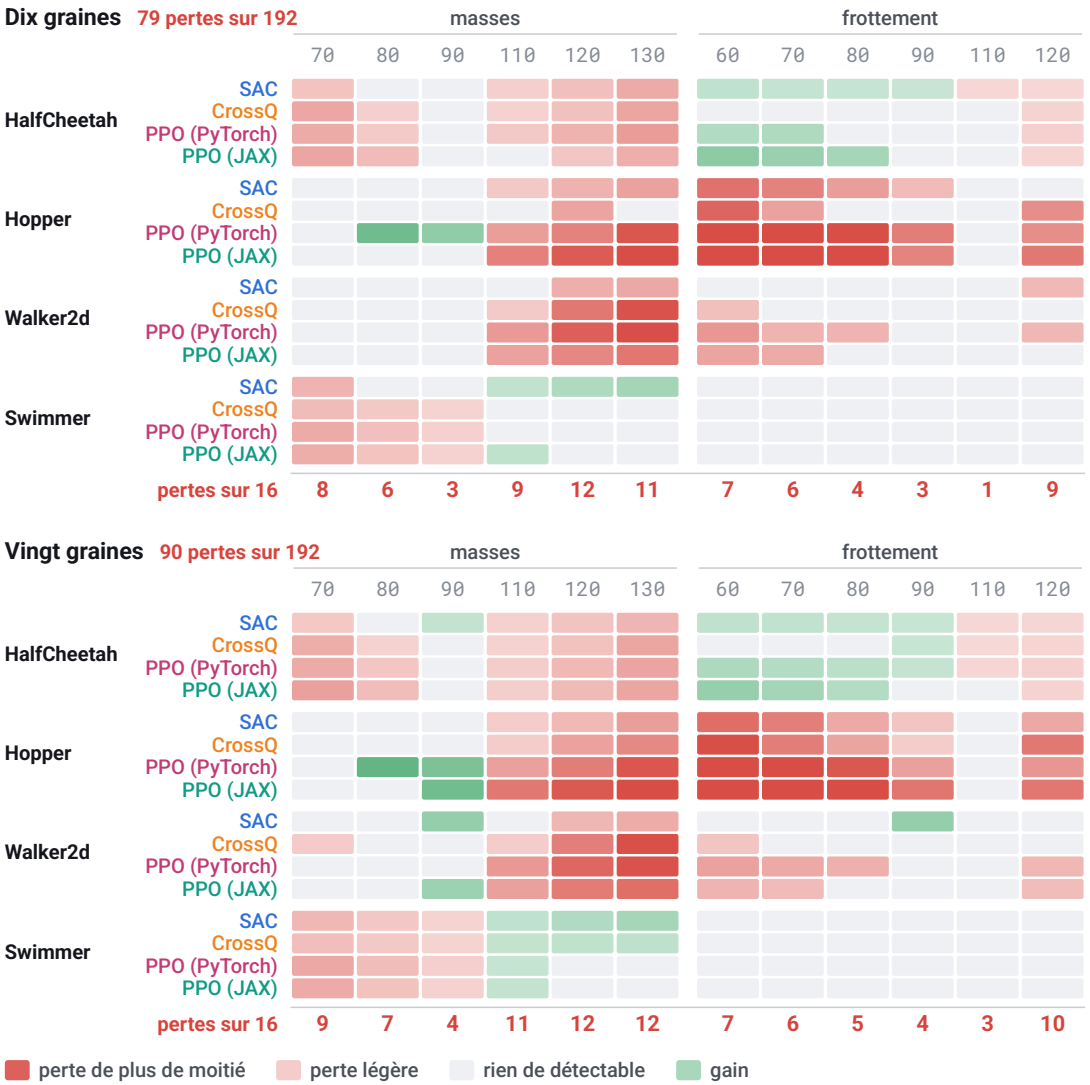
Question et plan. Un article qui ne définit pas sa mesure peut-il choisir son vainqueur ? B3 classe les quatre programmes selon trois mesures calculées sur les mêmes courbes : le score final, le meilleur score atteint le long de la courbe et la moyenne de la courbe (section 5.10.9).

Dix graines aléatoires : huit inversions, aucune établie. Huit paires de programmes changent d'ordre d'une mesure à l'autre (tableau 6.11). Le programme en tête change de mesure en mesure sur deux tâches.

- Sur HalfCheetah, la moyenne de la courbe place CrossQ devant SAC, en tête selon le score final et le meilleur score.
- Sur Hopper, le meilleur score place PPO PyTorch devant PPO JAX, en tête selon les deux autres mesures.

Loin de la physique d'entraînement, les pertes s'accumulent : 12 sur 16 à 120 % des masses

Part du score gardée, une quantité changée à la fois : masses de 70 à 130 %, frottement de 60 à 120 %. Une case par programme, tâche et intensité, colorée quand l'intervalle à 95 % évite 100 %.



En haut des colonnes, l'intensité en pour cent de la valeur d'entraînement. Vingt graines rétrécissent les intervalles : plus de pertes deviennent détectables.

Données : sim/analysis/dose.json et dose_s20.json, intervalles bootstrap stratifiés (rliable, 2 000 tirages)

Deraison.ai

FIGURE 6.13. Part du score gardée dans douze physiques modifiées, avec dix graines aléatoires puis vingt, côte à côte. Une rangée par programme et par tâche (16), une colonne par physique : masses de 70 % à 130 % de leur valeur d'entraînement, puis frottements de glissement de 60 % à 120 %, une seule grandeur changée à la fois. Une case n'est colorée que si l'intervalle à 95 % de la moyenne interquartile du rapport au score d'origine évite 1 : en rouge une perte, plus foncée quand plus de la moitié du score est perdue, en vert un gain. Sous chaque colonne, le nombre de couples qui perdent, sur 16.

Les autres inversions touchent CrossQ et PPO PyTorch sur Hopper et sur Swimmer. Aucune n'est établie : elles opposent des programmes proches, que dix graines aléatoires ne départagent sous aucune mesure. La moyenne de la courbe couvre un million de pas pour PPO et 200 000 pour SAC et CrossQ. Elle favorise un programme qui apprend vite au début.

Vingt graines aléatoires et budget égal.

- À budgets inégaux et vingt graines aléatoires, 8 paires changent encore d'ordre, sans inversion établie. Le programme en tête ne change de mesure en mesure que sur Hopper.
- À budget égal et dix graines aléatoires, 4 inversions, aucune établie. Le programme en tête ne change de mesure en mesure que sur HalfCheetah, entre SAC et CrossQ.
- À budget égal et vingt graines aléatoires, 6 inversions, dont 1 établie : sur Swimmer, PPO PyTorch et SAC changent d'ordre entre le meilleur score atteint et la moyenne de la courbe. Le programme en tête ne change de mesure en mesure sur aucune tâche.

Ce que B3 établit. Entre deux programmes proches, la mesure choisie décide parfois du vainqueur apparent. Entre programmes nettement séparés, elle ne change rien. Une seule inversion est établie dans les quatre analyses. Elle oppose deux programmes loin du premier. Un article qui ne dit pas sa mesure laisse ouverte, pour les comparaisons serrées, la question du vainqueur.

6.11 Dix ou vingt graines aléatoires, budget égal : ce qui tient et ce qui change

Les résultats à dix graines aléatoires restent les résultats principaux, comme le protocole l'annonçait. Les ajouts de l'écart 12 les confirment ou les nuancent.

Ce qui tient.

- Le programme en tête change avec la tâche aux budgets du protocole (E2), de façon établie, avec ou sans correction de Bonferroni.
- La sélection des trois meilleurs essais sur dix gonfle toujours la moyenne et inverse 3 comparaisons sur 24, sur les graines aléatoires 10 à 19 comme sur les graines aléatoires 0 à 9 (E4).
- La physique modifiée renverse le classement de Hopper. Les verdicts de E7 restent les mêmes pour 14 couples sur 16. La physique par degrés garde sa forme en V.
- Les inversions de B3 à budgets inégaux restent au nombre de 8, sans inversion établie.

Ce qui change.

- Sur HalfCheetah, CrossQ prend la tête à SAC, sans écart établi entre eux (E2).
- Les deux versions de PPO se séparent sur Walker2d et Swimmer, dans des sens opposés (E3).

TABLE 6.11. Expérience B3, la mesure change-t-elle le classement ? (écart 8 de `sim/SIM_PROTOCOL.md`, sans entraînement). Sur les courbes d'évaluation enregistrées (un point tous les 50 000 pas), trois mesures par essai : le score final, le meilleur score atteint le long de la courbe, la moyenne de la courbe. Par tâche, IQM de chaque mesure et classement des programmes ; chef : plus forte IQM. Une paire s'inverse quand son ordre par IQM change d'une mesure à l'autre ; l'inversion est tranchée quand, sous chacune des deux mesures, l'intervalle de la probabilité d'amélioration (bootstrap stratifié rliable, 2 000 rééchantillons) exclut 0,5 dans des sens opposés. **En gras**, les chefs qui changent avec la mesure et les inversions tranchées. La moyenne de la courbe couvre un million de pas pour PPO et 200 000 pour SAC et CrossQ, sauf à budget égal (écart 12 (iii)). Sources : `sim/analysis/b3.json`, `b3_s20.json`, `b3_equal.json`, `b3_equal_s20.json`.

Tâche	chef, final	chef, meilleur	chef, moyenne	paires inversées (mesures)
10 graines, budgets propres (résultat principal) : 8 inversions, 0 tranchée				
HalfCheetah	**SAC**	**SAC**	**CrossQ**	CrossQ, SAC (final/moyenne, meilleur/moyenne)
Hopper	**PPO JAX**	**PPO PyTorch**	**PPO JAX**	PPO PyTorch, PPO JAX (final/meilleur, meilleur/moyenne) ; CrossQ, PPO PyTorch (final/moyenne, meilleur/moyenne)
Walker2d	CrossQ	CrossQ	CrossQ	
Swimmer	PPO JAX	PPO JAX	PPO JAX	CrossQ, PPO PyTorch (final/moyenne, meilleur/moyenne)
20 graines, budgets propres : 8 inversions, 0 tranchée				
HalfCheetah	CrossQ	CrossQ	CrossQ	
Hopper	**PPO JAX**	**PPO PyTorch**	**CrossQ**	CrossQ, PPO PyTorch (final/meilleur, meilleur/moyenne) ; PPO PyTorch, PPO JAX (final/meilleur, final/moyenne) ; CrossQ, PPO JAX (final/moyenne, meilleur/moyenne)
Walker2d	CrossQ	CrossQ	CrossQ	
Swimmer	PPO JAX	PPO JAX	PPO JAX	CrossQ, PPO PyTorch (final/meilleur, final/moyenne)
10 graines, budget égal (200 000 pas) : 4 inversions, 0 tranchée				
HalfCheetah	**SAC**	**SAC**	**CrossQ**	CrossQ, SAC (final/moyenne, meilleur/moyenne)
Hopper	CrossQ	CrossQ	CrossQ	
Walker2d	CrossQ	CrossQ	CrossQ	
Swimmer	CrossQ	CrossQ	CrossQ	PPO PyTorch, SAC (final/meilleur, meilleur/moyenne)
20 graines, budget égal : 6 inversions, 1 tranchée				
HalfCheetah	CrossQ	CrossQ	CrossQ	
Hopper	CrossQ	CrossQ	CrossQ	PPO PyTorch, PPO JAX (final/moyenne, meilleur/moyenne)
Walker2d	CrossQ	CrossQ	CrossQ	PPO PyTorch, PPO JAX (final/moyenne, meilleur/moyenne)
Swimmer	CrossQ	CrossQ	CrossQ	PPO PyTorch, SAC (final/meilleur, **meilleur/moyenne**)

- Les comparaisons proches sont plus difficiles à ordonner : avec cinq graines aléatoires, le bon ordre n'est retrouvé que dans 62 % des tirages en médiane, au lieu de 71 % (E5).
- À budget égal et vingt graines aléatoires, CrossQ mène sur les quatre tâches : le classement ne change plus avec la tâche (E2).

Le dernier point est la nuance qui pèse le plus. Le verdict de E2, tel qu'il était annoncé, tient. Mais à vingt graines aléatoires, le changement de programme en tête d'une tâche à l'autre s'explique en bonne part par le budget cinq fois plus grand de PPO.

6.12 Ce que ces expériences ne disent pas

Les limites propres à chaque expérience sont dites dans sa section. Quelques limites valent pour toutes.

- Tous les essais ont tourné sur une seule machine, un Mac à puce Apple M2 Max, sur processeur. Une relance ailleurs donnerait d'autres courbes pour les mêmes graines aléatoires.
- Les programmes gardent les valeurs par défaut de leurs bibliothèques. Les classements valent pour ces réglages. Ils ne disent rien des algorithmes au mieux de leur forme.
- Quatre tâches MuJoCo, toutes des corps plans ou presque, ne représentent pas l'apprentissage par renforcement de 2026. Le volet des modèles de langue (chapitre 7) en touche une autre part, à petite échelle.
- Dix épisodes d'évaluation, joués sans hasard sur dix positions de départ fixes, mesurent une politique sur une petite partie de ses situations possibles.
- Dix graines aléatoires, puis vingt, laissent des intervalles larges pour les comparaisons serrées. E5 le chiffre.
- Plusieurs analyses ont été ajoutées après avoir vu un résultat : la seconde statistique de E1, le diagnostic de E6, le classement descriptif de E7. La section 5.12 les liste. Le texte les présente comme telles.

CHAPITRE 7

Le volet des modèles de langue

Les chapitres précédents traitent de programmes qui apprennent à agir dans un monde simulé. Ce chapitre décrit un volet plus petit, consacré aux modèles d'IA générative qui écrivent du texte. Il présente le vocabulaire retenu, l'algorithme tel que nous l'avons écrit, le protocole, les essais et leurs résultats, puis le piège de mesure que ces essais ont révélé. Le protocole complet se trouve dans `sim/llm/LLM_PROTOCOL.md`. Ses dix-sept écarts sont résumés au chapitre 9.

7.1 Le vocabulaire

Le vocabulaire du RL a gagné les modèles d'IA générative. On emprunte au RL certains de ses algorithmes pour affiner les réponses d'un modèle de langue, sans agent qui agit dans un monde. Un modèle de langue est un modèle IA qui produit un texte jeton par jeton. Un jeton est un morceau de mot. Pour un problème de mathématiques, un programme compare la réponse finale à la bonne réponse et la récompense vaut 1 quand elles concordent. La littérature parle de RL à récompense vérifiable (RLVR, *reinforcement learning with verifiable rewards*). Nous gardons ce sigle quand nous citons un article qui l'emploie, sans en reprendre l'idée qu'il s'agit du même problème que celui d'un agent dans un monde.

Trois différences séparent ce cadre du RL classique des chapitres 5 et 6 :

- un « épisode » se réduit à une seule réponse : aucun état n'évolue sous l'effet des actions, aucune récompense n'est différée sur plusieurs décisions ;
- la politique de départ est un modèle déjà entraîné sur d'immenses corpus, qui sait souvent répondre avant tout affinage ;
- la récompense ne dépend que du texte final.

Nous n'appelons jamais RL l'affinage par retours humains (RLHF, *reinforcement learning from human feedback*, Ouyang et al. 2022) : nous le désignons par son sigle, comme une technique d'affinage parmi d'autres.

7.2 Pourquoi ce volet

En 2025, Shao et al. (2025) ont montré qu'une récompense tirée au sort, sans regarder si la réponse est juste, améliore les modèles Qwen en mathématiques, mais pas les modèles Llama ou OLMo. Sur le modèle Qwen2.5-Math-7B et le banc d'essai MATH-500, la vraie récompense fait gagner 29,1 points de pourcentage. Une récompense qui félicite une réponse fausse fixée d'avance en fait gagner 24,1, une récompense tirée au hasard 21,4. Les auteurs l'expliquent par un comportement que l'entraînement amplifie sans le créer : le modèle écrit son raisonnement sous forme de code, sans l'exécuter, dans 65 % des réponses avant l'entraînement et dans plus de 90 % après un entraînement

à récompense trompeuse. Ils recommandent de valider une méthode sur plusieurs familles de modèles.

Ce résultat prolonge deux failles de 2018 : le meilleur algorithme change avec la tâche (F13) et les bancs d'essai trop simples se laissent apprendre (F16). Le sondage du chapitre 4 mesure le même risque dans les articles : seize articles sur 27 du premier tirage de 2026 qui appliquent ce qu'ils appellent RL à de grands modèles de langue n'essaient qu'une seule famille de modèles (critère X6). Notre volet regarde le phénomène de près, à petite échelle, avec deux familles de modèles. Le protocole annonce d'avance sa portée : une illustration sur un ordinateur portable. Elle ne réplique pas Shao et al. (2025).

7.3 L'algorithme GRPO, tel que nous l'avons écrit

GRPO (*Group Relative Policy Optimization*) vient de Shao et al. (2024). Il reprend la fonction objectif coupée de PPO (Schulman et al. 2017) et remplace le réseau critique, qui estime la valeur d'un état, par une normalisation au sein d'un groupe de réponses au même problème. Nous l'avons écrit à la main, en PyTorch, dans un seul fichier de 159 lignes, `sim/llm/grpo.py`, pour contrôler chaque étape. Aucune bibliothèque d'affinage n'intervient : le fichier n'importe que PyTorch et la bibliothèque Transformers, qui charge le modèle et son découpeur en jetons.

7.3.1 La tâche et la récompense

La tâche retenue, `mul2`, demande le produit de deux entiers de 12 à 99. Le fichier énumère les $88 \times 88 = 7\,744$ couples ordonnés, les mélange avec un générateur de graine aléatoire fixe (`random.Random(12345)`), indépendant de la graine aléatoire de l'essai, puis réserve les 300 premiers au test. Les 7 444 autres servent à l'entraînement. L'énoncé suit un gabarit fixe, « What is a * b? Think step by step, then finish with the final answer as an integer. », passé dans le gabarit de conversation du modèle.

Une réponse est juste quand le dernier entier qu'elle contient, virgules de milliers retirées, est égal au produit. Aucun format n'est exigé, pour ne pas récompenser le format. Soit $c(o, q) \in \{0, 1\}$ ce verdict pour la réponse o au problème q. Les deux conditions du protocole définissent la récompense r de chaque réponse :

$$\text{vraie récompense (\texttt{true}) :} \quad r = c(o, q), \tag{7.1}$$

$$\text{récompense tirée au sort (\texttt{random}) :} \quad r \sim \text{Bernoulli}(1/2), \text{ tirée sans regarder } o. \tag{7.2}$$

Le tirage de la pièce et celui des problèmes de chaque pas viennent du même générateur, `random.Random(graine)`, qui dépend de la seule graine aléatoire de l'essai.

7.3.2 Un pas d'entraînement

Chaque pas tire sans remise $B = 4$ problèmes d'entraînement. Pour chaque problème q_b, la politique courante $\pi_{\theta_{\text{old}}}$ produit $G = 8$ réponses $o_{b,1}, \ldots, o_{b,8}$, échantillonnées à la température 1, sans troncature de la distribution (ni *top-p* ni *top-k*), limitées à 160 jetons. Chaque réponse reçoit sa récompense $r_{b,i}$. L'avantage d'une réponse compare

sa récompense à celles de son groupe :

$$\hat{A}_{b,i} = \frac{r_{b,i} - \bar{r}_b}{s_b + 10^{-4}}, \qquad \bar{r}_b = \frac{1}{G}\sum_{j=1}^{G} r_{b,j}, \qquad s_b = \Big(\frac{1}{G-1}\sum_{j=1}^{G}(r_{b,j} - \bar{r}_b)^2\Big)^{1/2}. \tag{7.3}$$

L'écart type s_b suit la convention de PyTorch, avec le facteur $G - 1$. Un groupe où les huit réponses reçoivent la même récompense donne un avantage nul à chacune et n'apporte aucun gradient.

Pour le jeton t de la réponse $o_{b,i}$, le rapport de probabilités compare la politique en cours de mise à jour à celle qui a produit la réponse :

$$\rho_{b,i,t}(\theta) = \exp\big(\log \pi_\theta(o_{b,i,t} \mid q_b, o_{b,i,<t}) - \log \pi_{\theta_{\text{old}}}(o_{b,i,t} \mid q_b, o_{b,i,<t})\big). \tag{7.4}$$

La perte minimisée est l'opposé de l'objectif coupé, moyenné par jeton dans chaque réponse puis par réponse dans le pas :

$$\mathcal{L}(\theta) = -\frac{1}{BG}\sum_{b=1}^{B}\sum_{i=1}^{G}\frac{1}{|o_{b,i}|}\sum_{t=1}^{|o_{b,i}|}\min\Big(\rho_{b,i,t}(\theta)\,\hat{A}_{b,i},\ \text{clip}\big(\rho_{b,i,t}(\theta), 1-\varepsilon, 1+\varepsilon\big)\,\hat{A}_{b,i}\Big), \qquad \varepsilon = 0{,}2. \tag{7.5}$$

La longueur $|o_{b,i}|$ compte les jetons de la réponse qui ne sont pas du remplissage (le masque `target != pad_token_id` de la fonction `token_logps`). Les log-probabilités de référence $\log \pi_{\theta_{\text{old}}}$ sont calculées une fois, sans gradient, avant la première mise à jour.

Chaque lot de 32 réponses sert à deux passes de mise à jour (`EPOCHS = 2`). À chaque passe, le gradient s'accumule sur des sous-lots de huit réponses (deux pour Llama, réglage `MICRO`), ce qui donne le même gradient avec moins de mémoire. La norme du gradient est ensuite coupée à 1, puis l'optimiseur Adam (Kingma et Ba 2015), avec ses moments par défaut, applique un taux d'apprentissage de 3×10^{-6}. Les poids restent en virgule flottante sur 32 bits.

Ce qui distingue notre GRPO de celui de Shao et al. (2024). Notre perte n'a aucun terme de divergence de Kullback-Leibler par rapport au modèle de départ, ce que le protocole du 8 novembre 2026 annonçait. Le groupe compte 8 réponses par problème et le pas 4 problèmes, bien moins que dans les articles de référence. La limite de 160 jetons est très courte pour un raisonnement écrit. Le taux d'apprentissage n'a pas été choisi par le pilote prévu, interrompu au pas 10 (écarts 4 et 6). Ces choix suffisent à une illustration. Ils interdisent de lire nos chiffres comme une mesure de GRPO à l'échelle des articles.

7.3.3 Les garde-fous contre un calcul corrompu

Les premiers essais sur le processeur graphique du Mac (GPU, via l'interface MPS de PyTorch) ont planté sur des probabilités non finies. L'écart 10 du protocole en documente la cause : macOS interrompt un calcul GPU qui gêne l'usage interactif de la machine. Ce calcul revient parfois sans erreur, avec des valeurs corrompues. Le même pas, refait sur le processeur central (CPU), donne une norme de gradient de 14,4 avant écrêtage, sans aucune valeur non finie. Depuis cet écart, `grpo.py` applique quatre règles :

1. une mise à jour dont la norme du gradient avant écrêtage est non finie ou dépasse 1 000 est recalculée, jusqu'à cinq fois de suite, puis l'essai s'arrête (`GRAD_SANE`, `RETRIES`) ;

2. une génération qui lève une erreur GPU est refaite, jusqu'à cinq fois ;
3. l'essai s'arrête si les log-probabilités de référence ou les poids deviennent non finis ;
4. le journal compte les recalculs à chaque pas (champ `retries`).

Le seuil de 1 000 vient de la norme de 14,4 mesurée sur CPU. Un gradient corrompu mais fini, sous ce seuil, puis écrêté à 1, passerait sans trace : le protocole le note comme une limite, en particulier pour les trois premiers essais, qui ont tourné avant ces règles.

7.3.4 La mesure

L'évaluation se fait sur le jeu de test et alterne deux modes. La justesse gloutonne (*greedy*) prend une seule réponse sans hasard, le jeton le plus probable à chaque pas, pour chacun des 150 premiers problèmes de test. La justesse échantillonnée tire 8 réponses à la température 1 pour chacun des 50 premiers problèmes de test. Avec c_{jk} le verdict de la réponse k au problème j et τ_{jk} l'indicateur d'une réponse coupée :

$$\text{pass@1} = \frac{1}{50}\sum_{j=1}^{50}\frac{1}{8}\sum_{k=1}^{8} c_{jk}, \qquad \text{pass@8} = \frac{1}{50}\sum_{j=1}^{50}\max_{k} c_{jk}, \qquad \text{part coupée} = \frac{1}{400}\sum_{j,k}\tau_{jk}. \tag{7.6}$$

Une réponse est coupée quand ses 160 jetons ne contiennent ni jeton de fin ni jeton de remplissage (fonction `truncated`). Le pass@8 est ici la fréquence observée sur exactement huit tirages, sans l'estimateur sans biais employé quand on tire plus de réponses que k. L'évaluation a lieu au pas 0, au pas 50 et au pas 100 (`EVAL_EVERY = 50`). Tous les dix pas, le journal note aussi la part de réponses justes et coupées du lot d'entraînement, la perte, la norme du gradient, les recalculs et le temps écoulé. Chaque essai écrit un fichier JSONL, `sim/llm/results/<modèle>/mul2/<récompense>/seed<graine>.jsonl`.

La mesure principale est le pass@1 échantillonné, celle que GRPO entraîne (écart 3). La justesse gloutonne et le pass@8 restent dans les journaux. Le pass@8 répond à la question de Yue et al. (2025) : l'affinage élargit-il ce que le modèle sait faire, ou concentre-t-il sa probabilité sur des réponses qu'il trouvait déjà en plusieurs tirages ?

7.4 Le protocole et ses essais

Modèles. Deux familles, deux petits modèles ouverts : Qwen2.5 de 0,5 milliard de paramètres, version *Instruct* (Qwen Team 2025), puis Llama 3.2 de 1 milliard, version *Instruct*, dans la copie publiée par `unsloth` (Grattafiori et al. 2024 ; Meta AI 2024). Depuis l'écart 15 (c), `grpo.py` fixe la révision exacte des fichiers téléchargés (`7ae5576…` pour Qwen, `5a8abab…` pour Llama), les seules présentes dans le cache de la machine, téléchargées les 22 et 27 octobre 2026, avant le premier essai.

Pilote. Un pilote sans entraînement, sur des problèmes hors test, devait fixer la tâche : celle où les deux modèles se situent entre 15 % et 70 % de justesse. Aucune tâche ne place les deux modèles dans cette plage en glouton. La somme de trois nombres (`add3`) est trop facile pour Llama. La multiplication `mul2` donne en glouton 7 % (Qwen) et 73 % (Llama), en pass@1 échantillonné 11 % et 23 % (écart 3). Le même pilote a montré que 64 jetons coupaient les réponses de Qwen, qui en écrit 100 à 160, d'où 0,7 % de

justesse apparente : la limite est passée à 160 jetons, au vu de vraies sorties, avant tout entraînement (écart 1).

Plan. Deux modèles, deux récompenses, 100 pas par essai, taux d'apprentissage 3×10^{-6} pour toutes les conditions. Le protocole prévoyait 200 pas et cinq graines aléatoires. L'écart 6 a réduit le plan à 100 pas et trois graines aléatoires (0 à 2), avant tout essai retenu, pour tenir dans le calcul disponible. L'écart 14, écrit le 15 novembre 2026 avant tout nouvel essai, a rétabli les graines aléatoires 3 et 4, avec le même code et les mêmes réglages. Le point de départ de chaque essai est le modèle de base, mesuré avec la graine aléatoire de l'essai.

Statistique. Pour chaque graine aléatoire s, le gain vaut $g_s = \text{pass@1}_{100} - \text{pass@1}_{0}$. Chaque configuration (modèle, récompense) se résume par la moyenne des gains et un intervalle bootstrap par percentile à 95 % : 10 000 rééchantillonnages avec remise des gains, générateur `random.Random(0)`, bornes aux rangs 250 et 9 750 des moyennes triées (`sim/llm/summary.py`). Les détails du bootstrap ont été fixés après les premiers résultats, sans choix entre plusieurs variantes (écart 13). L'intervalle est très grossier : trois graines aléatoires n'offrent que dix rééchantillons distincts, cinq graines aléatoires 126.

Un essai qui échoue deux fois. L'essai Llama à récompense tirée au sort et graine aléatoire 4 s'est arrêté deux nuits de suite sur le premier garde-fou, dès sa première mise à jour. L'écart 16 compte ce second échec comme un résultat : la configuration se résume sur ses quatre essais finis, l'échec écrit à côté. Une analyse de sensibilité remplace aussi le gain manquant par le plus petit puis par le plus grand gain des dix-neuf essais finis.

Matériel. Tous les essais ont tourné sur le Mac du projet (chapitres D et 8). Les trois premiers essais Qwen ont tourné sur le GPU, tôt le matin, machine au repos. Après les plantages de l'écart 8 et le diagnostic de l'écart 10, les essais Qwen restants sont passés sur CPU, avec deux fils d'exécution (`cpu.sh`, écart 11). Les essais Llama, environ deux fois et demie plus lents que Qwen sur CPU, sont restés sur GPU, la nuit seulement (`night.sh`), avec les garde-fous. Le tableau 7.1 donne l'appareil de chaque essai.

7.5 Les résultats

Le tableau 7.1 donne, pour chacun des vingt essais, le pass@1 au départ et après 100 pas, l'appareil et la part de réponses coupées à la fin. La figure 7.1 montre chaque essai par une flèche, la figure 7.2 les trois points d'évaluation de chaque essai.

7.5.1 La récompense tirée au sort

À pile ou face, la réussite tombe à 1 % ou moins dans les neuf essais finis. Le gain moyen vaut -10 points sur Qwen (de -11 à -9) et -24 points sur Llama (de -24 à -23). Remplacer le gain manquant de Llama par le plus petit gain des dix-neuf essais finis change peu (-24, de -24 à -23). Le remplacer par le plus grand, $+51$, relève la moyenne à -9, avec un intervalle de -24 à $+21$. Chez nous, à cette échelle, la récompense tirée au sort détruit la compétence de départ des deux familles. Elle ne reproduit pas le gain que Shao et al. (2025) observent sur Qwen2.5-Math-7B. Nos conditions diffèrent des

Table 7.1. Volet GRPO (`sim/llm/LLM_PROTOCOL.md`) : deux petits modèles ouverts, Qwen2.5 à 0,5 milliard de paramètres et Llama 3.2 à 1 milliard, entraînés 100 pas sur la tâche `mul2` (produit de deux nombres de 12 à 99), avec la vraie récompense puis une récompense tirée à pile ou face, cinq graines (0 à 2 puis 3 et 4, écart 14). Réussite au premier essai : pass@1 échantillonné sur les problèmes de test, en pourcentage ; gain : pas 100 moins pas 0, en points ; réponses coupées : part des réponses qui atteignent la limite de 160 jetons sans donner de résultat, toutes comptées fausses. Matériel : GPU Apple (MPS) ou CPU ; trois essais Qwen sont passés sur CPU après des gradients non finis sur MPS (écarts 10 et 11), si bien que matériel et graine se confondent pour Qwen. Partie basse : statistique préregistrée, moyenne des gains par graine et intervalle bootstrap percentile à 95 % (10 000 tirages, `random.Random(0)`), très grossier avec cinq graines ; **en gras**, les intervalles qui excluent zéro. La configuration Llama, récompense au hasard, est résumée sur ses quatre essais finis (écart 16), avec la sensibilité au gain manquant. Sources : `sim/llm/results/summary_5.json`, `gains_5.json`, `extra_5.json` ; part coupée au pas 0 : première évaluation de chaque fichier `sim/llm/results/<modèle>/mul2/<récompense>/seed<g>.jsonl`.

				Réussite au premier essai (%)			Réponses coupées (%)	
Modèle	Récompense	Graine	Matériel	pas 0	pas 100	gain (pts)	pas 0	pas 100
Qwen2.5 0,5B	vraie	0	GPU (MPS)	10	36	+26	77	29
		1	GPU (MPS)	10	61	+51	75	0
		2	CPU	11	13	+2	76	0
		3	CPU	10	13	+4	74	0
		4	CPU	11	0	−11	73	0
	au hasard	0	GPU (MPS)	10	1	−9	77	59
		1	CPU	12	0	−11	71	88
		2	CPU	11	0	−11	76	98
		3	CPU	10	0	−10	74	16
		4	CPU	11	0	−11	73	86
Llama 3.2 1B	vraie	0	GPU (MPS)	25	20	−4	2	9
		1	GPU (MPS)	25	48	+22	1	0
		2	GPU (MPS)	23	25	+2	2	3
		3	GPU (MPS)	25	27	+2	3	1
		4	GPU (MPS)	23	33	+10	3	1
	au hasard	0	GPU (MPS)	25	0	−25	2	99
		1	GPU (MPS)	25	1	−24	1	100
		2	GPU (MPS)	23	0	−23	2	2
		3	GPU (MPS)	24	0	−24	2	16
		4	GPU (MPS)	échec deux fois (écart 16)				

Gain moyen en points [IC bootstrap 95 %]	
Qwen2.5 0,5B, vraie, 5 graines	+14 [−3 ; +34]
Qwen2.5 0,5B, au hasard, 5 graines	**−10 [−11 ; −9]**
Llama 3.2 1B, vraie, 5 graines	+6 [−0,4 ; +15]
Llama 3.2 1B, au hasard, 4 graines	**−24 [−24 ; −23]**
Qwen2.5 0,5B, vraie, CPU seul, graines 2, 3, 4	−2 [−11 ; +4]
Qwen2.5 0,5B, au hasard, CPU seul, graines 1, 2, 3, 4	**−11 [−11 ; −10]**
Llama 3.2 1B, au hasard, graine 4 = plus petit gain, −25	−24 [−24 ; −23]
Llama 3.2 1B, au hasard, graine 4 = plus grand gain, +51	−9 [−24 ; +21]

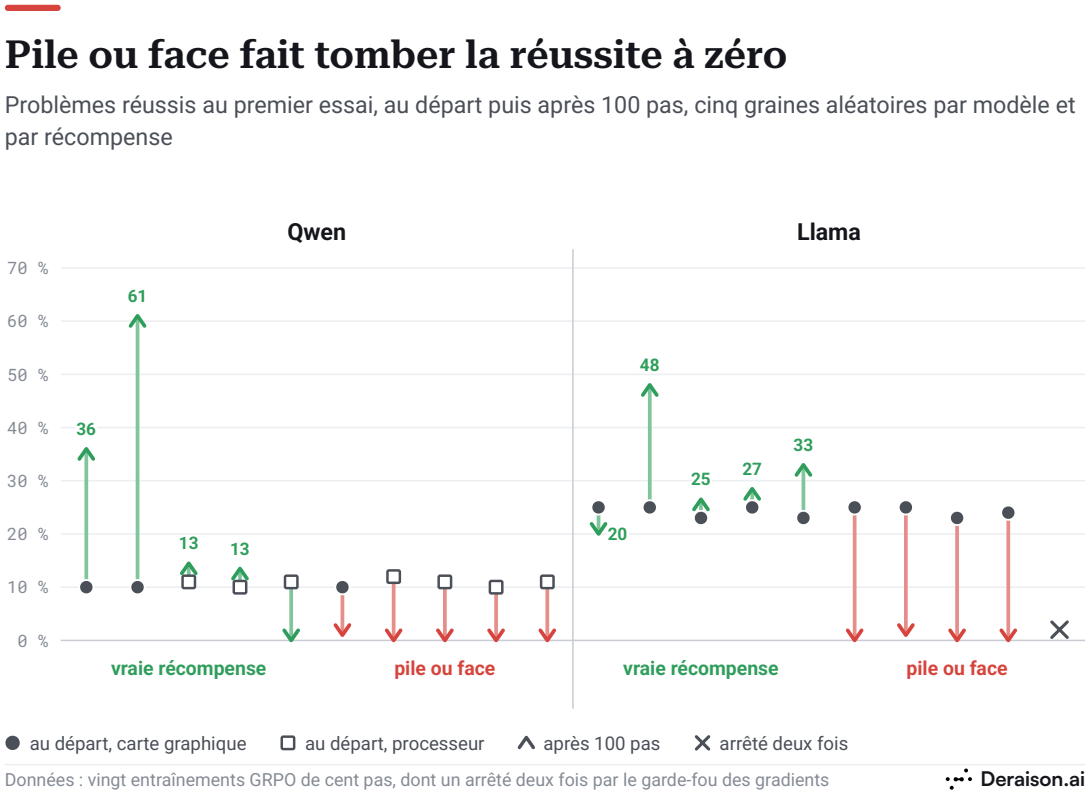


FIGURE 7.1. Pass@1 au départ et après cent pas d'entraînement, pour chaque modèle, chaque récompense et chacune des cinq graines aléatoires. Un rond plein désigne un essai sur GPU, un carré vide un essai sur CPU, une croix l'essai arrêté deux fois. Avec la vraie récompense, le même programme donne des résultats très différents d'une graine aléatoire à l'autre. À pile ou face, la réussite tombe à zéro ou presque.

leurs sur presque tout (taille et spécialisation du modèle, tâche, longueur des réponses, nombre de pas, absence de terme KL) : ce désaccord ne réfute pas leur résultat.

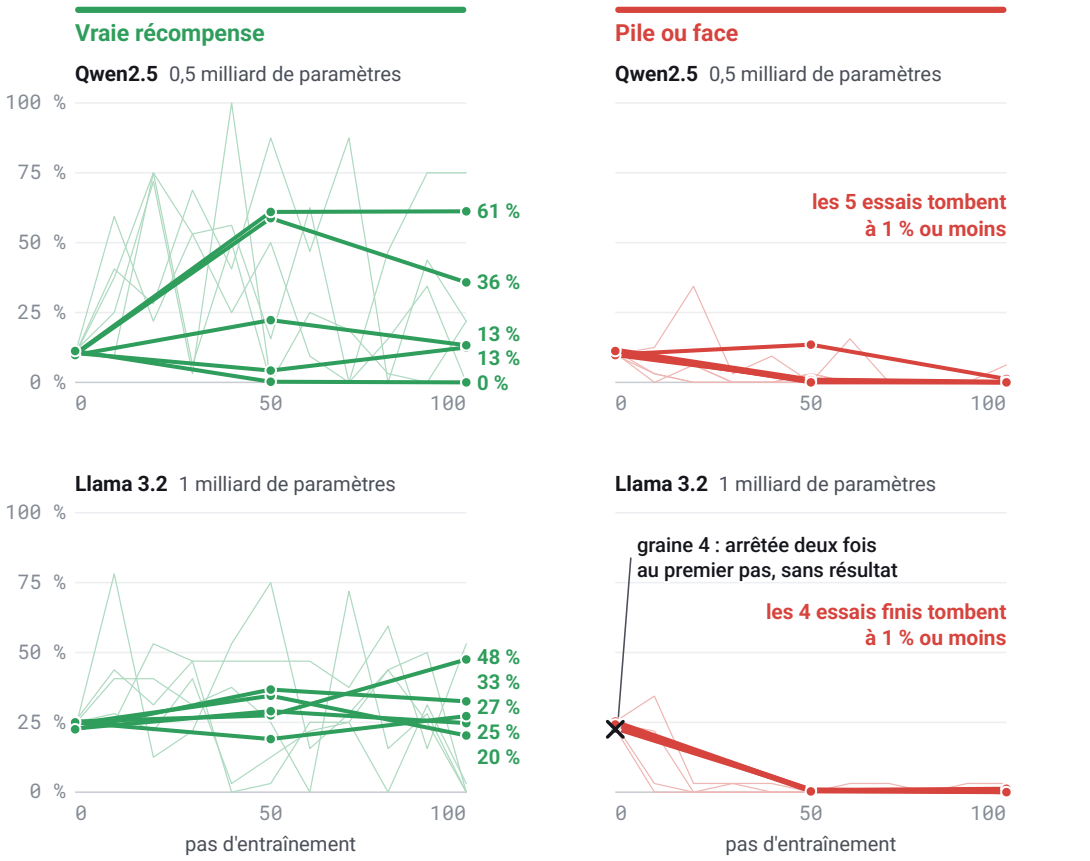
7.5.2 La vraie récompense, en deux temps

Avec trois graines aléatoires, Qwen part d'environ 10 % et finit à 36 %, 61 % et 13 %. Le gain moyen vaut $+26$ points, avec un intervalle de $+2$ à $+51$. Sur Llama, la réussite part de 23 % à 25 % et finit à 20 %, 48 % et 25 %, soit un gain moyen de $+7$ points, de -4 à $+22$. Le même programme donne trois résultats très différents : c'est la faille F11, même code, autre graine aléatoire, en petit.

Avec deux graines aléatoires de plus, les nouveaux essais Qwen finissent à 13 % et 0 %. Sur cinq graines aléatoires, le gain moyen tombe à $+14$ points, de -3 à $+34$: l'intervalle contient désormais zéro. Le gain que nous lisions sur Qwen avec trois graines aléatoires ne tient plus. Sur Llama, les deux nouveaux essais finissent à 27 % et 33 %. Le gain moyen vaut $+6$ points, de $-0{,}4$ à $+15$: zéro reste dans l'intervalle. Avec cinq graines aléatoires, nous ne pouvons pas dire que GRPO fait progresser l'un ou l'autre modèle sur cette tâche, dans ces conditions.

Vraie récompense : à chaque graine son résultat. Pile ou face : la réussite s'éteint.

Part des problèmes résolus à la première tentative pendant les cent pas de GRPO, une ligne par graine aléatoire (cinq par configuration). Trait épais : évaluation sur les problèmes tenus à part, aux pas 0, 50 et 100. Trait fin : réussite sur le lot d'entraînement, tous les dix pas.



Nombre à droite : réussite au pas 100, arrondie. Croix : essai sans résultat. Le lot d'entraînement compte 32 réponses : son trait fin saute d'un pas à l'autre.

Données : sim/llm/results/<modèle>/mul2/<récompense>/seed<k>.jsonl, summary_5.json Deraison.ai

FIGURE 7.2. Pass@1 de test aux pas 0, 50 et 100 de chaque essai, par modèle, récompense et graine aléatoire. L'évaluation n'a lieu que tous les cinquante pas : la courbe relie trois points.

7.5.3 Le matériel et la graine aléatoire, mêlés

Le gain de Qwen avec la vraie récompense vient des deux essais faits sur GPU, graines aléatoires 0 et 1 : $+26$ et $+51$ points. Les trois essais à vraie récompense qui ont tourné sur CPU (graines aléatoires 2 à 4) ont un gain moyen de -2 points, de -11 à $+4$. Les quatre essais sur CPU à récompense tirée au sort (graines aléatoires 1 à 4) ont un gain moyen de -11 points, de -11 à -10. Sur CPU seul, la vraie récompense n'apporte à Qwen aucun gain détectable. Données : `sim/llm/results/gains_5.json` et `extra_5.json`.

Ces chiffres ne séparent pas deux causes. Le matériel a changé avec la graine aléatoire : chaque graine aléatoire de Qwen à vraie récompense n'a tourné que sur un appareil. Deux explications restent ouvertes. Selon la première, le hasard des graines aléatoires suffit : les deux graines favorables sont tombées sur le GPU. Selon la seconde, l'appareil change la trajectoire de l'entraînement, par l'arrondi des calculs ou par des calculs GPU corrompus mais finis, que les premiers essais n'auraient pas détectés. Cinq essais ne départagent pas ces explications.

Les journaux montrent que l'appareil modifie déjà la mesure de départ. Le pass@1 au pas 0 dépend du modèle de base et de la graine aléatoire seulement, puisque la récompense n'intervient qu'après cette première évaluation : pour un même modèle et une même graine aléatoire, les deux conditions partent du même point si le calcul est déterministe. C'est le cas quand les deux essais ont tourné sur le même appareil et sans incident, par exemple Qwen graines aléatoires 2, 3 et 4 sur CPU. Pour Qwen graine aléatoire 1, l'essai à vraie récompense, sur GPU, part de 0,102 5, l'essai à récompense tirée au sort, sur CPU, de 0,117 5. Pour Llama graine aléatoire 3, les deux essais ont tourné sur GPU et partent de 0,25 et de 0,242 5 : le calcul MPS n'est pas déterministe d'une tentative à l'autre, ce que l'écart 7 avait déjà constaté (tableau 7.1).

Un essai hors protocole. L'écart 16 (e) prévoyait de lancer une fois sur CPU l'essai Llama à récompense tirée au sort et graine aléatoire 4, après tous les autres calculs, pour savoir si son double échec tenait au GPU. L'essai a abouti le 21 novembre 2026, sans aucun recalcul. Son pass@1 passe de 0,232 5 au pas 0 à 0,002 5 au pas 100 (`sim/llm/results/llama1_cpu/mul2/random/seed4.jsonl`). Le résultat sort de toute statistique, comme l'écart l'avait fixé. Il cadre avec un échec dû au GPU, sans le prouver : un seul essai ne sépare pas l'appareil du hasard. Sur CPU, ce dixième essai se comporte comme les neuf autres essais à récompense tirée au sort : la réussite s'effondre.

7.6 Le piège de la coupure

7.6.1 Le diagnostic

L'incident de l'écart 4 a mis ce piège au jour. Pendant le pilote du taux d'apprentissage, la part de réponses justes du lot d'entraînement valait 0,91 au pas 10, alors que le pass@1 de test du modèle de base valait 0,11. Le diagnostic de l'écart 5 (`sim/llm/diag/replay_batch.py`) rejoue, sans entraînement, le lot exact du pas 10 du pilote : les quatre multiplications 48×49, 14×36, 21×50 et 45×17, huit réponses chacune, avec le modèle Qwen de départ. Le modèle donne 5 réponses justes sur 32, ce qui concorde avec le test. Mais 23 réponses sur 32 atteignent la limite de 160 jetons avant de donner un résultat. Aucune de ces réponses coupées n'est juste. Parmi les 9 réponses complètes, 5 sont justes (figures 7.3 et 7.4).

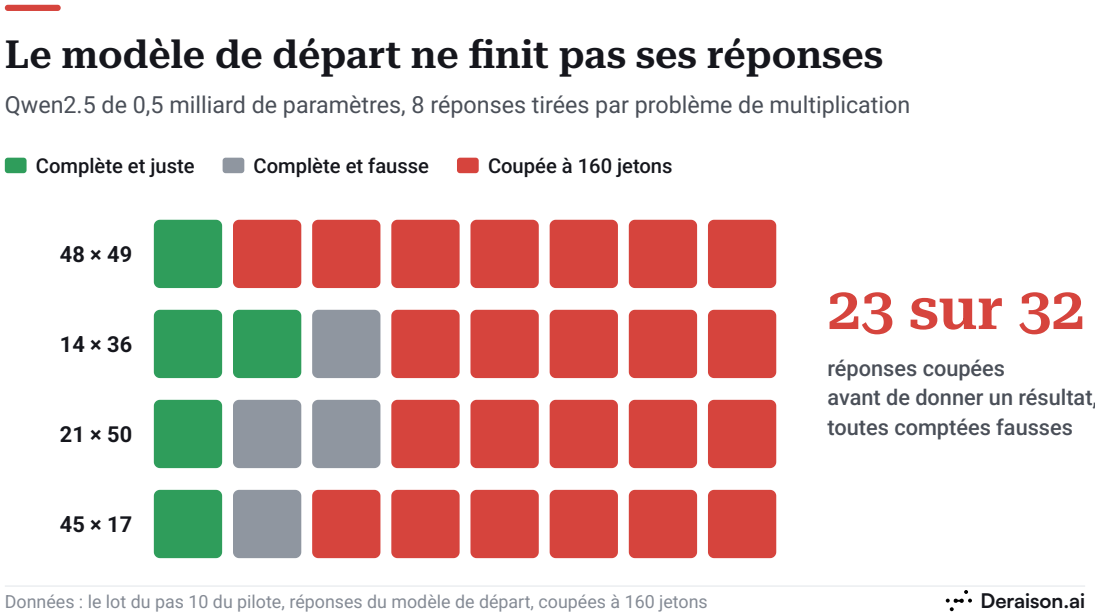


FIGURE 7.3. Les réponses du modèle Qwen2.5 de 0,5 milliard de paramètres avant tout entraînement, huit par problème, sur les quatre multiplications du dixième pas du pilote. Une réponse complète et juste est en vert, une réponse complète et fausse en gris, une réponse coupée en rouge. La plupart atteignent la limite de 160 jetons avant de donner un résultat et comptent pour fausses.

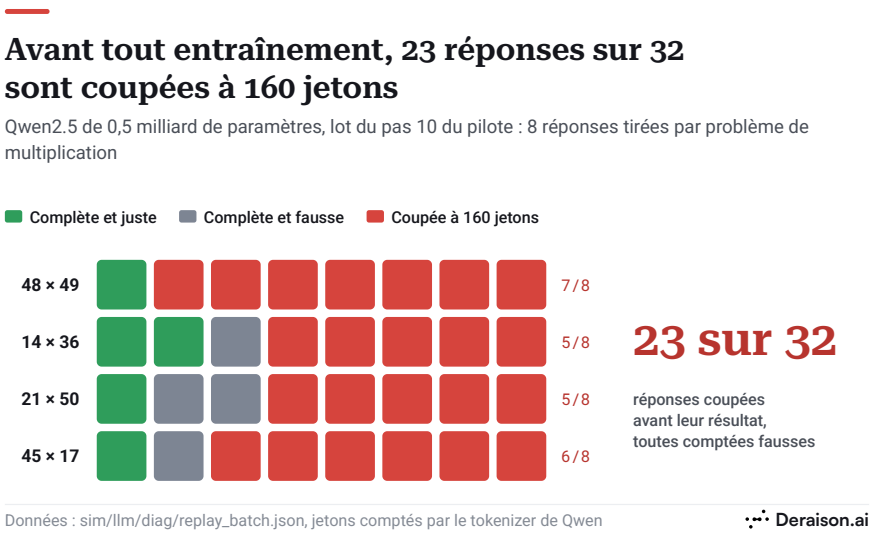


FIGURE 7.4. Les mêmes trente-deux réponses, en barres empilées : pour chaque problème, la part des huit réponses complètes et justes, complètes et fausses, coupées à 160 jetons.

Le 0,91 du pas 10 n'était pas une erreur de décompte. L'explication la plus simple : la vraie récompense apprend d'abord au modèle à terminer avant la coupure, si bien que le lot d'entraînement était devenu plus facile à finir. La récompense tirée au sort n'a pas ce levier, puisqu'elle ne voit pas la réponse. Le protocole a gardé les 160 jetons et ajouté la mesure de la part coupée, à l'entraînement et au test, avant tout essai retenu.

7.6.2 Ce que montrent les vingt essais

Au départ, la part de réponses coupées est élevée pour Qwen et faible pour Llama : de 71 % à 77 % pour Qwen, 3 % au plus pour Llama, d'après les journaux d'évaluation au pas 0. Avec la vraie récompense, sur les dix essais, elle finit entre 0 % et 29 %. À pile ou face, elle finit au-dessus de la moitié dans six essais sur neuf, entre 2 % et 16 % dans les trois autres. Deux essais Llama à récompense tirée au sort finissent avec 99 % et 100 % de réponses coupées. Un troisième n'en coupe que 2 % et ne donne plus aucune bonne réponse : la récompense tirée au sort ne se contente pas d'allonger les réponses au-delà de la limite.

Une réponse coupée compte pour fausse. Un modèle qui apprend à répondre plus court gagne donc des points de pass@1 sans mieux calculer. Une part du gain mesuré avec la vraie récompense peut venir de ce détail du banc d'essai : apprendre à se taire à temps. Nos journaux ne mesurent pas la justesse des seules réponses complètes à chaque évaluation, si bien que nous ne pouvons pas chiffrer cette part. Pour Qwen, dont les réponses de départ sont coupées à plus de 70 % au test, elle peut être grande.

> **Leçon pour un banc d'essai de modèle de langue.** Une limite de longueur transforme une mesure de justesse en une mesure mixte de justesse et de concision. Un article qui rapporte un gain de pass@1 se lit avec la limite de longueur, la part de réponses coupées avant et après l'entraînement et la justesse des réponses complètes. Sans ces trois nombres, le lecteur ignore ce que le gain mesure.

7.7 Ce que ce volet établit et ce qu'il n'établit pas

Ce volet établit trois faits, pour deux petits modèles, une tâche de multiplication et 100 pas. Avec la vraie récompense, l'écart entre graines aléatoires est aussi grand que le gain moyen, comme dans le RL classique de 2018. Une récompense tirée au sort fait chuter la réussite des deux familles. Une part du gain apparent peut venir de la limite de longueur, que le protocole n'avait pas prévue.

Il n'établit ni l'effet de GRPO à l'échelle des articles, ni l'absence de l'effet de Shao et al. (2025) sur des modèles plus grands ou spécialisés en mathématiques. Il ne sépare pas l'effet du matériel de celui de la graine aléatoire pour Qwen. Les limites qui en découlent sont reprises au chapitre 10.

CHAPITRE 8

Logiciel et reproductibilité

Pineau demandait en 2018 un travail reproductible, réutilisable et robuste. Ce chapitre décrit comment le dépôt du projet, `github.com/warith-harchaoui/pineau-rl`, tente de satisfaire ces trois exigences. Il présente l'organisation du dépôt, le paquet Python qui réunit les calculs réutilisables, les formats de fichiers, la vérification mécanique de chaque nombre publié, les tests, l'intégration continue, les environnements figés, le matériel et le calcul consommé. Un dernier point dit ce qui manque au dépôt et comment l'obtenir.

8.1 L'organisation du dépôt

Le dépôt suit un flux à sens unique, des essais vers les documents (figure 8.1). Un essai écrit un fichier de résultats brut : une courbe d'évaluation au format CSV pour une simulation, un journal JSONL pour un essai GRPO, un fichier JSON de lecture pour un article du sondage. Un script d'analyse lit ces fichiers et écrit un fichier JSON de résultats. Deux scripts de vérification recalculent ensuite chaque nombre des textes à partir de ces fichiers JSON. Aucun nombre d'un document ne vient d'ailleurs.

8.2 Le paquet `pineau_rl`

Les calculs qui servent à plusieurs scripts forment un paquet Python installable, `src/pineau_rl/`, version 1.0.0, sous licence BSD 3-Clause. Ses seules dépendances sont NumPy, SciPy, statsmodels et rliable. Chaque script d'analyse l'importe. Le `Makefile` place `src/` dans le chemin de recherche de Python, si bien que les scripts tournent sans installer le paquet. Les sous-modules se chargent à la demande : `import pineau_rl.stats` ne charge pas rliable. Le paquet compte quatre modules publics (`stats`, `bootstrap`, `runs`, `grid`) et un module de mise en forme (`fmt`).

8.2.1 `pineau_rl.stats`

Ce module réunit les statistiques du sondage et des expériences. Chaque fonction est déterministe. L'en-tête du module avertit que les calculs suivent l'ordre exact des scripts d'origine, pour que les fichiers de résultats se régénèrent à l'octet près : simplifier une expression change le dernier chiffre d'un résultat.

- `wilson(k, n, z=1.96)` et `wilson_ci(k, n, alpha=0.05)` : l'intervalle de Wilson (1927) d'une proportion k/n. Avec $\hat{p} = k/n$, son centre vaut $(\hat{p} + z^2/2n)/(1 + z^2/n)$ et sa demi-largeur $z\sqrt{\hat{p}(1-\hat{p})/n + z^2/4n^2}\,/(1 + z^2/n)$. La seconde fonction délègue à statsmodels, avec $z = \Phi^{-1}(1 - \alpha/2)$.
- `newcombe_diff(ka, na, kb, nb)` : l'intervalle à 95 % de la différence de deux proportions par la méthode hybride de Newcombe (1998), construite à partir

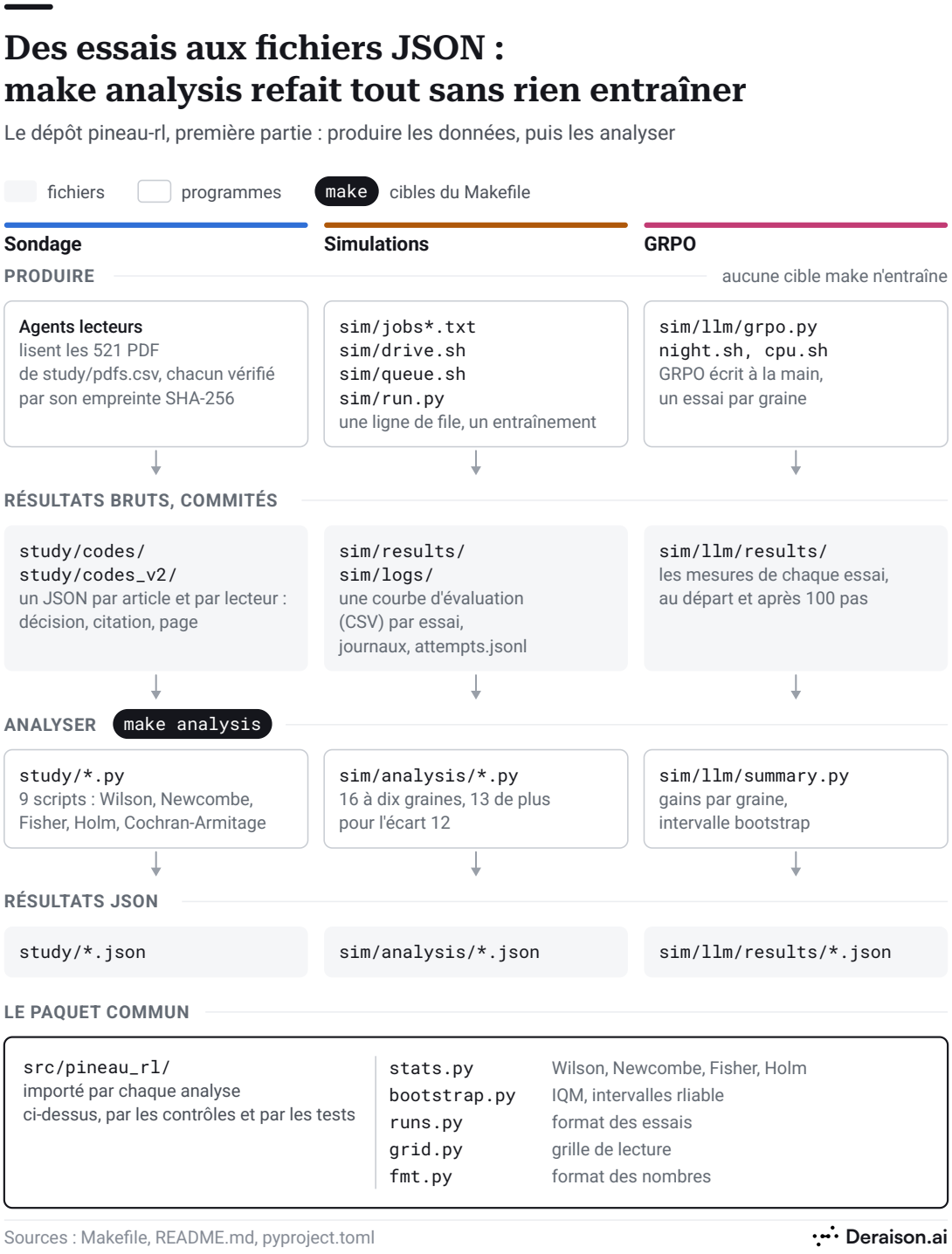


FIGURE 8.1. Le dépôt, première partie : des essais aux fichiers JSON. Les entraînements écrivent les résultats bruts (CSV, JSONL, journaux). Les scripts d'analyse, appuyés sur le paquet `pineau_rl`, en tirent un fichier JSON par expérience. `make analysis` refait toute cette chaîne sans aucun entraînement.



FIGURE 8.2. Le dépôt, seconde partie : des fichiers JSON aux documents. `check_numbers.py` et `check_article.py` confrontent chaque nombre des textes à son fichier JSON, `tools/tables.py` génère les tableaux de ce rapport. `make verify` enchaîne analyses, contrôles et tests puis vérifie que chaque JSON revient à l'identique. L'intégration continue refait ce chemin à chaque envoi sur GitHub.

TABLE 8.1. Les dossiers du dépôt et leur rôle.

Chemin	Contenu
`study/`	Le sondage : protocole (`PROTOCOL.md`), fiche du lecteur (`GRILLE.md`), tirages, lectures (`codes/`, `codes_v2/`), harmonisation (`overrides*.json`), provenance des lecteurs (`readers/`), scripts d'analyse et fichiers JSON de résultats.
`sim/`	Les simulations : protocole (`SIM_PROTOCOL.md`), un essai (`run.py`), une réévaluation sous physique modifiée (`dose.py`), listes d'essais (`jobs*.txt`), lanceurs et surveillant (`drive.sh`, `queue.sh`, `worker.sh`, `scale.sh`, `watchdog.sh`), courbes (`results/`), journaux (`logs/`), politiques finales (`models/`).
`sim/analysis/`	Un script et un fichier JSON par expérience (E1 à E7, B1, B3, dose, correction de Bonferroni), avec leurs variantes à vingt graines aléatoires et à budget égal.
`sim/llm/`	Le volet GRPO : protocole, `grpo.py`, lanceurs, diagnostics (`diag/`), journaux, résultats et `summary.py`.
`src/pineau_rl/`	Le paquet Python des calculs réutilisables (section 8.2).
`tests/`	Les tests unitaires (section 8.7).
`envs/`	Les environnements figés, la plateforme et la chaîne de documents (chapitre D).
`docs/`	Le document de synthèse en deux langues, sa bibliographie, ses figures et `check_numbers.py`.
`article/`	L'article de blog en deux langues, ses figures et `check_article.py`.
`report/`	Le présent rapport technique, ses tableaux générés et ses figures.
`tools/`	`compare_results.py`, qui compare les fichiers JSON valeur par valeur.

des deux intervalles de Wilson (l_a, u_a) et (l_b, u_b) : $d - \sqrt{(\hat{p}_a - l_a)^2 + (u_b - \hat{p}_b)^2}$ et $d + \sqrt{(u_a - \hat{p}_a)^2 + (\hat{p}_b - l_b)^2}$, avec $d = \hat{p}_a - \hat{p}_b$.

- `fisher(ka, na, kb, nb)` : la valeur p bilatérale du test exact de Fisher sur le tableau 2×2.
- `compare(a, b)` : les deux précédentes réunies, `{diff, lo, hi, p}`, en pleine précision.
- `holm(pvalues)` : les valeurs p ajustées par la procédure descendante de Holm (1979), $\tilde{p}_{(i)} = \min\big(1, \max_{j \le i}(m - j + 1)\, p_{(j)}\big)$.
- `cochran_armitage(k, n, x)` : le test de tendance linéaire des proportions (Armitage 1955). Avec $\bar{p} = \sum k_i / \sum n_i$ et $\bar{x} = \sum x_i n_i / \sum n_i$, la statistique vaut $z = \sum x_i (k_i - n_i \bar{p}) / \sqrt{\bar{p}(1-\bar{p}) \sum n_i (x_i - \bar{x})^2}$.
- `cohen_kappa(a, b)` : le kappa de Cohen (1960), $(p_o - p_e)/(1 - p_e)$, None quand $p_e = 1$.
- `probability_of_improvement(x, y)` : $P(X > Y) + \frac{1}{2} P(X = Y)$ sur toutes les paires d'essais, soit la statistique U de Mann-Whitney divisée par $n_x n_y$ (Agarwal et al. 2021).
- `welch_curve(a, b)`, `welch_pooled(a, b)` et `henderson_script(M, S, a, b, n)` : les trois lectures de la statistique de Henderson et al. (2018) étudiées par l'expérience B1. La première applique le test de Welch (1947) à chaque point d'évaluation, puis moyenne les t ou les p. La deuxième empile tous les points des courbes comme des observations indépendantes. La troisième refait le calcul du script `from_same_dist.py` (Henderson 2017) : un test de Welch construit à partir

des moyennes et des écarts types entre épisodes, où la dispersion entre graines aléatoires n'entre jamais.

```
def henderson_script(M, S, a, b, n):
    t, p = _st.ttest_ind_from_stats(M[a].mean(0), S[a].mean(0), n,
                                    M[b].mean(0), S[b].mean(0), n, equal_var=
    False)
    return t.mean(), p.mean()
```

Listing 8.1. Le calcul du script de Henderson, dans `pineau_rl.stats`.

8.2.2 `pineau_rl.bootstrap`

Ce module enveloppe les intervalles bootstrap stratifiés de rliable (Agarwal et al. 2021). `iqm(x)` rend la moyenne interquartile, moyenne de la moitié centrale des scores. `iqm_interval(scores, reps=2000)` rend, pour chaque groupe d'essais, la moyenne interquartile et son intervalle bootstrap par percentile à 95 %. `improvement_interval(x, y, reps=2000, confidence=0.95)` rend la probabilité d'amélioration et son intervalle, au niveau voulu : l'analyse de Bonferroni de l'écart 18 des simulations s'en sert avec $1 - 0{,}05/24$. `probability_of_improvement_bootstrap(x, y, rng)` rééchantillonne x et y avec un générateur NumPy explicite, sans rliable.

Un défaut de rliable 1.2.0 commande l'usage de ce module. La bibliothèque accepte un argument `random_state` mais tire ses rééchantillons dans le générateur global de NumPy (écart 13 des simulations). Un script fixe donc ce générateur une seule fois, au début (`np.random.seed(2026)`), puis appelle les fonctions dans un ordre fixe : chaque appel consomme le générateur global, si bien qu'un intervalle dépend des appels qui le précèdent. Le module importe rliable dès son propre chargement, jamais dans une fonction, pour qu'aucun import ne survienne après la graine aléatoire.

8.2.3 `pineau_rl.runs`

Ce module définit le format d'un essai de simulation (section 8.3) et ses lecteurs. `run_path(root, algo, task, seed)` donne le chemin d'un essai. `read_run(path, final_step=None, upto=None)` lit un fichier et le vérifie : colonnes requises présentes, pas strictement croissants. Elle rend `None` pour un fichier absent ou un essai inachevé. L'argument `upto` coupe la courbe à un budget plus petit, ce qui sert au budget égal. `write_run(path, rows)` écrit le format depuis n'importe quelle bibliothèque. `finals(…)` rend le score de chaque graine aléatoire au pas voulu, `load_grid(…)` la grille complète `{tâche: {algorithme: scores}}` avec la liste des couples manquants.

8.2.4 `pineau_rl.grid`

Ce module lit la grille du sondage. Il définit les listes de critères (`V1` : Q1 à Q8 et X1 à X7 ; `V2` : Y1 à Y12), les critères entiers (`INT` : X1, X6, Y3), les valeurs permises et les types d'incertitude de Y11 (`KINDS`). `load_reads(root, readers)` charge les fichiers de lecture, `load_overrides(path)` l'harmonisation. La valeur finale d'un critère suit l'une de deux sémantiques, gardées à part parce que les scripts diffèrent : `value_v1` (une lecture et son harmonisation) et `value` (la lecture de la grille v2 quand elle a le critère, sinon celle de la grille v1, avec Y3 changé en oui ou non à partir de 2). `tally(values)`

rend le numérateur et le dénominateur d'une part, les na exclus. `validate(rec, …)` contrôle un fichier de lecture (section 8.3).

8.2.5 `pineau_rl.fmt`

Ce module écrit les nombres comme les textes les écrivent. `half_up(x, nd)` arrondit la moitié vers l'extérieur, comme un lecteur ($1{,}15 \to 1{,}2$), alors que la fonction `round` de Python arrondit au pair. Le flottant passe par sa représentation la plus courte, pour que 1,15 reste 1,15. `pct`, `num`, `ci` et `pts` produisent une part en pourcentage, un entier avec séparateur de milliers, un intervalle en points entiers et les bornes signées d'une différence, en style français ou anglais.

8.3 Les formats de fichiers

Un essai de simulation. Un fichier CSV par programme, tâche et graine aléatoire, `sim/results/<algo>/<tâche>/seed<k>.csv`, une ligne par évaluation :

```
step,eval_mean,eval_std,perturbed_mean,wall
50000,329.39900605740297,26.605081154353954,,85.5
100000,690.435139859871,70.99379344477602,,188.4
```

`eval_mean` et `eval_std` donnent la moyenne et l'écart type des retours des dix épisodes d'évaluation, `perturbed_mean` le retour moyen sous physique modifiée (vide quand il n'est pas mesuré), `wall` les secondes écoulées. Un essai est fini quand son dernier pas atteint le budget du programme. Les deux lignes ci-dessus sont les premières de PPO PyTorch sur HalfCheetah, graine aléatoire 0.

Un essai GRPO. Un fichier JSONL par modèle, récompense et graine aléatoire (chapitre 7). Les lignes d'évaluation contiennent `step`, `greedy`, `pass1`, `pass8` et `truncated`. Les lignes d'entraînement, tous les dix pas, `train_right`, `truncated`, `loss`, `grad_norm`, `retries` et `wall`.

Une lecture d'article. Un fichier JSON par article et par lecteur, rangé sous `study/codes_v2/` dans le dossier du lecteur. Pour chaque critère, la valeur, une citation exacte de l'article et sa page, ou la mention « absent après lecture des sections expériences, annexes et liens » avec une page nulle (chapitre A) :

```
{"id": "K001", "coder": "H", "title": "...",
 "criteria": {
  "Q1":  {"value": "yes", "quote": "We evaluate on six MuJoCo tasks", "page":
    6},
  "X1":  {"value": 5, "quote": "averaged over 5 random seeds", "page": 7},
  "Y11": {"value": "yes", "kind": "se",
          "quote": "shaded areas denote one standard error", "page": 7}},
 "doubts": "..."}
```

Le validateur. `pineau_rl.grid.validate` rend la liste des problèmes d'un fichier de lecture, vide quand il est valide. Il vérifie le schéma : chaque critère présent, des valeurs `yes`, `no` ou `na`, des entiers ou `null` pour X1, X6 et Y3, une famille X5 parmi `llm`, `classic` et `theory`, un champ `kind` quand Y11 vaut oui, une citation pour chaque critère. Il

vérifie les règles de portée des écarts 3 et 14 du protocole : Q1 non impose na partout ; Y2 et Y11 valent na si et seulement si X2 vaut non ; Y4 vaut na quand Y3 vaut 1 ; Y5 vaut na si et seulement si Q4 vaut na. Avec une fonction de recherche dans le texte du PDF, il vérifie enfin que chaque citation s'y trouve. La fonction `segments` retire d'abord les notes du lecteur entre crochets, coupe une citation cousue par « […] » en morceaux et ignore les morceaux de moins de douze lettres. Le script `study/check_codes.py` applique ce contrôle avec les PDF. La fiche du lecteur lui demande de le lancer sur ses fichiers jusqu'à zéro problème.

8.4 Chaque nombre vérifié contre son fichier

Deux scripts vérifient les textes publiés. Ils ne lisent aucun nombre écrit à la main : ils le recalculent.

`docs/check_numbers.py`. Pour chaque affirmation chiffrée du document de synthèse, le script recalcule la valeur à partir d'un fichier de résultats, la met en forme comme chaque langue l'écrit (`pineau_rl.fmt`), puis cherche la chaîne obtenue dans le texte, espaces et retours à la ligne normalisés. Quelques comptes du sondage que les scripts d'analyse n'écrivent pas sont recomptés depuis les lectures, avec les mêmes règles. Des assertions protègent les faits qualitatifs, par exemple le nom de la tâche où le programme en tête change sous physique modifiée. Le script vérifie 188 affirmations dans les deux langues et s'arrête au premier nombre manquant, si bien que `docs/build.sh` ne compile pas un document dont un chiffre a dérivé.

```
add("agreement",
    f"{same} décisions sur {total} ({pct(same / total)})",
    f"{same} decisions out of {total} ({pct(same / total, lang='en')})")
```

Listing 8.2. Une affirmation de `docs/check_numbers.py` : la chaîne attendue est construite depuis les données, jamais tapée.

`article/check_article.py`. Le script de l'article de blog ajoute un contrôle de couverture. Chaque affirmation construit, depuis un fichier de résultats, la phrase que les deux langues écrivent, puis la cherche dans le texte. Les quelques nombres sans fichier (une diapositive, un article cité, une taille d'échantillon) sont cherchés dans le document de synthèse lui-même. Le script retire ensuite du texte toutes les phrases vérifiées et les noms propres qui contiennent des chiffres : tout nombre qui reste est un échec. Les lignes de la bibliographie sont confrontées au fichier `docs/refs.bib`. L'article compte 161 affirmations vérifiées dans chaque langue.

8.5 Les cibles du `Makefile`

Aucune cible n'entraîne quoi que ce soit. L'entraînement passe par `sim/run.py`, `sim/drive.sh` et `sim/llm/grpo.py`.

`make install`
: installe le paquet en mode éditable dans l'environnement d'analyse, sans dépendance téléchargée : elles viennent de `envs/analysis.lock`.

`make analysis`
relance tous les scripts d'analyse, dans l'ordre des dépendances : les neuf du sondage, les seize des simulations à dix graines aléatoires, l'analyse statistique du pipeline officiel du Zoo, les variantes de l'écart 12 (vingt graines aléatoires, réplication de E4 sur les graines aléatoires 10 à 19, budget égal), puis `summary.py` à trois et à cinq graines aléatoires. Les variantes passent par des variables d'environnement (`SEEDS`, `SUFFIX`, `BUDGET`).

`make check`
lance les deux scripts de vérification de la section 8.4.

`make test`
lance les tests unitaires avec pytest.

`make lint`
lance ruff et mypy sur le paquet et les tests.

`make verify`
enchaîne `analysis`, `check` et `test`, puis demande à git si chaque fichier JSON de résultats est revenu à l'octet près.

`make verify-head`
fait la même chose dans un export neuf du dernier commit, sans les politiques, les essais du Zoo ni les journaux d'entraînement, qu'aucune analyse ne lit. Cette cible laisse intacte une copie de travail où des essais écrivent encore.

`make figures`
régénère les figures du document de synthèse (`docs/figures/build.py`).

`make docs`
lance `check`, puis construit le document de synthèse (`docs/build.sh`).

`make report`
construit le présent rapport (`report/build.sh`) : tableaux générés depuis les fichiers JSON, figures, puis LuaLaTeX et biber, en français et en anglais.

```
python3.11 -m venv sim/env-an
sim/env-an/bin/pip install -r envs/analysis.txt -r envs/test.txt
make verify          # analyses, nombres des textes, tests, diff octet par
    octet
make verify-head     # la même chose dans un export neuf du dernier commit
```

Listing 8.3. Recalculer tous les nombres depuis les résultats commités, sans entraînement.

8.6 L'intégration continue

Le fichier `.github/workflows/verify.yml` rejoue la vérification à chaque envoi et à chaque demande de fusion, sur deux plateformes. Les actions GitHub employées sont fixées par empreinte de commit, pas par étiquette. Chaque tâche installe l'environnement d'analyse depuis `envs/analysis.lock`, chaque paquet contrôlé par son empreinte SHA-256 (`pip install --require-hashes`), puis installe le paquet, lance les tests, relance toutes les analyses, vérifie les nombres des textes et compare chaque fichier de résultats au commit avec `tools/compare_results.py`.

- Sur Ubuntu 24.04 (x86-64), avec Python 3.11.18, la version employée par l'auteur, la comparaison accepte un écart relatif de 10^{-12} sur les flottants. Elle exige l'égalité des comptes, des verdicts, des classements et de toutes les chaînes. Les valeurs p bougent à leur quinzième ou seizième chiffre avec la bibliothèque mathématique.
- Sur macOS 15 (arm64), la famille de machines dont viennent les résultats, la comparaison exige l'égalité exacte.

Une troisième tâche construit l'image du `Dockerfile` et y lance les tests et la vérification des nombres. L'image part de `python:3.11-slim`, fixée par empreinte, installe le même fichier verrouillé et monte le dépôt au lancement au lieu de le copier : la même image sert à n'importe quel commit.

8.7 Les tests

Le dossier `tests/` contient huit fichiers de tests et un fichier d'outils communs (`conftest.py`). Les tests importent le paquet et la couche mince `sim/analysis/runs.py`, jamais un script d'analyse, qui calcule dès son import. Trois sortes de contrôles s'y côtoient : des valeurs publiées dans la littérature, un calcul indépendant (statsmodels, SciPy, NumPy) et des valeurs de référence sur des données fabriquées, figées à 10^{-12} près en relatif. Une valeur de référence qui change annonce un fichier de résultats qui change.

`test_stats.py`
: Wilson contre une valeur publiée, contre statsmodels et sur ses propriétés (l'intervalle contient la part et reste dans $[0, 1]$, il est symétrique, il s'élargit à 99 %) ; Newcombe contre une valeur publiée et reconstruit depuis les deux intervalles de Wilson ; Fisher contre SciPy ; Holm calculé à la main et ses propriétés ; Cochran-Armitage égal à $\sqrt{N}$ fois la corrélation de Pearson entre score et issue ; le kappa de Cohen et sa liste de catégories ; la probabilité d'amélioration.

`test_bootstrap.py`
: la moyenne interquartile ; les intervalles de rliable, identiques sous une graine aléatoire globale fixée et dépendants de cette seule graine ; la probabilité d'amélioration bootstrap, qui n'emploie que son propre générateur.

`test_henderson.py`
: les trois lectures de la statistique de Henderson contre SciPy ; la valeur p du t moyen avec les degrés de liberté de Welch moyens ; le calcul du script, qui ignore la dispersion entre graines aléatoires : sur six graines fabriquées, il atteint un t de 9 là où un test sur les graines aléatoires ne voit rien.

`test_runs.py`
: le format d'un essai : écriture puis lecture, essais absents ou inachevés rendus `None`, fichiers mal formés qui lèvent une erreur, scores finaux et grille de référence.

`test_grid.py`
: les deux sémantiques de valeur, la conversion de Y3, l'harmonisation, le validateur (fichier valide, chaque problème signalé, règles de portée, citations) et le découpage des citations.

`test_formatting.py`
: l'arrondi de la moitié vers l'extérieur, les pourcentages, les milliers, les intervalles et le signe moins des deux langues.

`test_pipelines.py`
: la lecture des essais par `sim/analysis/runs.py` et le résumé GRPO de bout en bout, sur trois graines aléatoires fabriquées : un essai inachevé est écarté, le gain moyen et l'intervalle bootstrap sont ceux attendus.

`test_draw.py`
: le tirage de 2026 revient de sa règle (`study/draw_2026.py`) et dépend de sa graine aléatoire.

8.8 Les environnements figés

Quatre environnements Python ont fait tourner le projet, tous sous Python 3.11.18. Chaque fichier de `envs/` est le `pip freeze` de l'un d'eux, pris le 15 novembre 2026, avec la version de Python et la plateforme en tête. Les fichiers `*.lock` ajoutent l'empreinte SHA-256 de chaque paquet. Le chapitre D donne les versions. L'environnement d'entraînement (`envs/train.txt`) sert à `run.py` et `dose.py`, l'environnement d'analyse (`analysis.txt`) aux analyses, à la vérification et à l'intégration continue, l'environnement du Zoo (`zoo.txt`) au pipeline officiel de l'expérience E6, l'environnement des modèles de langue (`llm.txt`) à GRPO et aux figures du document de synthèse. Un cinquième fichier, `test.txt`, ajoute pytest à l'environnement d'analyse.

8.9 Les politiques entraînées

Les politiques finales permettent de réévaluer un agent sans le réentraîner, ce qu'a fait la réévaluation par degrés dans des physiques modifiées (`sim/dose.py`). Celles de PPO, de TRPO et de l'expérience E6 sont dans le dépôt. Les poids de SAC et de CrossQ, trop lourds pour git, sont attachés à la publication GitHub `policies-v1`, sous forme d'archives tar listées avec leur empreinte SHA-256 dans un fichier `SHA256SUMS`. Le script `sim/fetch_models.py` télécharge chaque archive, vérifie son empreinte, la décompresse sous `models/`, puis vérifie chaque fichier de politique contre `sim/models/MANIFEST.sha256`, les empreintes prises le 15 novembre 2026 sur la machine qui les a entraînées. Les téléchargements se suivent un par un, séparés par une pause aléatoire de 1,5 à 4 secondes. Une requête refusée attend selon un recul exponentiel à gigue complète, cinq tentatives au plus.

```
cd sim && python3 fetch_models.py          # télécharger, décompresser, tout
    vérifier
cd sim && python3 fetch_models.py --check  # vérifier seulement les politiques
     présentes
```

Listing 8.4. Récupérer et vérifier les politiques de SAC et de CrossQ.

8.10 Le matériel et le calcul

Tous les essais ont tourné sur un seul ordinateur portable, un Mac à puce Apple M2 Max sous macOS 27.2 (arm64), partagé avec d'autres projets. Chaque entraînement de simulation tourne sur un seul fil d'exécution. Le nombre de tâches en parallèle a changé au fil du projet, sans effet sur les résultats puisque chaque essai est déterminé par sa graine aléatoire : quatre cœurs au départ, jusqu'à huit accordés chaque heure selon la charge de la machine (écart 15 des simulations), puis deux cœurs pour tout le projet à partir du 19 novembre 2026 (écart 19). La section 5.11 décrit les lanceurs, le surveillant et la file d'attente. Le tableau 5.3 donne, par programme et par tâche, le nombre d'essais, le nombre de pas, la durée médiane d'un essai et les heures de cœur consommées.

Les essais GRPO ont des durées très différentes selon l'appareil. D'après le champ `wall` de leurs journaux au pas 100, un essai Qwen a pris de 42 à 52 minutes sur GPU et de 4,3 à 11,4 heures sur CPU, un essai Llama de 69 à 148 minutes sur GPU. Ces écarts reflètent aussi la charge variable de la machine.

8.11 Ce qui n'est pas dans le dépôt

Les PDF des articles lus sont protégés par le droit d'auteur. Le script `study/fetch.py` les télécharge de nouveau depuis les sources publiques (`cd study && python3 fetch.py final_2026.json pdf`). Les diapositives de Pineau ne sont pas dans le dépôt pour la même raison : elles se trouvent sur les sites d'ICLR 2018 et de NeurIPS 2018. Le fichier `slides-en.md` les décrit une à une. Les journaux des sessions de travail avec les agents qui ont lu les articles ne sont pas publics. La consigne exacte, le modèle et les dates de chaque lecteur, reconstruits à partir de ces journaux, le sont (`study/readers/`).

8.12 Licence et citation

Le code, les protocoles, les données dérivées et les documents sont publiés sous la licence BSD 3-Clause, celle de scikit-learn (fichier `LICENSE`, Copyright (c) 2026, Warith Harchaoui). Le fichier `CITATION.cff` donne la référence, que GitHub propose en APA et en BibTeX.

CHAPITRE 9

Les écarts aux protocoles

Un protocole écrit d'avance ne protège un résultat que si chaque changement ultérieur se voit. Ce chapitre décrit la règle que nous avons suivie, ce qu'un lecteur peut en vérifier et ce qui repose sur notre parole. Il donne ensuite la liste complète des écarts des trois protocoles et commente ceux qui pèsent le plus sur les résultats.

9.1 Le principe

Les trois protocoles datent du 8 novembre 2026 : celui du sondage (`study/PROTOCOL.md`), écrit avant toute lecture d'article, celui des simulations (`sim/SIM_PROTOCOL.md`) et celui du volet GRPO (`sim/llm/LLM_PROTOCOL.md`), écrits avant tout entraînement. Chacun fixe la question, la population ou les essais, les mesures et les analyses. Tout changement ultérieur s'ajoute en bas du fichier, dans une liste numérotée d'« écarts », jamais en réécrivant le corps du protocole.

Chaque écart donne trois informations : sa date, sa raison et ce que nous avions déjà vu au moment de le décider. La dernière est la plus importante. Une règle fixée avant de voir un résultat ne peut pas avoir été choisie pour lui. Une règle fixée après a pu l'être. Les écarts disent donc « AVANT » ou « APRÈS », en capitales, avec l'objet précis de la comparaison : avant tout entraînement, après les résultats de PPO et de SAC mais avant tout résultat de CrossQ, après l'affichage des valeurs au pas 100 mais avant tout calcul de gain. Trois règles complètent le dispositif. Aucun essai n'est retiré. Un essai qui plante est relancé une fois avec la même graine aléatoire, puis l'incident est noté. Un résultat se publie quel qu'il soit, une fois l'analyse annoncée.

Les trois protocoles comptent 66 écarts : 29 pour le sondage, 20 pour les simulations, 17 pour le volet GRPO. Le tableau 9.1 les résume tous, la figure 9.1 les place dans le temps, du 8 au 21 novembre 2026.

9.2 Ce qui repose sur notre parole

Un lecteur peut recalculer chaque nombre à partir des résultats commités (chapitre 8) et vérifier chaque décision du sondage contre la citation et la page sur lesquelles elle s'appuie. Il ne peut pas vérifier seul l'ordre dans lequel les choses ont été écrites.

Le dépôt est devenu public le 15 novembre 2026. Son premier commit, du 9 novembre à 6 h 07, contient déjà des résultats à côté des protocoles. Pour tout ce qui précède le 15 novembre, l'ordre d'écriture repose sur notre parole et sur les dates des commits, que leur auteur aurait pu forger. Une relecture critique du dépôt public a de plus trouvé plusieurs déclarations « avant »/« après » fausses dans leurs détails. Les corrections sont datées et publiées : écart 14 des simulations, écart 15 du volet GRPO. Elles montrent que la règle a été appliquée avec des erreurs : le lecteur en est averti avant d'accorder sa confiance aux autres déclarations.

Le dépôt fournit trois appuis pour cette confiance. Les journaux d'évaluation de chaque essai sont commités, y compris ceux des essais arrêtés quand ils n'ont pas été écrasés. Le registre des tentatives liste chaque lancement d'entraînement. La provenance des lecteurs du sondage (consigne, modèle, dates, articles écrits) est reconstruite à partir des transcriptions et publiée dans `study/readers/` (écart 28 du sondage).

9.3 Les registres des tentatives

Le protocole des simulations dit qu'un essai qui plante est relancé une fois et que l'incident est noté. Jusqu'au 15 novembre, aucun registre ne gardait la trace de chaque lancement. Le registre reconstruit, `sim/logs/attempts_reconstructed.csv`, s'appuie sur les dates de création des journaux, les sorties des lanceurs, les versions commitées et nos journaux de session. Il donne une ligne par tentative : algorithme, tâche, graine aléatoire, numéro de tentative, début, fin, issue, score vu avant la relance et preuve. L'écart 14 (e) des simulations en tire le bilan : 292 lancements pour les 250 entraînements de la première phase, 42 tentatives arrêtées (2 seulement déclarées), aucune plantée. Chaque arrêt a frappé tous les essais en cours à la fois. Aucun essai n'a été relancé ni écarté après que son score final était connu. Pour PPO PyTorch et PPO JAX, les relances retrouvent à l'identique les points d'évaluation des tentatives arrêtées. Pour SAC et CrossQ, les courbes arrêtées ont été écrasées et ne permettent pas cette vérification.

Depuis le 15 novembre, le lanceur `queue.sh` garde un journal par tentative, jamais écrasé. Il ajoute chaque tentative au registre `sim/logs/attempts.jsonl` : début, fin, code de sortie, empreinte SHA-256 du résultat. Le surveillant y inscrit chaque tâche qu'il arrête. Un arrêt par signal n'est plus relancé automatiquement. Les essais GRPO arrêtés y figurent aussi.

9.4 La liste des écarts

Table 9.1. Les 66 écarts aux trois protocoles, dans l'ordre de leurs listes : sondage (`study/PROTOCOL.md`), expériences (`sim/SIM_PROTOCOL.md`), volet GRPO (`sim/llm/LLM_PROTOCOL.md`). Date : jour de novembre 2026 donné par le protocole (« n. d. » quand l'écart n'est pas daté ; les écarts 1 à 10 du sondage ont été notés pendant le codage, avant l'agrégation). Moment : ce que le protocole déclare avoir déjà vu quand l'écart a été écrit (avant ou après les résultats concernés, ou sans effet sur les résultats pour l'organisation du calcul). Le résumé tient en une ligne ; le texte complet, qui fait foi, est dans le protocole (version anglaise : fichiers `-en.md`). Le nombre des écarts, leur numérotation et leurs dates sont vérifiés contre les fichiers par `report/tools/tables.py`.

N°	Date	Moment	Résumé
Sondage (`study/PROTOCOL.md`, 29 écarts)			
1	n. d.	pendant le codage, avant l'agrégation	X4 harmonisé : `yes` exige un réglage décrit des références ; « même configuration » sans réglage devient `no`.
2	n. d.	pendant le codage, avant l'agrégation	X7 harmonisé : `yes` exige une quantité de calcul (heures, GPU-heures, FLOP) ; le nom du matériel seul vaut `no`.
3	n. d.	pendant le codage, avant l'agrégation	Articles sans expérience (Q1 `no`) : Q2 à Q8 et X1 à X4 en `na`, hors des dénominateurs.
4	n. d.	pendant le codage, avant l'agrégation	Q7 pour un article théorique à expériences jouets : `no`, pas `na`.

suite page suivante

Tableau 9.1, suite

N°	Date	Moment	Résumé
5	n. d.	pendant le codage, avant l'agrégation	X3 : une sélection non dite mais visible dans les tableaux de l'article (LR10) vaut `yes`.
6	n. d.	pendant le codage, avant l'agrégation	Q5 : un code « rendu public plus tard » vaut `no` ; une page de projet vaut `yes` seulement si elle mène au code.
7	n. d.	pendant le codage, avant l'agrégation	Calibration sur 12 articles de 2018 : notre grille plus sévère que Pineau sur Q3, plus indulgente sur Q7 ; un changement doit tenir contre les deux références.
8	n. d.	pendant le codage, avant l'agrégation	Annexes séparées de 2018 lues (ICML, NeurIPS) ; en 2026 elles sont dans le PDF ; C21 n'a pas d'annexe publiée.
9	n. d.	pendant le codage, avant l'agrégation	Adresse de contact retirée de l'en-tête User-Agent des téléchargements ; sans effet sur les données.
10	n. d.	pendant le codage, avant l'agrégation	Scripts de travail écrasés dans l'espace partagé des lecteurs ; les codages, dans des dossiers séparés, sont intacts.
11	9 nov.	avant les résultats	Grille v2 (Y1 à Y12) pour les failles que la v1 ne mesure pas ; Y11 et Y12 ajoutés le même jour, avant tout codage.
12	9 nov.	avant les résultats	Élargissement : 2018 porté à 50 articles (A2), RL classique de 2026 à 50 (A3, tri sur le titre) ; analyse fixée (Newcombe, Fisher, Holm) ; double lecture de 10 articles.
13	9 nov.	avant les résultats	Relecture humaine à l'aveugle de 10 articles prévue (A1), abandonnée à l'écart 23.
14	9 nov.	avant les résultats	Règles de décision de la grille v2 (portée, cas limites de Y1 à Y12) ; fiche `GRILLE.md`, contrôle `check_codes.py`.
15	9 nov.	après les résultats	Règles des écarts 1, 2 et 4 appliquées aussi aux 12 articles de 2018, oubliés jusque-là ; aucune règle nouvelle.
16	9 nov.	avant les résultats	Pilote de trois articles (LR08, ML15, C218) pour repérer les règles ambiguës ; ses lectures n'entrent dans aucune analyse.
17	9 nov.	avant les résultats	Règles précisées après le pilote (annexes de 2018, Y1, Y3, Y5 à Y9, Y11, format) ; 9 articles d'ICLR 2026 injoignables sautés dans la file A3.
18	9 nov.	après la lecture de A3, avant tout pourcentage	X5 (famille) lu sur le modèle que le RL entraîne, pour décider l'entrée des articles de A3 dans l'échantillon classique.
19	9 nov.	après l'audit des lectures, avant tout pourcentage	Harmonisation de la grille v2 : 80 codes changés, chacun motivé dans `overrides_v2.json`, pour les deux années.
20	9 nov.	après les résultats	Règle de verdict : seule la comparaison avec la même grille (RL classique 2026 contre notre lecture de 2018) tranche ; les chiffres de Pineau restent un contexte.
21	10 nov.	après les résultats, avant ce calcul	Contrôles exploratoires (provenance, source du PDF, Mantel-Haenszel, lecteur retiré) et test à l'aveugle de la thèse du coût et du risque.
22	10 nov.	avant les résultats	Listes de contrôle des 30 articles NeurIPS 2025 lus, extraites et confrontées à nos codes (A5).
23	10 nov.	avant les résultats	Relecture humaine (A1) abandonnée, décision de Warith Harchaoui ; dite parmi les limites.
24	10 nov.	avant les résultats	Sensibilité au filtre de titre (A6) : 400 articles hors filtre, résumés lus, puis 20 articles de RL caché lus avec les deux grilles.
25	10 nov.	avant les résultats	Trajectoire de 2018 à 2026 (A4) : 20 articles de RL classique par année d'ICML et NeurIPS 2019, 2020, 2022 et 2024 ; Cochran-Armitage.
26	11 nov.	avant les résultats	Seconde lecture à l'aveugle de 10 articles de A4 (lecteur R3).
27	15 nov.	après les résultats, avant toute recherche	Recherche par mots-clés contre les « non » de la comparaison principale, arbitre à l'aveugle ; décisions retournées, puis douteuses.

suite page suivante

Tableau 9.1, suite

N°	Date	Moment	Résumé
28	15 nov.	après les résultats	Provenance des lecteurs publiée (`readers/`) : deux lecteurs ont fini sur `claude-opus-5` (13 articles) ; les lectures ne changent pas.
29	15 nov.	après les résultats, avant ce calcul	Comparaison principale et trajectoire refaites sans les 13 articles lus en partie par `claude-opus-5`.
Expériences (`sim/SIM_PROTOCOL.md`, 20 écarts)			
1	n. d.	avant le premier entraînement	SAC et CrossQ : 200 000 pas au lieu de 300 000, faute de calcul ; TD3 non lancé.
2	n. d.	après les résultats	E6 : erreur de notre côté (gamma de normalisation 0,99 au lieu de 0,999) ; recette corrigée relancée, diagnostic sur Hopper v4 et v5.
3	9 nov.	après les résultats	E1 : seconde statistique, le t moyen le long de la courbe comme Henderson et al. (`e1_curve.py`).
4	9 nov.	avant tout entraînement TRPO	B1 : 30 entraînements de TRPO sur HalfCheetah et trois lectures de la statistique de Henderson et al.
5	9 nov.	après `e6_diag`, avant tout entraînement officiel	E6 : dix agents entraînés avec le script officiel du Zoo, évalués avec notre protocole.
6	9 nov.	après `b1_trpo`, avant ce calcul	B1 : le script publié par Henderson en 2017 (`from_same_dist.py`) appliqué à nos courbes.
7	10 nov.	après PPO et SAC, avant de regarder CrossQ	Détails d'analyse de E2, E4 et E5 (score final, IQM, probabilité d'amélioration, inversions, tirages).
8	10 nov.	après PPO et SAC, avant de regarder CrossQ	B3 : trois mesures (score final, meilleur score, moyenne de la courbe) et inversions de classement.
9	11 nov.	sans effet sur les résultats	Organisation : le second worker prend la file CrossQ par l'autre bout.
10	12 nov.	sans effet sur les résultats	Arrêt par le surveillant (swap) ; deux entraînements CrossQ relancés depuis zéro.
11	13 nov.	après les résultats	Fin des simulations ; complément descriptif de E7 (classement par IQM dans les deux physiques, sans test).
12	15 nov.	avant tout nouvel entraînement	Consolidation : graines 10 à 19 (160 entraînements), E7 à plusieurs intensités, lecture à budget égal (200 000 pas).
13	15 nov.	après les résultats	Intervalles rendus reproductibles : rliable ignore `random_state`, le générateur global de NumPy est fixé ; verdicts inchangés.
14	15 nov.	après les résultats	Corrections après relecture critique : date du script `dose.py`, comptes des écarts 7 et 8, SB3 2.9.2, pas réels, relances non déclarées, graines lues, registre.
15	15 nov.	sans effet sur les résultats	Jusqu'à huit cœurs accordés au projet selon la charge de la machine.
16	16 nov.	sans effet sur les résultats	Nuit du 15 au 16 novembre : arrêt par le surveillant, worker survivant corrigé, règle de batterie.
17	16 nov.	sans effet sur les résultats	Condition de batterie levée.
18	16 nov.	après les résultats, avant ce calcul	E2 : les 24 intervalles de probabilité d'amélioration rendus simultanés par Bonferroni (99,79 %).
19	19 nov.	sans effet sur les résultats	Plafond de deux cœurs pour tout le projet ; surveillant relancé.
20	21 nov.	après les entraînements, avant tout calcul sur les graines 10 à 19	Calcul des ajouts de l'écart 12 ; défaut de `e5.py` corrigé (bon ordre sur toutes les graines choisies).
GRPO (`sim/llm/LLM_PROTOCOL.md`, 17 écarts)			

suite page suivante

Tableau 9.1, suite

N°	Date	Moment	Résumé
1	n. d.	après lecture de sorties, avant tout entraînement	Limite de longueur portée de 64 à 160 jetons.
2	n. d.	avant tout entraînement	Mesures réduites à 150 problèmes en glouton et 50 en échantillonnage, pour le coût.
3	n. d.	après le pilote, avant tout entraînement	Tâche `mul2` retenue ; mesure principale : pass@1 échantillonné.
4	8 nov.	constat, sans résultat retenu	Le pilote du taux d'apprentissage sature la mémoire (swap) ; arrêté, aucun résultat utilisable.
5	9 nov.	constat, sans résultat retenu	Diagnostic sans entraînement : 23 réponses sur 32 coupées à 160 jetons au pas 10 du pilote.
6	9 nov.	avant tout essai retenu	Version réduite : 100 pas, graines 0 à 2, taux 3e-6, mises à jour par lots de 8 réponses, plafond mémoire.
7	9 nov.	règle fixée avant tout résultat de la graine	Pause à la demande de Warith ; plantage `nan` d'un essai Qwen ; une seule relance prévue.
8	9 nov.	constat, sans résultat retenu	Reprise : aucun des neuf essais restants n'aboutit (`nan`, coupure par macOS, mémoire MPS).
9	9 nov.	avant tout résultat de ces essais	Relance de nuit (`night.sh`) avec un essai de contrôle en tête.
10	10 nov.	constat, sans résultat retenu	Cause des `nan` trouvée : gradients non finis sur MPS, propres sur CPU ; garde-fous de recalcul.
11	10 nov.	avant tout essai retenu avec ce code	Les essais Qwen restants passent sur CPU ; Llama reste sur GPU la nuit.
12	10-11 nov.	corrections avant toute relance	Qwen fini sur CPU ; les six essais Llama manquent de mémoire MPS ; cache vidé à chaque pas, plafond relevé.
13	11-12 nov.	détails de la statistique fixés après les résultats	Les douze essais finis ; bootstrap percentile sur les graines, 10 000 tirages.
14	15 nov.	avant tout nouvel essai	Cinq graines : graines 3 et 4 ajoutées pour les deux modèles et les deux récompenses.
15	15 nov.	après les résultats	Corrections après relecture critique (comptes de l'écart 8, dates) ; aucun résultat ne change.
16	17 nov.	après les valeurs au pas 100, avant tout gain	Llama, récompense au hasard, graine 4 échoue deux fois : compté comme résultat ; statistique sur les essais finis, sensibilité au gain manquant.
17	19 nov.	sans effet sur les résultats	L'essai Llama sur CPU part après la file, sous le plafond de deux cœurs.

9.5 Les écarts du sondage

L'harmonisation (écarts 1 à 6, 15 et 19). Les règles de décision du protocole laissaient des cas limites ouverts. Les écarts 1 à 6, notés pendant la lecture et avant tout pourcentage, fixent les plus fréquents : X4 exige un réglage décrit pour les références, X7 une quantité de calcul, Q5 un lien qui mène à du code. Chaque changement se recode depuis la citation, dans un fichier d'harmonisation (`overrides.json`, `overrides_v2.json`) qui donne la raison de chaque valeur modifiée. L'accord entre lecteurs se calcule toujours sur les lectures brutes, avant harmonisation. L'écart 15 corrige une faute de notre part, vue après les pourcentages : les règles d'harmonisation n'avaient été appliquées qu'aux articles de 2026. Les mêmes règles, sans aucune règle nouvelle, ont été appliquées aux douze articles de 2018 de la calibration. L'écart 19 applique l'harmonisation de la grille

35 des 66 écarts datent des deux premiers jours

Les trois protocoles et leurs écarts, jour par jour, du 8 au 21 novembre 2026

● écart daté dans son texte ○ sans date : noté avant le premier commit, placé au 8

	Sondage 29	Simulations 20	GRPO 17	Jalons
8 dim.	○1 ○2 ○3 ○4 ○5 ○6 ○7 ○8 ○9 ○10	○1 ○2	○1 ○2 ○3 ●4	Les trois protocoles, avant toute lecture et tout entraînement
9 lun.	●11 ●12 ●13 ●14 ●15 ●16 ●17 ●18 ●19 ●20	●3 ●4 ●5 ●6	●5 ●6 ●7 ●8 ●9	Premier commit, 6 h 07
10 mar.	●21 ●22 ●23 ●24 ●25	●7 ●8	●10 ●11	
11 mer.	●26	●9	●12	
12 jeu.		●10	●13	Les 12 essais GRPO à trois graines finis
13 ven.		●11		Les 40 essais CrossQ finis : fin des simulations prévues
14 sam.				
15 dim.	●27 ●28 ●29	●12 ●13 ●14 ●15	●14 ●15	**Dépôt public ; consolidation écrite avant tout essai**
16 lun.		●16 ●17 ●18		
17 mar.			●16	
18 mer.				
19 jeu.		●19	●17	
20 ven.				
21 sam.		●20		**Fin des entraînements, consolidation calculée**

● **Article publié sur deraison.ai**

Sources : écarts de study/PROTOCOL.md, sim/SIM_PROTOCOL.md, sim/llm/LLM_PROTOCOL.md ; historique git ; article de blog sur deraison.ai

Deraison.ai

FIGURE 9.1. Les protocoles et leurs écarts datés, du 8 au 21 novembre 2026. Une ligne par protocole (sondage, simulations, volet GRPO). Chaque marque est un écart, placée à sa date. Sa forme dit s'il a été écrit avant ou après les résultats qu'il concerne.

v2 aux deux années, après un audit des lectures brutes et avant tout pourcentage : 80 changements, chacun avec sa raison. Pour X3, X4 et X7, aucune règle nouvelle n'a été ajoutée, afin de lire les deux années avec la même grille. Les cas limites restent codés comme le lecteur les a lus.

La grille v2 et sa planification (écarts 11, 14, 16 et 17). La grille v1 ne mesurait qu'une partie des failles. La grille v2, Y1 à Y12, a été écrite le 9 novembre 2026 avant toute lecture v2 (écart 11), ses règles de décision le même jour (écart 14), puis précisées après un pilote de trois articles (écarts 16 et 17), toujours avant le codage principal. Le plan de travail qui précède ces écarts, `study/planification_v2.md`, a été gardé tel quel et rendu public le 15 novembre 2026. Il annonce, faille par faille, quels critères v2 sont chargés de sortir chaque verdict de l'état « à mesurer ».

L'élargissement (écart 12). L'audit de puissance du premier tirage montre que douze articles de 2018 et quatorze articles de RL classique de 2026 ne suffisent pas à détecter un écart de trente points. L'écart 12, écrit avant tout tirage et toute lecture des articles ajoutés, élève chacun des deux échantillons à 50 articles et fixe la comparaison principale : le RL classique de 2026 contre notre propre lecture de 2018, avec les mêmes grilles. Les articles de la file de 2026 passent par un tri sur le titre qui écarte les titres présumés de grands modèles de langue. Le chapitre 10 en discute l'effet.

La règle de verdict (écart 20). C'est l'écart du sondage qui pèse le plus sur les conclusions. Il a été décidé après avoir vu les premiers résultats de la lecture élargie. Jusque-là, un critère avait changé quand l'écart tenait contre les chiffres de Pineau. La comparaison secondaire montre que notre grille lit 2018 autrement qu'elle : 55 % contre 90 % pour les hyperparamètres de la méthode (Q3), 33 % contre 10 % pour l'évaluation tenue à part (Q7). Comparer 2026 à ses chiffres mêlerait le changement des pratiques et celui de la lecture. Depuis cet écart, un verdict repose seulement sur la comparaison principale de l'écart 12, fixée avant toute lecture des articles ajoutés. Les comparaisons avec Pineau restent montrées comme contexte. Le changement de règle est déclaré comme décidé après les résultats. La comparaison qu'il retient avait été fixée avant eux.

La relecture humaine abandonnée (écarts 13 et 23). L'écart 13 prévoyait que Warith Harchaoui relise seul dix articles de 2026, à l'aveugle, sur sept critères. L'écart 23 l'abandonne le 10 novembre 2026, avant toute relecture. Aucun humain n'a donc relu les articles. Chaque décision reste vérifiable par sa citation. Le dossier `study/relecture_humaine/` est gardé comme trace du plan.

Les analyses de réponse à une relecture (écarts 21, 22, 24 à 29). Après les résultats, plusieurs analyses ont été ajoutées pour répondre à des objections, chacune écrite avant son propre calcul et publiée quelle qu'elle soit : les coupes de robustesse et la mise à l'épreuve de la thèse du coût (écart 21), les listes de contrôle déclarées par les auteurs (écart 22), le filtre de titre (écart 24), la trajectoire annuelle et sa seconde lecture (écarts 25 et 26), la recherche mécanique contre les décisions « non » (écart 27), la provenance des lecteurs (écart 28) et le retrait des treize articles lus en partie par un autre modèle (écart 29). L'écart 21 a retiré une lecture du document : la thèse « Pineau écoutée là où cela coûtait peu » ne résiste pas au test de corrélation qu'il annonçait (chapitre 4).

9.6 Les écarts des simulations

Le budget (écart 1). SAC et CrossQ s'entraînent 200 000 pas au lieu de 300 000, décidé avant le premier entraînement pour tenir le calcul. PPO garde son million de pas. Cet écart crée la comparaison à budgets inégaux que l'écart 12 (iii) corrige ensuite par une lecture à budget égal.

B3, écrit en cours de route (écart 8). L'expérience B3, qui demande si la mesure (score final, meilleur score, moyenne de la courbe) change le classement, a été écrite le 10 novembre 2026 après les résultats de PPO et de SAC. Elle n'entraîne rien : elle relit les courbes déjà enregistrées. L'écart 8 la datait « AVANT tout résultat de CrossQ ». L'écart 14 (b) corrige ce compte, sept essais CrossQ étant finis à l'écriture. Nos journaux de session, non publics, montrent qu'aucun score CrossQ n'avait été lu. La formulation exacte devient « avant d'avoir regardé le moindre résultat de CrossQ ». L'écart 7, qui fixe les détails de E2, E4 et E5, reçoit la même correction.

Le classement descriptif sous physique modifiée (écart 11). Le renversement le plus cité de l'article de blog, PPO écrit en JAX premier sur Hopper dans la physique d'entraînement et dernier dans la physique modifiée, vient d'un complément décidé après avoir vu les résultats de E7, sans test (`analysis/e7_rank.py`). Le document le présente comme tel. La réévaluation par degrés de l'écart 12 (ii), écrite avant tout nouveau calcul, a ensuite mis ce renversement à l'épreuve : PPO écrit en JAX n'est plus premier sur Hopper dans 10 physiques modifiées sur 12.

La consolidation (écart 12). Écrit le 15 novembre 2026, avant tout nouvel entraînement et toute nouvelle évaluation, cet écart ajoute trois analyses avec le même code et les mêmes réglages : vingt graines aléatoires par couple (160 entraînements), la réévaluation de chaque politique sous des masses et des frottements modifiés par degrés, enfin la lecture de PPO à 200 000 pas, au budget de SAC et de CrossQ. Les résultats à dix graines aléatoires restent les résultats principaux. Le chapitre 6 donne les deux côte à côte et dit chaque divergence.

La graine aléatoire du bootstrap (écart 13). En préparant le dépôt public, six scripts ne redonnaient pas leurs intervalles à l'identique. rliable ignore son argument `random_state` et tire dans le générateur global de NumPy. Chaque script fixe désormais ce générateur au départ ; deux passages donnent des fichiers identiques à l'octet près. Les estimations ponctuelles et les verdicts ne changent pas. Les bornes bougent de l'ordre du bruit de Monte-Carlo, au plus 16 points sur une moyenne interquartile de 2 423 pour E6.

Sept corrections après une relecture critique (écart 14). Une relecture hostile du dépôt public a établi sept points, tous sans effet sur les verdicts. (a) Le commit qui disait le script de dose « écrit avant les résultats de dose » se trompait : 30 réévaluations sur 160 existaient déjà. Le script applique une règle écrite avant tout calcul, sans paramètre libre. (b) Les comptes d'essais CrossQ finis des écarts 7 et 8 étaient faux. (c) Le protocole annonçait Stable-Baselines3 2.6.0. Tous les entraînements ont tourné avec la version 2.9.2, installée avant le premier entraînement. (d) PPO et TRPO collectent par paquets de 2 048 pas, si bien que chaque bloc de 50 000 pas en fait 51 200 : les modèles de PPO et de TRPO comptent 1 024 000 pas, contre 200 000 pour SAC et CrossQ. (e) Les relances non déclarées, décrites à la section 9.3. (f) Six scripts lisaient tous les fichiers présents.

Avec les graines aléatoires 10 à 19, E6 serait passé de dix à vingt graines aléatoires sans le dire. Chaque script nomme désormais ses graines aléatoires. (g) Le registre `attempts.jsonl` et les journaux par tentative, à partir du 15 novembre.

L'organisation du calcul (écarts 9, 10, 15 à 17 et 19). Ces écarts notent des arrêts du surveillant, des relances, le passage à huit cœurs puis à deux, une suspension liée à la batterie du Mac, levée le même matin. Aucun ne change un résultat : chaque essai tourne sur un seul fil et dépend de sa seule graine aléatoire.

Le calcul des ajouts (écart 20). Écrit le 21 novembre 2026, après la fin de tous les entraînements et avant tout calcul sur les graines aléatoires 10 à 19, cet écart corrige un défaut de `analysis/e5.py`, qui prenait pour ordre « correct » la moyenne des dix premiers essais même avec vingt graines aléatoires. Il liste ensuite les calculs, tous dans `make analysis`. Deux d'entre eux, le classement descriptif et la correction de Bonferroni à vingt graines aléatoires, ne figuraient pas dans l'écart 12 : ils appliquent des règles déjà écrites et sont publiés quels qu'ils soient.

9.7 Les écarts du volet GRPO

Le plan réduit puis rétabli (écarts 6 et 14). L'écart 6, décidé avant tout essai retenu, réduit le plan à 100 pas et trois graines aléatoires, fixe le taux d'apprentissage à 3×10^{-6} sans pilote complet et découpe la mise à jour en sous-lots. L'écart 14, écrit le 15 novembre 2026 avant tout nouvel essai, ajoute les graines aléatoires 3 et 4. Ce rétablissement est la décision qui a le plus changé la lecture du volet : le gain de Qwen avec la vraie récompense, net à trois graines aléatoires, ne l'est plus à cinq (chapitre 7).

Les plantages et leur cause (écarts 7 à 11). Les premiers essais retenus ont planté sur des probabilités non finies ou manqué de mémoire GPU. Les écarts 7 et 8 posent une règle de relance, décidée avant de voir un résultat de ces essais. L'écart 9 déplace les essais vers la nuit. L'écart 10 trouve la cause, un calcul GPU interrompu par macOS qui revient avec des valeurs corrompues. Il ajoute les garde-fous de `grpo.py`. Il remet aussi à zéro le compte des échecs, puisque les plantages passés venaient de la machine : ce changement de règle est décidé après avoir vu ces plantages, alors qu'aucun résultat de ces essais n'existait. L'écart 11 fait passer les essais Qwen restants sur CPU. C'est l'origine du mélange entre matériel et graine aléatoire discuté à la section 7.5.3.

Corrections après relecture (écart 15). Écrit le 15 novembre 2026, après les résultats, il corrige la répartition des neuf échecs du 9 novembre, la date d'un essai, fixe la révision exacte des deux modèles dans `grpo.py`, documente les arrêts du 15 novembre et promet la comparaison des seuls essais CPU, puisque l'appareil change entre les graines aléatoires de Qwen.

L'essai qui échoue deux fois (écart 16). Écrit le 17 novembre 2026, après l'affichage, lors d'un point d'avancement, des valeurs au pas 100 des sept autres nouveaux essais, mais avant tout calcul de gain ou d'intervalle à cinq graines aléatoires. L'essai Llama à récompense tirée au sort et graine aléatoire 4 échoue deux fois sur le garde-fou du gradient. Selon l'écart 14, ce second échec est un résultat. La configuration se résume sur ses quatre essais finis, avec une analyse de sensibilité au gain manquant. L'écart prévoit aussi, hors protocole, un essai unique sur CPU, dont le résultat reste à part.

L'ordre de l'essai sur CPU (écart 17). Avec le plafond de deux cœurs de l'écart 19 des simulations, l'essai Llama sur CPU part après tous les autres calculs. Cet ordre ne change aucun résultat.

9.8 Ce que ces écarts disent du travail

La plupart des écarts tiennent à trois causes : le calcul disponible sur un seul portable, nos propres erreurs, des objections de relecture. Les écarts décidés après les résultats sont dits tels. Les trois qui pèsent le plus sur une conclusion sont la règle de verdict du sondage (écart 20), le classement descriptif sous physique modifiée (écart 11 des simulations) et la remise à zéro du compte des échecs GRPO (écart 10). Pour le premier, la comparaison retenue avait été fixée avant les résultats. Le deuxième a été mis à l'épreuve par une analyse écrite d'avance. Le troisième ne touche aucun nombre publié, puisqu'aucun essai planté n'avait de résultat. Un lecteur qui refuse ces trois décisions peut refaire chaque calcul sans elles à partir des fichiers commités.

CHAPITRE 10

Limites et menaces sur la validité

Ce chapitre rassemble les limites du travail, pour qu'aucun lecteur n'ait à les chercher. Il les range selon les quatre familles de menaces de Shadish et al. (2002). La validité interne demande si l'effet mesuré vient bien de la cause supposée. La validité externe demande à quelles populations, tâches et conditions le résultat s'étend. La validité de construit demande si la mesure saisit la notion qu'elle prétend saisir. La validité de la conclusion statistique demande si les tests et les intervalles autorisent la conclusion tirée. Une dernière section dit ce que testerait un nouvel entraînement complet sur une autre machine.

10.1 Validité interne

10.1.1 Les lecteurs du sondage sont des agents de modèle de langue

Tous les articles ont été lus par des agents de modèle de langue, copies d'un même modèle (`claude-opus-5-5`). Deux lecteurs ont fini sur un modèle voisin (`claude-opus-5`) pour treize articles, après l'arrêt d'une réponse par un filtre de contenu du fournisseur (écart 28 du sondage). Aucun humain n'a relu les articles à l'aveugle : la relecture prévue par l'écart 13 a été abandonnée avant de commencer (écart 23). Cinq garde-fous réduisent le risque sans l'annuler.

1. **La citation obligatoire.** Chaque décision « oui » s'appuie sur une citation exacte et sa page. Chacun peut vérifier chaque décision contre le PDF. Le script `study/check_codes.py` vérifie mécaniquement que chaque citation figure dans le texte.
2. **L'audit des citations.** Après les premiers pourcentages, nous avons cherché les citations que la vérification mécanique ne retrouvait pas (écart 15, fichier `study/v1_quotes_audit.md`) : sur 89 citations, aucune n'est inventée, 83 figurent mot pour mot dans l'article, une est une paraphrase fidèle, 5 donnent des valeurs contraires aux règles.
3. **Les doubles lectures à l'aveugle.** Vingt articles du sondage ont été relus par un second agent qui ignorait la première lecture. Sur les dix articles relus du premier tirage, les deux lecteurs s'accordent sur 127 décisions sur 130 (98 %). Deux agents s'accordent sur au moins 80 % des articles pour chaque critère, à une exception près, la mesure de performance (Y1), à 60 % sur le premier tirage, avant que ses règles soient précisées. Dix articles de la trajectoire annuelle, relus de même, concordent à 80 % au moins sur chacun des 23 critères (tableau 4.7).
4. **La recherche contre les « non ».** Un « non » ne s'appuie sur aucune citation. Chaque « non » des critères des verdicts a été confronté au texte de l'article avec des mots-clés fixés d'avance, puis chaque extrait trouvé a été jugé par un arbitre qui ne voyait ni l'article ni l'année (écart 27). Sur 377 décisions « non », 10 passent à « oui » (2 en 2018, 8 en 2026) et aucun verdict ne change. Si l'on retourne aussi

les 24 décisions jugées douteuses, l'incertitude (X2) et la sensibilité (Y9) passent juste au-dessus du seuil ($p = 0{,}07$ et $p = 0{,}054$).

5. **Le retrait du modèle voisin.** Sans les treize articles lus en partie par `claude-opus-5`, les verdicts tiennent, sauf l'incertitude rapportée (X2), qui ne passe plus la correction de Holm ($p = 0{,}07$, écart 29).

Ces garde-fous laissent une menace entière. Les deux lecteurs d'une double lecture sont des copies du même modèle : leur accord mesure la stabilité de ce modèle, non l'accord de deux lecteurs indépendants. Un biais systématique du modèle, partagé par toutes ses copies, échappe aux doubles lectures. La consigne exacte, le modèle et les dates de chaque lecteur sont publiés (`study/readers/`), pour qu'un autre lecteur, humain ou non, refasse la lecture.

10.1.2 Les décisions prises après les résultats

Plusieurs règles ont été fixées après avoir vu des résultats : la règle de verdict du sondage (écart 20), l'harmonisation de la calibration de 2018 (écart 15), le classement descriptif sous physique modifiée (écart 11 des simulations), les détails du bootstrap GRPO (écart 13) et la remise à zéro du compte des échecs GRPO (écart 10). Le chapitre 9 discute chacune. L'ordre d'écriture des écarts antérieurs au 15 novembre 2026 repose sur notre parole et sur les dates des commits (section 9.2).

10.1.3 Les essais des simulations

Le registre reconstruit compte 42 tentatives arrêtées que le protocole ne déclarait pas (section 9.3). Chaque arrêt a frappé tous les essais en cours à la fois, sans lien avec un score. Pour SAC et CrossQ, les courbes des tentatives arrêtées ont été écrasées : nous ne pouvons pas montrer que la relance retrouve la même courbe. Le protocole annonçait Stable-Baselines3 2.6.0. Tous les entraînements ont tourné avec la 2.9.2, la même pour tous les essais. Ces écarts ne favorisent aucun programme. Ils sont dits ici pour que le lecteur les connaisse.

10.1.4 Le matériel et la graine aléatoire du volet GRPO

Pour Qwen, le matériel a changé avec la graine aléatoire : les deux essais à vraie récompense qui donnent le gain ont tourné sur GPU, les trois qui n'en donnent pas sur CPU. Cinq essais ne séparent pas ces deux causes (section 7.5.3). Les trois premiers essais ont tourné avant les garde-fous contre un calcul corrompu. Un gradient corrompu mais fini, écrêté à 1, a pu y passer sans trace. Le calcul MPS n'est pas déterministe d'une tentative à l'autre, si bien que la même graine aléatoire ne garantit pas le même essai sur GPU.

10.2 Validité externe

10.2.1 La population des articles

- **La population a changé.** 27 articles du premier tirage de 2026 sur 50 traitent de grands modèles de langue, une famille absente en 2018. Nos verdicts valent pour le RL classique.
- **Le filtre de titre.** L'expression régulière appliquée aux titres rate les articles de RL dont le titre ne le dit pas, en 2018 comme en 2026. Mesuré pour 2026 avec un plan écrit d'avance (écart 24) : sur 400 articles tirés hors du filtre, 25 ont un résumé qui parle de RL, soit environ 1 037 articles manqués (de 708 à 1 503), 45 % de l'ensemble. C'est un majorant, puisque la règle mécanique compte aussi des faux positifs. Lus avec les deux grilles, 20 de ces articles ressemblent au premier tirage : le code en lien dans 60 % d'entre eux (67 % dans le tirage), une incertitude dans 50 % (47 %). Les 7 articles de RL classique parmi eux sont trop peu nombreux pour éprouver nos verdicts.
- **Le tri sur le titre de la file de 2026.** Un article de RL classique dont le titre contient « reasoning », « preference » ou « math » n'a pas pu entrer dans l'échantillon classique de 2026. Les 9 articles d'ICLR 2026 en tête de la file, injoignables le 9 novembre 2026, ont été sautés.
- **La source des PDF.** Les articles de 2026 ont été lus surtout dans leur version arXiv (45 articles de RL classique sur 50), ceux de 2018 dans les actes des conférences. Une version arXiv est parfois plus longue ou plus récente. Avec 5 articles seulement lus sur le site d'une conférence, cet effet ne se mesure pas.
- **La trajectoire sans ICLR.** Les années 2019 à 2024 ne couvrent qu'ICML et NeurIPS : OpenReview refuse les téléchargements automatisés.
- **Le tirage de Pineau.** Sa méthode de tirage de 2018 n'est pas décrite. Nos comparaisons avec ses chiffres supposent un tirage comparable au nôtre et servent de contexte, jamais de verdict.

10.2.2 Les simulations

Les expériences se limitent au simulateur MuJoCo et à quatre tâches de locomotion (HalfCheetah, Hopper, Walker2d, Swimmer), celles de Henderson et al. (2018). Elles ne disent rien des jeux à images, de la robotique réelle ni du RL hors ligne. E1 et B1 se limitent à une seule tâche (HalfCheetah) et à deux algorithmes (PPO et TRPO). Les quatre programmes sont pris avec leurs hyperparamètres par défaut, sauf dans E6 : un autre réglage pourrait changer les classements. La physique modifiée ne change que deux grandeurs, les masses et le frottement.

10.2.3 Les budgets

PPO s'entraîne 1 024 000 pas, SAC et CrossQ 200 000 (écarts 1 et 14 (d) des simulations). La lecture à budget égal (écart 12 (iii)) compare PPO à 200 000 pas, lu sur sa courbe d'un million de pas : un PPO réglé pour un budget de 200 000 pas pourrait faire mieux. Le budget de SAC et de CrossQ, réduit pour tenir le calcul, les place dans le régime « peu de données ». Un budget plus long pourrait changer leur ordre.

10.2.4 Une seule plateforme

Tous les essais viennent d'un seul Mac à puce Apple M2 Max, partagé avec d'autres projets, sous macOS et pour arm64. MuJoCo, JAX et PyTorch arrondissent différemment d'un processeur à l'autre : la même graine aléatoire donne une autre courbe sur une autre machine (section 10.5).

10.2.5 L'échelle du volet GRPO

Le volet GRPO reste une illustration : deux modèles de 0,5 et 1 milliard de paramètres, une tâche de multiplication, 100 pas, des groupes de 8 réponses, des réponses limitées à 160 jetons, aucun terme KL. Shao et al. (2025) travaillent sur des modèles de 7 milliards de paramètres spécialisés en mathématiques, sur MATH-500, avec bien plus de calcul. Notre volet ne réplique pas leur expérience et ne la réfute pas.

10.3 Validité de construit

- **Notre grille lit 2018 autrement que Pineau.** Notre lecture trouve 55 % d'articles de 2018 qui donnent tous les hyperparamètres de leur méthode, quand Pineau en comptait 90 %. Les critères ont le même intitulé mais des règles de décision implicites différentes. C'est la raison de la règle de verdict à grille égale (écart 20).
- **Un critère satisfait n'est pas une pratique bien faite.** Q5 vaut oui pour un lien vers du code, sans vérifier que ce code tourne. X2 vaut oui pour une barre d'erreur, sans juger si elle mesure la bonne dispersion : la barre peut résumer des épisodes au lieu des graines aléatoires, ce que l'expérience B1 montre dangereux. Q8 vaut oui pour un test appliqué, sans juger s'il est approprié.
- **La mesure de performance (Y1)** est notre critère le plus fragile : 60 % d'accord sur le premier tirage, avant que ses règles soient précisées. Le verdict de la faille F8 se lit avec cette réserve.
- **Les cas limites de X3, X4 et X7** n'ont reçu aucune règle d'harmonisation nouvelle, pour lire les deux années avec la même grille. Un calcul réduit au temps d'inférence ou des graines aléatoires ratées écartées restent codés comme le lecteur les a lus.
- **Ce qu'une liste de contrôle déclare.** Les réponses des auteurs à la liste de NeurIPS ne garantissent pas la pratique qu'elles nomment : les 18 articles qui répondent oui à la question sur la signification statistique n'appliquent aucun test. Le sondage mesure ce que l'article fait, la liste de contrôle ce qu'il déclare.
- **Le score d'un essai de simulation** est le retour moyen de dix épisodes déterministes au dernier pas. L'expérience B3 montre qu'une autre mesure (meilleur score, moyenne de la courbe) change l'ordre de certaines paires.
- **Le pass@1 du volet GRPO** mêle justesse et concision, puisqu'une réponse coupée compte pour fausse (section 7.6).
- **Les notes de coût et de risque** qui ont mis à l'épreuve la thèse du coût viennent de trois agents de modèle de langue, de trois tailles différentes. Leur accord entre eux ne garantit pas qu'un chercheur noterait de même.

10.4 Validité de la conclusion statistique

La taille des échantillons du sondage. Avec 50 articles de chaque côté, un changement de dix points passe encore souvent inaperçu : « pas de changement détectable » ne veut pas dire « aucun changement ». Le premier tirage comptait 14 articles de RL classique, trop peu pour détecter un écart de trente points, d'où l'élargissement de l'écart 12. La trajectoire repose sur 20 articles par année : une part y a une incertitude d'environ vingt points de part et d'autre.

Les comparaisons multiples. Le sondage compare dix-sept critères dans la comparaison principale. La correction de Holm protège le taux d'erreur de la famille, au prix de la puissance : un écart dont l'intervalle de Newcombe évite zéro sans passer Holm, comme Q6 ou Y12, est rapporté mais ne fait pas un verdict. Pour E2, 24 intervalles de probabilité d'amélioration sont aussi donnés au niveau simultané de 95 % par la correction de Bonferroni (écart 18 des simulations, tableau 6.4).

Les verdicts fragiles. La hausse de l'incertitude rapportée (X2) ne passe plus la correction de Holm dans deux contrôles ($p = 0{,}07$ chaque fois). Elle reste le plus fragile des quatre changements. Le code et le calcul passent le seuil dans tous nos contrôles, la sensibilité à un hyperparamètre dans presque tous (tableau 4.3).

Le nombre de graines aléatoires des simulations. Les résultats principaux reposent sur dix graines aléatoires par couple. L'expérience E5 montre ce que ce nombre permet : entre deux programmes dont les moyennes diffèrent de moins de 20 %, cinq graines aléatoires ne retrouvent le bon ordre que 71 fois sur cent en médiane, 62 fois mesuré sur vingt graines aléatoires. Les classements à dix graines sur les tâches où le premier ne bat pas nettement le deuxième ne valent pas plus. Les vingt graines aléatoires de l'écart 12 confirment ou nuancent chaque résultat. Le chapitre 6 dit chaque divergence.

Le nombre de graines aléatoires du volet GRPO. Avec cinq graines aléatoires, le bootstrap n'a que 126 rééchantillons distincts, dix avec trois graines aléatoires. Les intervalles du chapitre 7 sont très grossiers. Un intervalle qui contient zéro dit que nous ne pouvons pas conclure, pas que l'effet est nul.

Les intervalles bootstrap. Les intervalles de rliable dépendent du générateur global de NumPy et de l'ordre des appels. Leurs bornes bougent avec la graine aléatoire du bootstrap, de l'ordre du bruit de Monte-Carlo (écart 13 des simulations) : un verdict dont la borne frôle le seuil reste fragile.

10.5 Ce que testerait un nouvel entraînement sur une autre machine

Un nouvel entraînement complet sur un autre processeur ne redonnerait pas nos courbes. MuJoCo, JAX et PyTorch arrondissent différemment d'un processeur à l'autre, si bien que la même graine aléatoire donne une autre trajectoire dès les premiers pas. Le volet GRPO l'illustre : pour un même modèle et une même graine aléatoire, le pass@1 de départ diffère entre GPU et CPU (section 7.5.3).

Un tel entraînement testerait donc les statistiques, pas chaque courbe. Il dirait si la dispersion entre graines aléatoires, les classements, les gonflements dus à la sélection

et les pertes sous physique modifiée restent du même ordre avec d'autres tirages. Une conclusion qui ne survivrait qu'à nos propres graines aléatoires ne dirait rien des algorithmes. L'intégration continue fait déjà le test équivalent pour les analyses : sur Linux, les comptes, les verdicts et les classements reviennent exactement, les flottants à 10^{-12} près en relatif (section 8.6). Aucun entraînement n'a été refait sur une autre machine.

CHAPITRE 11

Conclusion

11.1 La réponse

> Joelle Pineau a été écoutée sur ce qui se publie à côté du résultat : le code, le calcul, l'effet d'un réglage. Elle l'a été beaucoup moins sur la façon de juger le résultat lui-même. Le test statistique reste rare, les réglages des méthodes de référence restent peu décrits, l'évaluation se fait encore souvent dans le monde de l'entraînement.

Cette réponse vaut pour le RL classique, lu par des agents de modèle de langue sur cinquante articles de chaque côté, puis comparé à notre propre lecture de 2018 avec les mêmes grilles (chapitres 3 et 4). Elle s'appuie aussi sur des expériences refaites avec les algorithmes de 2026 sous des protocoles écrits d'avance (chapitres 5 à 7).

11.2 Les nombres principaux

Quatre pratiques ont changé. Le lien vers le code passe de 31 % à 74 % des articles, la sensibilité à un réglage montrée de 44 % à 80 %, le calcul chiffré de 27 % à 58 %, une incertitude rapportée de 63 % à 88 % (tableaux 4.1 et 4.2). Le code et le calcul résistent à toutes les coupes que nous avons tentées, la sensibilité à presque toutes. L'incertitude est fragile : elle ne passe plus la correction de Holm quand on retire les treize articles lus en partie par un autre modèle ($p = 0{,}07$). Lues sur 20 articles par année entre les deux dates, ces quatre pratiques montent lentement, sans saut juste après la liste de contrôle que NeurIPS impose depuis 2019 (tableau 4.6).

D'autres n'ont pas bougé de façon détectable. Un test statistique passe de 6 % à 12 %, des références décrites avec tous leurs réglages de 35 % à 29 %, une évaluation tenue à part de 33 % à 45 %, la version des bancs d'essai de 23 % à 26 %. Presque tous les articles tracent désormais des barres d'erreur, bien peu appliquent un test.

Ce qu'une case cochée garantit. Les 18 articles de NeurIPS 2025 qui répondent oui à la question de la liste de contrôle sur la signification statistique n'appliquent aucun test. Sur le code, 5 articles sur 22 répondent oui sans lien vers leur code. Sur le calcul, 6 articles sur 21 répondent oui sans chiffrer de temps de calcul.

Les graines aléatoires et la sélection. Sur trente essais de PPO sur HalfCheetah, deux groupes de cinq tirés au hasard ne trompent un test statistique que dans 1,2 % des cas. Triés, ils le trompent dans 31 % des cas. Garder les meilleurs gonfle le score de 33 % (tableaux 6.1 et 6.6). Le t de 9 de 2018 s'explique sans doute par le script publié par Peter Henderson, qui mesure la dispersion des épisodes et ignore celle des graines aléatoires. Appliqué à nos essais de TRPO, il dépasse 9 dans 38 % des tirages au hasard (tableau 6.2).

La portée des classements. Sur nos quatre programmes, le meilleur change avec la tâche. PPO y dispose pourtant de cinq fois plus de pas que SAC et CrossQ : à budget égal et avec vingt graines aléatoires, CrossQ mène sur les quatre tâches (tableau 6.3). Une physique modifiée de 20 % bouleverse le classement sur Hopper : le premier y finit dernier (tableau 6.9).

Le volet des modèles de langue. Avec cinq graines aléatoires, le gain moyen de Qwen avec la vraie récompense vaut $+14$ points, de -3 à $+34$, celui de Llama $+6$ points, de $-0{,}4$ à $+15$: les deux intervalles contiennent zéro. Le gain de Qwen repose sur les deux essais faits sur GPU. Le matériel a changé avec la graine aléatoire. À pile ou face, la réussite tombe à 1 % ou moins dans les neuf essais finis. Une part du gain apparent peut venir de la limite de 160 jetons (tableau 7.1).

11.3 Recommandations

Les recommandations suivantes découlent des mesures de ce rapport. Chacune nomme la faille qu'elle vise et le résultat qui la justifie.

11.3.1 Pour les auteurs

1. **Annoncer le nombre de graines aléatoires et en lancer au moins dix.** Sur 41 articles de RL expérimentaux du premier tirage de 2026, 20 ne disent pas combien d'essais ils lancent. La médiane des autres vaut cinq. Avec cinq graines aléatoires, un écart de moins de 20 % se lit à l'envers près d'une fois sur trois, plus d'une fois sur trois quand vingt graines aléatoires fixent le bon ordre (E5, failles F9 et F10).
2. **Ne jamais trier ni sélectionner les essais.** Rapporter tous les essais, y compris ceux qui échouent. Dire ceux qui ont été exclus (E1, E4, critères X3 et Y6).
3. **Tester sur les graines aléatoires, jamais sur les épisodes.** L'unité indépendante est l'entraînement complet. Un test qui compte les épisodes déclare une différence entre deux groupes du même code (B1, faille F11).
4. **Montrer un intervalle à côté de chaque courbe et dire ce qu'il représente.** Nommer l'écart type, l'erreur type ou l'intervalle de confiance, puis la dispersion qu'il résume (critères X2, Y2, Y11).
5. **Comparer à budget égal, ou dire l'écart de budget.** Un classement à budgets inégaux compare aussi les budgets (E2 à budget égal).
6. **Régler les références avec le même soin que la méthode et le dire.** Donner le budget de réglage de chacune et ses hyperparamètres finaux (E6, critères Q4, X4, Y5).
7. **Évaluer dans des conditions tenues à part.** Graines aléatoires d'environnement jamais vues, dynamiques modifiées ou distribution de test distincte (E7, critères Q7, Y8).
8. **Donner les versions.** Celle de chaque banc d'essai et de chaque dépendance, dans un fichier d'environnement figé (critères Y7, Y10).
9. **Pour un modèle de langue : plusieurs familles de modèles et la part de réponses coupées.** Un résultat obtenu sur une seule famille renseigne sur cette

famille. Un gain de justesse se lit avec la limite de longueur, la part coupée avant et après, enfin la justesse des réponses complètes (chapitre 7, critère X6).

10. **Écrire le protocole avant le premier essai et dater chaque écart.** Dire, pour chaque changement, ce qui avait déjà été vu (chapitre 9).
11. **Recalculer chaque nombre publié depuis ses fichiers.** Un script qui confronte chaque nombre du texte à son fichier de résultats coûte peu et attrape les dérives (section 8.4).

11.3.2 Pour les relecteurs

Les sept questions de l'article de blog s'adressent à tout lecteur. Un relecteur peut les poser à chaque soumission.

1. Combien de graines aléatoires ? Avec moins de cinq essais, ou avec « les meilleurs essais », le résultat dit peu de chose.
2. Quelle incertitude ? Une moyenne sans intervalle ni bande autour de la courbe ne permet pas de juger un écart.
3. Quelle méthode de référence, réglée comment ? Si la méthode a reçu un mois de réglages et la référence ses valeurs par défaut, l'écart mesure le mois de réglages.
4. Quel budget ? Comparer deux méthodes entraînées sur des nombres de pas différents, c'est aussi comparer ces nombres de pas.
5. Évalué où ? Un agent jugé sur les situations de son entraînement a pu les apprendre par cœur.
6. Quel code ? Sans code publié, le lecteur ignore ce que l'article a vraiment fait.
7. Quel calcul ? Un résultat que personne d'autre n'a les moyens de recalculer ne se reproduit pas.

Le relecteur peut ajouter une huitième question, que ce rapport justifie : quelle unité le test compare-t-il ? Un test sur les épisodes ou sur les points d'une courbe empilés ne mesure pas la variabilité entre entraînements.

11.3.3 Pour les conférences

1. **Demander le test, pas seulement les barres.** La question de NeurIPS sur la signification statistique accepte des barres d'erreur « ou une autre information appropriée ». Les 18 articles qui y répondent oui n'appliquent aucun test. Une question qui demande l'unité du test et le nombre d'entraînements indépendants serait plus exigeante.
2. **Vérifier une case cochée sur un échantillon.** Des articles répondent oui sur le code sans lien, sur le calcul sans chiffre. Un contrôle tiré au sort sur une part des articles acceptés mesurerait l'écart entre déclaration et pratique.
3. **Exiger le lien vers le code dans la version finale.** « Le code sera publié » ne vaut pas un lien.

4. **Demander la version de chaque banc d'essai.** Elle n'a pas bougé de façon détectable depuis 2018 (23 % puis 26 %), alors que les trois agents IA qui ont noté le coût de chaque pratique (chapitre 4) la jugent presque gratuite.
5. **Soutenir les défis de reproduction.** Le défi d'ICLR 2018 présenté par Pineau a montré qu'une reproduction par des tiers est faisable : parmi les participants qui ont répondu au sondage, 54,5 % en ont reproduit une partie, 32,7 % l'essentiel. Le refaire par grandes familles de résultats, avec un protocole écrit d'avance, mesurerait ce que la liste de contrôle ne mesure pas.

11.4 Travaux futurs

- **Une relecture humaine à l'aveugle** d'un échantillon d'articles, sur les critères des verdicts, mesurerait l'accord entre un humain et nos lecteurs, ce que les doubles lectures entre copies d'un même modèle ne mesurent pas. Le plan de l'écart 13 du sondage reste prêt.
- **Une seconde famille de lecteurs**, un autre modèle IA, relirait les mêmes articles avec la même consigne, publiée dans `study/readers/`.
- **ICLR dans la trajectoire.** Les années 2019 à 2024 manquent d'ICLR. Une lecture de ses actes, par une voie de téléchargement qu'OpenReview accepte, compléterait la série.
- **Une lecture, avec la même grille, des articles qui appliquent ce qu'ils appellent RL à de grands modèles de langue**, sur un échantillon de taille comparable, dirait si l'évaluation tenue à part, qui y progresse, s'accompagne des autres pratiques.
- **Un nouvel entraînement complet sur une autre plateforme**, Linux sur x86-64, testerait les statistiques de nos expériences avec d'autres tirages (section 10.5).
- **D'autres familles de tâches** que la locomotion MuJoCo : jeux à images, RL hors ligne, robotique réelle, où les conclusions de E2 à E7 restent à éprouver.
- **Le volet GRPO sur un seul appareil**, avec plus de graines aléatoires, une limite de longueur plus large et la justesse des réponses complètes mesurée à chaque évaluation, séparerait l'effet du matériel de celui de la graine aléatoire. Le même protocole sur des modèles de 7 milliards de paramètres rapprocherait le volet de Shao et al. (2025).

Pineau demandait en 2018 d'être aussi critique envers ses bons résultats qu'envers ses mauvais. Ce rapport a tenté de s'y tenir. En vérifiant nos résultats surprenants, nous avons trouvé deux pièges : une erreur de notre code dans E6, dont la correction n'a rien changé, puis la limite de longueur du volet GRPO. Deux graines aléatoires de plus ont défait le gain de Qwen que nous lisions à trois graines aléatoires. Le dépôt garde la trace de chaque correction, datée, pour qui voudra refaire le travail.

ANNEXE A

La grille de lecture complète

Cette annexe reproduit la grille de lecture du sondage, critère par critère, avec la question posée au lecteur et sa règle de décision. Elle réunit la fiche du lecteur (`study/GRILLE.md`) et les règles du protocole (`study/PROTOCOL.md`) : écarts 1 à 6 pour l'harmonisation de la grille v1, écarts 11, 14, 17 et 19 pour la grille v2. En cas de doute, le protocole fait foi.

A.1 Ce que fait le lecteur

Le lecteur lit l'article entier, annexes comprises. Il remplit un fichier JSON par article (section 8.3). Pour chaque critère, il donne la valeur, une citation exacte de l'article, copiée mot pour mot du PDF, puis la page. Si rien ne justifie un `yes`, la citation est « absent après lecture des sections expériences, annexes et liens » et la page vaut `null`. Une citation sur deux pages reçoit une liste de pages. Dans un PDF à deux colonnes, une citation qui chevauche des lignes se découpe en morceaux d'une ligne joints par « [...] ». Le lecteur ne lit jamais les codes d'un autre lecteur, ni les résultats, ni le document de synthèse. Il lance `study/check_codes.py` sur ses fichiers et corrige jusqu'à zéro problème. Les valeurs possibles sont `yes`, `no` et `na` (non applicable), sauf pour les critères entiers et pour la famille X5.

A.2 La grille de Pineau (Q1 à Q8)

Joelle Pineau a présenté ces huit critères à NeurIPS 2018 (diapositives 29 et 30), avec la part des 50 articles de RL de 2018 qu'elle avait lus qui les satisfaisaient. Sa méthode de tirage n'est pas décrite. Ses intitulés sont donnés en anglais, tels qu'elle les a écrits.

TABLE A.1. Les huit critères de Pineau et ses chiffres de 2018.

Code	Intitulé de Pineau	2018
Q1	Paper has experiments	100 %
Q2	Paper uses neural networks	90 %
Q3	All hyperparams for proposed algorithm are provided	90 %
Q4	All hyperparams for baselines are provided	60 %
Q5	Code is linked	55 %
Q6	Method for choosing hyperparams is specified	20 %
Q7	Evaluations on some variation of a hold-out test set	10 %
Q8	Significance testing applied	5 %

TABLE A.2. Grille v1, critères de Pineau : question et règle de décision.

Code	Question	Règle de décision
Q1	L'article a-t-il des expériences ?	no : tous les autres critères valent na et sortent des dénominateurs (écart 3). Les dénominateurs de Pineau sont les articles expérimentaux.
Q2	Emploie-t-il des réseaux de neurones ?	Lecture directe.
Q3	Tous les hyperparamètres de la méthode sont-ils donnés ?	yes si un tableau ou un paragraphe les donne pour toutes les méthodes principales. « Valeurs par défaut de [bibliothèque] » sans les lister compte yes seulement si la version est donnée.
Q4	Tous les hyperparamètres des références sont-ils donnés ?	Même règle que Q3. na s'il n'y a aucune référence comparée.
Q5	Le code est-il en lien ?	Un lien vers du code dans l'article. « Sur demande », « sera publié », « will be made public at <URL> » : no. Une page de projet compte yes seulement si elle mène à du code, vérifié en la visitant le 9 novembre 2026 ; un miroir anonyme qui contient le code compte yes (écarts 6 et 19).
Q6	La façon de choisir les hyperparamètres est-elle décrite ?	La méthode de choix est dite : grille, recherche aléatoire, budget, critère. « Tuned » sans plus : no.
Q7	L'évaluation se fait-elle sur des conditions tenues à part ?	Tâches, environnements, graines aléatoires d'environnement ou jeux de données non vus à l'entraînement ni au réglage. Pour un article llm, des bancs d'essai distincts des données d'entraînement RL comptent yes. Article theory à expériences jouets : no (écart 4). Des graines aléatoires d'entraînement nouvelles, utilisées pour entraîner, ne sont pas des conditions tenues à part (écart 17).
Q8	Un test statistique est-il appliqué à une comparaison ?	Un test (t, Welch, Mann-Whitney, bootstrap avec intervalle, moyenne interquartile de rliable avec intervalles stratifiés) appliqué à une comparaison. Des barres d'erreur seules : no pour Q8, yes pour X2.

A.3 Nos ajouts de la grille v1 (X1 à X7)

Ces critères viennent de la conférence d'ICLR 2018 et du débat de 2025 sur les récompenses trompeuses des modèles de langue (section 7.2). Ils ont été écrits avec le protocole, le 8 novembre 2026, avant toute lecture.

TABLE A.3. Grille v1, nos ajouts : question et règle de décision.

Code	Question	Règle de décision
X1	Combien d'entraînements indépendants pour les résultats principaux ?	Un entier, ou `null` si non dit : le plus petit nombre annoncé pour un tableau ou une courbe principale. Pour un article `llm`, on compte les entraînements RL indépendants et non les échantillons d'évaluation : pass@k et « 16 samples » ne sont pas des entraînements.
X2	Une incertitude est-elle rapportée sur les résultats principaux ?	Écart type, erreur type, intervalle de confiance, bande autour de la courbe.
X3	Les meilleurs essais sont-ils sélectionnés ?	« Top-k », « best of N », meilleur point de contrôle choisi sur le test. `yes` aussi quand les propres tableaux de l'article montrent que le résultat principal est le meilleur de plusieurs entraînements identiques, même si le texte ne le dit pas (écart 5).
X4	L'article dit-il que les références sont réglées avec le même soin que la méthode ?	`yes` exige un réglage décrit pour les références, ou un budget de réglage égal annoncé. « Même configuration que la méthode », sans réglage décrit pour personne : `no` (écart 1).
X5	À quelle famille appartient l'article ?	`llm` si la politique optimisée est un modèle génératif préentraîné conditionné par le langage (modèle de langue de toute taille, modèle de vision-langage, modèle texte-image). `classic` si le RL entraîne autre chose, même au service d'un grand modèle, ou un réseau transformeur entraîné de zéro sur des tâches de RL. `theory` pour un article sans expérience ou à expériences jouets seulement. Un article à résultats théoriques qui compare des méthodes sur des bancs d'essai usuels, même en annexe, est `classic` (écart 18).
X6	Pour `llm` : combien de familles de modèles de base distinctes ?	Un entier, ou `null`. Qwen, Llama, Mistral, Gemma, DeepSeek, OLMo… ; deux tailles de Qwen comptent pour une famille.
X7	Le calcul employé est-il chiffré ?	`yes` exige au moins une quantité : heures, heures de GPU, FLOP, ou un nombre de GPU avec une durée. Le seul nom du matériel : `no` (écart 2).

A.4 La grille v2 (Y1 à Y12)

La grille v2 a été écrite le 9 novembre 2026, avant toute lecture v2 (écart 11), avec ses règles de décision (écart 14), précisées après un pilote de trois articles (écart 17) puis harmonisées après l'audit des lectures brutes, avant tout pourcentage (écart 19). Chaque critère renvoie à la source de 2018 qui le motive.

Règles de portée. Si Q1 vaut no, Y1 à Y12 valent na. Y2 et Y11 valent na si et seulement si X2 vaut no. Y4 vaut na si Y3 vaut 1. Y5 vaut na si et seulement si Q4 vaut na. Pour un article déjà lu avec la grille v1, le lecteur reçoit les valeurs v1 de Q1, Q4 et X2, qui fixent ces règles, sans voir les autres codes v1.

Table A.4. Grille v2 : question, source et règle de décision.

Code	Question	Règle de décision
Y1	La mesure de performance est-elle définie ? (ICLR 2018, diapositive 27)	yes si l'article dit la quantité et le point de mesure : retour final, moyenne sur les N derniers épisodes, meilleur point, aire sous la courbe, pass@k avec k et le nombre d'échantillons, taux de réussite et sur quoi. « Return » ou « accuracy » sans plus : no. Un score normalisé dont on ne dit pas à quel point il est pris : no. Le nombre d'épisodes d'évaluation n'est pas exigé (écart 17). Une courbe dont les points d'évaluation sont dits vaut un point de mesure, « entraîné N pas » puis un seul chiffre vaut le point final. Pour une quantité qui n'est pas le score d'une politique entraînée (erreur d'un estimateur, temps de calcul), yes si la quantité est définie (écart 19).
Y2	Les barres d'erreur sont-elles définies ? (liste de contrôle v1.2)	yes si l'article dit ce que représentent la bande ou l'intervalle : écart type, erreur type, intervalle de confiance à 95 %, minimum et maximum. Des barres présentes mais non définies : no.
Y3	Combien de tâches la conclusion principale couvre-t-elle ? (Henderson et al., tableau 3)	Un entier : le nombre de cadres d'entraînement distincts sur lesquels la méthode est entraînée et comparée, un environnement ou un jeu de données d'entraînement pour `llm`. Deux niveaux de difficulté d'un même jeu, ou les variantes de qualité d'un jeu de données d'un même environnement (D4RL medium, replay, expert), comptent pour une tâche. Plusieurs bancs d'évaluation d'un même modèle entraîné ne multiplient pas les tâches (écart 17). Un seul méta-entraînement compte pour une tâche. Un cadre présent seulement en annexe ne compte pas (écart 19). Valeur dérivée pour l'analyse : yes si l'entier vaut au moins 2.
Y4	Une tâche où la méthode ne gagne pas est-elle montrée ? (Henderson et al., tableau 3)	yes si un résultat tâche par tâche est donné et qu'au moins une tâche où la méthode ne gagne pas est montrée ou dite. no si l'article ne donne que des moyennes, ou ne montre que des tâches où la méthode gagne.
Y5	L'implémentation des références est-elle identifiée ? (Henderson et al., figure 6)	yes si, pour chaque référence principale, l'article dit d'où vient le code : dépôt, bibliothèque ou cadre logiciel nommé, avec ou sans version, « réimplémenté par nous », chiffres recopiés d'articles avec leur source, modèle publié désigné par son nom exact, cadre d'entraînement des auteurs s'il est dit que les références y tournent (écarts 17 et 19). Une seule référence principale sans origine donne no.
Y6	Les exclusions sont-elles dites ? (liste de contrôle v1.2)	yes si l'article dit quels essais, graines aléatoires ou données ont été retirés, ou dit explicitement qu'aucun ne l'a été. Une combinaison omise avec sa raison compte (écart 17), un filtrage des données d'entraînement dit avec ce qu'il retire aussi (écart 19). Le silence vaut no.

Code	Question	Règle de décision
Y7	La version des environnements ou des bancs d'essai est-elle donnée ? (liste de contrôle v1.2)	`yes` si la version de chaque environnement ou banc principal est donnée : suffixe Gym ou Gymnasium, version de la suite, nom exact et date de la séparation de test pour `llm`. « MuJoCo » seul : `no`. « MATH-500 » compte `yes`, « MATH » seul `no`. Un jeu publié sur un dépôt nommé compte comme versionné, une séparation construite par les auteurs seulement si elle est publiée (écart 17). Un nom qui couvre plusieurs ensembles sans dire lequel (« AMC », « Minerva ») ne compte pas. Un environnement entièrement spécifié dans l'article compte (écart 19).
Y8	Un test de généralisation hors des conditions d'entraînement est-il fait ? (NeurIPS 2018, feuille de route)	`yes` si au moins une évaluation se fait sur des conditions absentes de l'entraînement et du réglage : autres graines aléatoires d'environnement annoncées comme telles, tâches nouvelles, dynamiques modifiées, distribution de test distincte. Plus strict que Q7 : pour `llm`, un banc différent ne compte que s'il est présenté comme un test hors distribution. Une séparation de test de même distribution relève de Q7. Y8 `yes` suppose Q7 `yes` (écart 19).
Y9	La sensibilité à au moins un hyperparamètre est-elle montrée ? (ICLR 2018, diapositives 24 à 26)	`yes` si au moins un hyperparamètre de la méthode, présenté comme tel, est balayé et que le résultat est montré par une mesure de performance. Une ablation de composant ne compte pas. Un paramètre réglé à l'inférence compte s'il est présenté comme un hyperparamètre. Une condition de l'environnement ne compte pas (écarts 17 et 19).
Y10	Les dépendances logicielles sont-elles listées avec leurs versions ? (liste de contrôle v1.2)	`yes` si les versions des bibliothèques principales sont dans l'article, ou si un fichier d'environnement est cité dans le dépôt lié. Le dépôt n'est pas visité, sauf si l'article renvoie explicitement au fichier.
Y11	Le type d'incertitude est-il nommé ? (comparable à Gorsane et al., 2022)	`yes` ou `no`, plus un champ `kind` : `std`, `se`, `ci`, `iqr`, `minmax` ou `other`. Plus large que Y2 : il suffit que le type soit nommé quelque part. Des bandes de centiles autres que l'intervalle interquartile : `other` (écart 17).
Y12	Un outil statistique postérieur à 2018 est-il employé ? (adoption d'Agarwal et al., 2021)	`yes` si l'article emploie au moins un de ces outils : moyenne interquartile, intervalle bootstrap stratifié, rliable, profils de performance, probabilité d'amélioration. Un simple bootstrap sur une moyenne compte s'il produit un intervalle rapporté.

A.5 L'harmonisation

Une règle d'harmonisation s'écrit dans le protocole, datée, avant le premier pourcentage qu'elle touche. Elle s'applique aux deux années, depuis les citations, dans `study/overrides.json` (grille v1) ou `study/overrides_v2.json` (grille v2), chaque valeur modifiée avec sa raison. L'accord entre lecteurs se calcule sur les lectures brutes, avant harmonisation. Pour X3, X4 et X7, aucune règle n'a été ajoutée après la lecture élargie, pour que les deux années restent lues avec la même grille (écart 19). La fonction `pineau_rl.grid.validate` contrôle le schéma et les règles de portée de chaque fichier (section 8.3).

ANNEXE B

Tableaux complémentaires

Cette annexe donne les tableaux qui ne figurent pas dans les chapitres, puis un index de tous les tableaux générés et la liste des fichiers de résultats d'où ils viennent. Chaque tableau généré est écrit par `report/tools/tables.py` à partir d'un fichier JSON de résultats : aucun nombre n'y est tapé à la main.

B.1 Le RL de grands modèles de langue dans le sondage

Le premier tirage de 2026 compte 27 articles qui appliquent ce qu'ils appellent RL à de grands modèles de langue (famille X5 `llm`, chapitre A). Les verdicts du sondage valent pour le RL classique. Le tableau 4.5 met ces articles en regard du RL classique de 2026, critère par critère, là où le fichier de résultats le permet. Les 18 articles `llm` lus en chemin dans la file de l'élargissement n'y entrent pas : le tri sur le titre les avait filtrés, si bien qu'ils ne forment pas un échantillon aléatoire (écart 12 du sondage). Le tableau se lit comme une description, sans verdict : il compare deux familles d'articles, pas deux époques.

B.2 Index des tableaux générés

TABLE B.1. Où trouver chaque tableau généré.

Tableau	Contenu	Fichier source
Tableau 4.1	Critères de Pineau, 2018 contre RL classique de 2026	`extension_stats.json`, `results_final.json`
Tableau 4.2	Nos critères Y1 à Y12, mêmes colonnes	`extension_stats.json`
Tableau 4.5	RL de grands modèles contre RL classique, 2026	`extension_stats.json`
Tableau 4.6	Part par pratique et par année, test de tendance	`trajectory.json`
Tableau 4.7	Accord des doubles lectures à l'aveugle	`agree_v2.json`
Tableau 4.3	Contrôles de robustesse du sondage	`robustness.json` et variantes
Tableau 6.1	E1 : trente essais de PPO, faux positifs	`e1.json`
Tableau 6.2	B1 : le t de 9,09, deux façons de le calculer	`b1_script.json`, `b1_trpo.json`
Tableau 6.3	E2 : classement par tâche, dix et vingt graines aléatoires, budget égal	`e2*.json`
Tableau 6.4	E2 avec intervalles de Bonferroni	`e2_bonferroni*.json`
Tableau 6.5	E3 : PPO PyTorch contre PPO JAX	`e3*.json`
Tableau 6.6	E4 : les trois meilleurs sur dix	`e4.json`, `e4_r10.json`

Tableau	Contenu	Fichier source
Tableau 6.7	E5 : combien de graines aléatoires retrouvent l'ordre	`e5*.json`
Tableau 6.8	E6 : la référence réglée	`e6*.json`
Tableau 6.9	E7 : score gardé sous physique modifiée	`e7*.json`
Tableau 6.10	Physique modifiée par degrés	`dose*.json`
Tableau 6.11	B3 : la mesure change-t-elle le classement	`b3*.json`
Tableau 7.1	GRPO : chaque essai	`summary_5.json`, `gains_5.json`, `extra_5.json`
Tableau 9.1	Les écarts des trois protocoles	les trois protocoles
Tableau D.2	Versions des environnements	`envs/*.lock`
Tableau 5.3	Le calcul par programme et par tâche	`sim/logs`, `sim/results`
Tableau C.1	Les articles de 2026 lus	`final_2026.json` et file A3
Tableau C.2	Les articles de 2018 lus	`calib_2018_v2.json`

B.3 Les fichiers de résultats

Chaque fichier JSON de résultats vient d'un seul script, relancé par `make analysis` (section 8.5). Les détails critère par critère et paire par paire que les tableaux résument se trouvent dans ces fichiers.

TABLE B.2. Les scripts d'analyse et ce que contient leur fichier JSON.

Script	Contenu du fichier de résultats
Sondage, dossier `study/`	
`analyze.py`	Parts du premier tirage de 2026 (`results_2026.json`).
`analyze_final.py`	Parts de 2026 et de la calibration de 2018 depuis les lectures brutes et l'harmonisation (`results_final.json`).
`audit_stats.py`	Audit statistique de la comparaison entre 2018 et 2026 : différences de Newcombe, Fisher, Holm, accord des lecteurs (`audit_stats.json`).
`agree_v2.py`	Accord entre lecteurs sur les lectures brutes, avant harmonisation (`agree_v2.json`).
`extension_stats.py`	Comparaison principale : RL classique de 2026 contre notre lecture de 2018, critère par critère, avec Wilson, Newcombe, Fisher et Holm (`extension_stats.json` ; variantes sans le modèle voisin et avec les décisions retournées).
`robustness.py`	Coupes de robustesse et test de la thèse du coût (`robustness.json`).
`title_filter.py`, `title_filter_compare.py`	Le filtre de titre : part de RL caché et comparaison des vingt articles cachés (`title_filter*.json`).
`trajectory.py`	La trajectoire de 2018 à 2026 : parts par année, Cochran-Armitage, écarts immédiat et tardif (`trajectory.json`).
`falsify_no.py`	La recherche contre les décisions « non » (`falsify_no.json`).
`checklist.py`	Les listes de contrôle déclarées à NeurIPS 2025 contre nos lectures (`checklist.json`).
Simulations, dossier `sim/analysis/`	

Script	Contenu du fichier de résultats
`e1.py`, `e1_curve.py`	E1 : trente essais de PPO, test de Welch sur les scores finaux puis sur les courbes.
`b1_trpo.py`, `b1_script.py`	B1 : les lectures du t moyen de 2018 sur TRPO et PPO, puis le calcul du script de Henderson.
`e2.py`, `e2_bonferroni.py`	E2 : moyenne interquartile par tâche, probabilités d'amélioration, changements de classement, puis les mêmes intervalles au niveau simultané de Bonferroni.
`e3.py`	E3 : les deux implémentations de PPO.
`e4.py`	E4 : moyenne et dispersion des dix essais puis des trois meilleurs, renversements.
`e5.py`	E5 : part des tirages qui retrouvent l'ordre, pour 3, 5, 10 et 20 graines aléatoires.
`e6.py`, `e6_diag.py`, `e6_official.py`	E6 : défaut contre Zoo, diagnostic de la recette, politiques officielles évaluées avec notre protocole.
`e7.py`, `e7_rank.py`	E7 : part du score gardée sous physique modifiée, puis le classement descriptif.
`dose.py`	La physique modifiée par degrés.
`b3.py`	B3 : trois mesures par essai et les renversements entre mesures.
Volet GRPO, dossier `sim/llm/`	
`summary.py`	Pass@1 aux pas 0 et 100 de chaque essai, gains, intervalles, sensibilité au gain manquant, essais CPU seuls (`summary*.json`, `gains*.json`, `extra*.json`).

Les suffixes des fichiers des simulations suivent l'écart 12 : `_s20` pour vingt graines aléatoires, `_r10` pour la réplication de E4 sur les graines aléatoires 10 à 19, `_equal` et `_equal_s20` pour le budget égal. Les fichiers du volet GRPO suivent l'écart 14 : sans suffixe pour trois graines aléatoires, `_5` pour cinq.

ANNEXE C

Les articles lus

Cette annexe liste les articles des deux échantillons de la comparaison principale, avec leur identifiant, leur titre et leur conférence. L'identifiant renvoie au fichier de lecture `study/codes/<lecteur>/<id>.json` ou `study/codes_v2/<lecteur>/<id>.json`, où chaque décision s'appuie sur une citation et sa page. Les PDF ne sont pas dans le dépôt (section 8.11). Le script `study/fetch.py` les télécharge depuis les adresses notées dans les fichiers d'échantillon.

C.1 Les articles de 2026

La population compte les 1 252 articles d'ICLR 2026, d'ICML 2026 et de NeurIPS 2025 dont le titre correspond à l'expression régulière du protocole, sur 17 836 articles acceptés (chapitre 3). Le préfixe de l'identifiant dit d'où vient l'article.

- `LR`, `ML`, `NE` : le premier tirage (`random.Random(2018)`), 17 articles d'ICLR 2026, puis ICML 2026 et NeurIPS 2025 (`study/final_2026.json`).
- `MLR`, `NER` : les huit articles de remplacement d'ICML et de NeurIPS, tirés par `random.Random(2026)` dans la même conférence, quand l'article tiré n'avait pas de PDF librement accessible (`replacements_2026.json`).
- `K` : la file de l'élargissement (A3), mélangée par `random.Random(2026)`, lue dans cet ordre après le tri sur le titre jusqu'à 36 articles de RL classique. La file complète se trouve dans `classic_queue_2026.json`.

L'échantillon de RL classique de 2026 de la comparaison principale réunit les 14 articles `classic` du premier tirage et les 36 de la file A3. Le tableau C.1 liste les articles lus.

Table C.1. Les articles de 2026 lus pour le sondage. Premier tirage : 50 articles tirés au hasard parmi les 1 252 dont le titre parle d'apprentissage par renforcement (ICLR 2026, ICML 2026, NeurIPS 2025). File A3 : les autres articles de la population dans l'ordre d'un mélange `random.Random(2026)`, après un tri sur le titre qui écarte les grands modèles de langage (écart 12) ; 70 articles lus, dont les 36 premiers articles de RL classique forment, avec les 14 articles classiques du premier tirage, l'échantillon de 50 de la comparaison principale. Les articles de grands modèles et de théorie lus avant le dernier article retenu restent à part (18 et 3) ; 13 articles lus après lui n'entrent dans aucune analyse. Famille : critère X5 harmonisé (écart 18), lu sur le modèle que le RL entraîne. Identifiants : ceux des fichiers de lecture `study/codes/` et `study/codes_v2/`, où chaque décision cite l'article avec sa page. Les 80 articles de la trajectoire 2019 à 2024 sont listés dans `study/trajectory_queue.json`.

Id.	Titre	Conférence	Famille, rôle
Premier tirage aléatoire (50 articles, `study/final_2026.json`)			
LR01	Optimal Robust Subsidy Policies for Irrational Agent in Principal-Agent MDPs	ICLR 2026	théorie, sans expérience
LR02	Beware Untrusted Simulators – Reward-Free Backdoor Attacks in Reinforcement Learning	ICLR 2026	classique
LR03	MOBODY : Model-Based Off-Dynamics Offline Reinforcement Learning	ICLR 2026	classique

suite page suivante

Tableau C.1, suite

Id.	Titre	Conférence	Famille, rôle
LR04	The Art of Scaling Reinforcement Learning Compute for LLMs	ICLR 2026	grands modèles
LR05	SPIKE-RL : Video-LLMs meet Bayesian Surprise	ICLR 2026	grands modèles
LR06	Near-Optimal Sample Complexity Bounds for Constrained Average-Reward MDPs	ICLR 2026	théorie, sans expérience
LR07	DeepSearch : Overcome the Bottleneck of Reinforcement Learning with Verifiable Rewards via Tree-based Search	ICLR 2026	grands modèles
LR08	TROLL : Trust Regions Improve Reinforcement Learning for Large Language Models	ICLR 2026	grands modèles
LR09	Wavelet Predictive Representations for Non-Stationary Reinforcement Learning	ICLR 2026	classique
LR10	Agnostics : Learning to Synthesize Code in Any Programming Language with a Universal Reinforcement Learning Environment	ICLR 2026	grands modèles
LR11	Predictive CVaR Q-learning	ICLR 2026	théorie
LR12	GAS : Enhancing Reward-Cost Balance of Generative Model-assisted Offline Safe RL	ICLR 2026	classique
LR13	MemAgent : Reshaping Long-Context LLM with Multi-Conv RL-based Memory Agent	ICLR 2026	grands modèles
LR14	TimeSearch-R : Adaptive Temporal Search for Long-Form Video Understanding via Self-Verification Reinforcement Learning	ICLR 2026	grands modèles
LR15	Asynchronous Policy Gradient Aggregation for Efficient Distributed Reinforcement Learning	ICLR 2026	classique
LR16	VeriRole : Verifiable Role-Awareness through Hint-Guided Reinforcement Learning	ICLR 2026	grands modèles
LR17	MedVR : Annotation-Free Medical Visual Reasoning via Agentic Reinforcement Learning	ICLR 2026	grands modèles
ML01	dTRPO : Trajectory Reduction in Policy Optimization of Diffusion Large Language Models	ICML 2026	grands modèles
ML03	CoCoEdit : Content-Consistent Image Editing via Region Regularized Reinforcement Learning	ICML 2026	grands modèles
ML06	MEAL : A Benchmark for Continual Multi-Agent Reinforcement Learning	ICML 2026	classique
ML07	PRPO : Paragraph-level Policy Optimization for Vision-Language Deepfake Detection	ICML 2026	grands modèles
ML08	Towards Practical World Model-based Reinforcement Learning for Vision-Language-Action Models	ICML 2026	grands modèles
ML10	Data- and Variance-dependent Regret Bounds for Online Tabular MDPs	ICML 2026	théorie, sans expérience
ML11	Randomized Advantage Transformation (RAT) : Computing Natural Policy Gradients via Direct Backpropagation	ICML 2026	classique
ML12	VideoVeritas : AI-Generated Video Detection via Perception Pretext Reinforcement Learning	ICML 2026	grands modèles
ML13	GenCircuit-RL : Reinforcement Learning from Hierarchical Verification for Genetic Circuit Design	ICML 2026	grands modèles
ML14	Position : Deployed Reinforcement Learning should be Continual	ICML 2026	théorie
ML15	Moment Matching Q-Learning	ICML 2026	classique
ML16	Causal Flow Q-Learning for Robust Offline Reinforcement Learning	ICML 2026	classique
ML17	InfoPO : Information-Driven Policy Optimization for User-Centric Agents	ICML 2026	grands modèles
NE02	VisionThink : Smart and Efficient Vision Language Model via Reinforcement Learning	NeurIPS 2025	grands modèles
NE04	OpenVLThinker : Complex Vision-Language Reasoning via Iterative SFT-RL Cycles	NeurIPS 2025	grands modèles
NE05	Router-R1 : Teaching LLMs Multi-Round Routing and Aggregation via Reinforcement Learning	NeurIPS 2025	grands modèles
NE06	SATURN : SAT-based Reinforcement Learning to Unleash LLMs Reasoning	NeurIPS 2025	grands modèles
NE07	Finite-Sample Analysis of Policy Evaluation for Robust Average Reward Reinforcement Learning	NeurIPS 2025	théorie, sans expérience
NE08	Pixel Reasoner : Incentivizing Pixel Space Reasoning via Curiosity-Driven Reinforcement Learning	NeurIPS 2025	grands modèles
NE10	Towards Robust Zero-Shot Reinforcement Learning	NeurIPS 2025	classique
NE12	Rainbow Delay Compensation : A Multi-Agent Reinforcement Learning Framework for Mitigating Observation Delays	NeurIPS 2025	classique

suite page suivante

Tableau C.1, suite

Id.	Titre	Conférence	Famille, rôle
NE13	Planning and Learning in Average Risk-aware MDPs	NeurIPS 2025	théorie
NE14	Beyond Scalar Rewards : An Axiomatic Framework for Lexicographic MDPs	NeurIPS 2025	théorie, sans expérience
NE15	Information-Theoretic Reward Decomposition for Generalizable RLHF	NeurIPS 2025	grands modèles
NE16	Sharp Analysis for KL-Regularized Contextual Bandits and RLHF	NeurIPS 2025	théorie
MLR02	REAL : Regression-Aware Reinforcement Learning for LLM-as-a-Judge	ICML 2026	grands modèles
MLR03	Stabilizing Reinforcement Learning for Diffusion Language Models	ICML 2026	grands modèles
MLR06	Estimation of Treatment Effects Under Nonstationarity via the Truncated Policy Gradient Estimator	ICML 2026	classique
MLR07	Text Before Vision : Staged Knowledge Injection Matters for Agentic RLVR in Ultra-High-Resolution Remote Sensing Understanding	ICML 2026	grands modèles
NER01	DAPO : An Open-Source LLM Reinforcement Learning System at Scale	NeurIPS 2025	grands modèles
NER02	Text-to-Decision Agent : Offline Meta-Reinforcement Learning from Natural Language Supervision	NeurIPS 2025	classique
NER04	Asymmetric REINFORCE for off-Policy Reinforcement Learning : Balancing positive and negative rewards	NeurIPS 2025	grands modèles
NER06	FairDICE : Fairness-Driven Offline Multi-Objective Reinforcement Learning	NeurIPS 2025	classique
File A3, articles lus (70 articles, `study/classic_queue_2026.json`)			
K002	MolEditRL : Structure-Preserving Molecular Editing via Discrete Diffusion and Reinforcement Learning	ICLR 2026	échantillon classique
K003	Refined Analysis of Entropy-Regularized Actor-Critic	ICML 2026	à part (théorie)
K004	Spotlight on Token Perception for Multimodal Reinforcement Learning	ICLR 2026	à part (grands modèles)
K006	Joint Selection for Large-Scale Pre-Training Data via Policy Gradient-based Mask Learning	ICLR 2026	échantillon classique
K012	Deep RL Needs Deep Behavior Analysis : Exploring Implicit Planning by Model-Free Agents in Open-Ended Environments	NeurIPS 2025	échantillon classique
K014	Principled RL for Flow Matching Emerges from the Chunk-level Policy Optimization	ICML 2026	à part (grands modèles)
K015	Tracking Drift : Variation-Aware Entropy Scheduling for Non-Stationary Reinforcement Learning	ICML 2026	échantillon classique
K016	Safety Generalization Under Distribution Shift in Safe Reinforcement Learning : A Diabetes Testbed	ICML 2026	échantillon classique
K017	Long-Horizon Model-Based Offline Reinforcement Learning Without Explicit Conservatism	ICML 2026	échantillon classique
K021	Beyond Distributions : Geometric Action Control for Continuous Reinforcement Learning	ICLR 2026	échantillon classique
K023	Sample-Efficient Tabular Self-Play for Offline Robust Reinforcement Learning	NeurIPS 2025	à part (théorie)
K028	PoLi-RL : A Point-to-List Reinforcement Learning Framework for Conditional Semantic Textual Similarity	ICLR 2026	à part (grands modèles)
K030	State Entropy Regularization for Robust Reinforcement Learning	NeurIPS 2025	échantillon classique
K032	Unbiased Dynamic Pruning for Efficient Group-Based Policy Optimization	ICML 2026	à part (grands modèles)
K035	Learning Interactive World Model for Object-Centric Reinforcement Learning	NeurIPS 2025	échantillon classique
K036	Q-Insight : Understanding Image Quality via Visual Reinforcement Learning	NeurIPS 2025	à part (grands modèles)
K037	MARC : Memory-Augmented RL Token Compression for Efficient Video Understanding	ICLR 2026	à part (grands modèles)
K038	1000 Layer Networks for Self-Supervised RL : Scaling Depth Can Enable New Goal-Reaching Capabilities	NeurIPS 2025	échantillon classique
K039	DeepDiver : Adaptive Web-Search Intensity Scaling via Reinforcement Learning	NeurIPS 2025	à part (grands modèles)
K042	Squeezing More from the Stream : Learning Representation Online for Streaming Reinforcement Learning	ICML 2026	échantillon classique

suite page suivante

Tableau C.1, suite

Id.	Titre	Conférence	Famille, rôle
K043	Intrinsic Benefits of Categorical Distributional Loss : Uncertainty-aware Regularized Exploration in Reinforcement Learning	NeurIPS 2025	échantillon classique
K044	From Verifiable Dot to Reward Chain : Harnessing Verifiable Reference-based Rewards for Reinforcement Learning of Open-ended Generation	ICLR 2026	à part (grands modèles)
K045	From Observations to Events : Event-Aware World Models for Reinforcement Learning	ICLR 2026	échantillon classique
K047	Adaptive Surrogate Gradients for Sequential Reinforcement Learning in Spiking Neural Networks	NeurIPS 2025	échantillon classique
K049	Safe Continuous-time Multi-Agent Reinforcement Learning via Epigraph Form	ICLR 2026	échantillon classique
K051	Hierarchical Reinforcement Learning for Sparse-Reward Search in Commutative Algebra	ICML 2026	échantillon classique
K052	RL-SPH : Learning to Achieve Feasible Solutions for Integer Linear Programs	ICML 2026	échantillon classique
K054	When Can You Poison Rewards ? A Tight Characterization of Reward Poisoning in Linear MDPs	ICML 2026	échantillon classique
K055	Towards Large-Scale In-Context Reinforcement Learning by Meta-Training in Randomized Worlds	NeurIPS 2025	échantillon classique
K056	RAMAC : Multimodal Risk-Aware Offline Reinforcement Learning and the Role of Behavior Regularization	ICML 2026	échantillon classique
K058	Adaptive Neighborhood-Constrained Q Learning for Offline Reinforcement Learning	NeurIPS 2025	échantillon classique
K059	ReTabSyn : Realistic Tabular Data Synthesis via Reinforcement Learning	ICML 2026	à part (grands modèles)
K060	ARM-FM : Automated Reward Machines via Foundation Models for Compositional Reinforcement Learning	ICLR 2026	échantillon classique
K061	Enhancing Geometric Perception in VLMs via Translator-Guided Reinforcement Learning	ICLR 2026	à part (grands modèles)
K062	From Absolute to Relative : Rethinking Reward Shaping in Group-Based Reinforcement Learning	ICML 2026	à part (grands modèles)
K063	Benchmarking the Limits of In-Context Reinforcement Learning for Ad-Hoc Teamwork	ICML 2026	échantillon classique
K064	Critique-Coder : Enhancing Coder Models by Critique Reinforcement Learning	ICLR 2026	à part (grands modèles)
K065	Reinforcement Learning for Tool-Calling Agents in Fast Healthcare Interoperability Resources (FHIR)	ICML 2026	à part (grands modèles)
K066	Bridging the performance-gap between target-free and target-based reinforcement learning	ICLR 2026	échantillon classique
K067	Reinforcement Learning Teachers of Test Time Scaling	NeurIPS 2025	à part (grands modèles)
K068	WebSailor-V2 : Bridging the Chasm to Proprietary Agents via Synthetic Data and Scalable Reinforcement Learning	ICLR 2026	à part (grands modèles)
K069	Expanding the Capabilities of Reinforcement Learning via Text Feedback	ICML 2026	à part (grands modèles)
K072	Continuous Q-Score Matching : Diffusion Guided Reinforcement Learning for Continuous-Time Control	NeurIPS 2025	échantillon classique
K073	SMAC : Score-Matched Actor-Critics for Robust Offline-to-Online Transfer	ICML 2026	échantillon classique
K075	CUDA-L1 : Improving CUDA Optimization via Contrastive Reinforcement Learning	ICLR 2026	à part (grands modèles)
K077	Reinforcement Unlearning via Group Relative Policy Optimization	ICLR 2026	à part (grands modèles)
K078	Diffusion Guided Adversarial State Perturbations in Reinforcement Learning	NeurIPS 2025	échantillon classique
K079	From Static Constraints to Dynamic Adaptation : Sample-Level Constraint Relaxation for Offline-to-Online Reinforcement Learning	ICML 2026	échantillon classique
K081	Structured Reinforcement Learning for Combinatorial Decision-Making	NeurIPS 2025	échantillon classique
K082	Retrosynthesis Planning via Worst-path Policy Optimisation in Tree-structured MDPs	NeurIPS 2025	échantillon classique

suite page suivante

Tableau C.1, suite

Id.	Titre	Conférence	Famille, rôle
K083	Use the Online Network If You Can : Towards Fast and Stable Reinforcement Learning	ICLR 2026	échantillon classique
K084	Approximating Shapley Explanations in Reinforcement Learning	NeurIPS 2025	échantillon classique
K085	Scalable Reinforcement Learning via Adaptive Batch Scaling	ICML 2026	échantillon classique
K086	Compute-Optimal Scaling for Value-Based Deep RL	NeurIPS 2025	échantillon classique
K089	Partially Equivariant Reinforcement Learning in Symmetry-Breaking Environments	ICLR 2026	échantillon classique
K090	Performative Policy Gradient : Optimality in Performative Reinforcement Learning	ICML 2026	à part (théorie)
K093	Cross-Embodiment Offline Reinforcement Learning for Heterogeneous Robot Datasets	ICLR 2026	échantillon classique
K095	Robust In-Context Reinforcement Learning Under Reward Poisoning Attacks	ICML 2026	lu après le 36e (classique)
K096	D²Evo : Dual Difficulty-Aware Self-Evolution for Data-Efficient Reinforcement Learning	ICML 2026	lu après le 36e (grands modèles)
K098	Towards Provable Emergence of In-Context Reinforcement Learning	NeurIPS 2025	lu après le 36e (théorie)
K099	On the Sample Complexity of Differentially Private Policy Optimization	NeurIPS 2025	lu après le 36e (théorie)
K102	Pretraining a Shared Q-Network for Data-Efficient Offline Reinforcement Learning	NeurIPS 2025	lu après le 36e (classique)
K103	GUI Exploration Lab : Enhancing Screen Navigation in Agents via Multi-Turn Reinforcement Learning	NeurIPS 2025	lu après le 36e (grands modèles)
K105	The Surprising Difficulty of Search in Model-Based Reinforcement Learning	ICML 2026	lu après le 36e (classique)
K106	Anchored Policy Optimization : Mitigating Exploration Collapse via Support-Constrained Rectification	ICML 2026	lu après le 36e (grands modèles)
K107	SOMBRL : Scalable and Optimistic Model-Based RL	NeurIPS 2025	lu après le 36e (classique)
K109	On the Global Optimality of Policy Gradient Methods in General Utility Reinforcement Learning	NeurIPS 2025	lu après le 36e (théorie)
K110	Kevin : Multi-Turn RL for Generating CUDA Kernels	ICLR 2026	lu après le 36e (grands modèles)
K111	Polychromic Objectives for Reinforcement Learning	ICLR 2026	lu après le 36e (classique)
K113	DAIL : Beyond Task Ambiguity for Language-Conditioned Reinforcement Learning	NeurIPS 2025	lu après le 36e (classique)

C.2 Les articles de 2018

Notre lecture de 2018 compte 50 articles de RL d'ICML 2018, de NeurIPS 2018 et d'ICLR 2018, les trois conférences que Pineau annonçait. Les 12 premiers forment la calibration du protocole, 6 d'ICML et 6 de NeurIPS (`study/calib_2018.json`). Les 38 autres viennent de l'élargissement A2, tirés dans l'ordre d'un mélange `random.Random(2018)` avec une répartition proportionnelle à la taille de chaque conférence (écart 12). L'échantillon réunit 21 articles d'ICML 2018 (préfixe `C1`), 20 de NeurIPS 2018 (`C2`) et 9 d'ICLR 2018 (`C3`) (`study/calib_2018_v2.json`). Les comparaisons de chaque critère comptent les articles où le critère s'applique, si bien qu'un dénominateur peut descendre sous 50.

Table C.2. Les 50 articles de 2018 lus avec les deux grilles. Les 12 premiers forment la calibration de la première version ; les 38 autres (A2, écart 12) sont tirés dans l'ordre d'un mélange `random.Random(2018)` de la population des articles de RL d'ICML, NeurIPS et ICLR 2018 repérés par leur titre (131 articles, `study/rl_population_2018.json`), les trois conférences que Pineau nomme. 48 des 50 articles ont des expériences (Q1) ; les deux autres sortent des dénominateurs. Annexes de 2018 lues à part (ICML, NeurIPS). Source : `study/calib_2018_v2.json`.

Id.	Titre	Conférence	Rôle
C11	RLlib : Abstractions for Distributed Reinforcement Learning	ICML 2018	calibration (12)
C12	Efficient Model-Based Deep Reinforcement Learning with Variational State Tabulation	ICML 2018	calibration (12)
C13	Policy and Value Transfer in Lifelong Reinforcement Learning	ICML 2018	calibration (12)
C14	Fast Bellman Updates for Robust MDPs	ICML 2018	calibration (12)
C15	Can Deep Reinforcement Learning Solve Erdos-Selfridge-Spencer Games ?	ICML 2018	calibration (12)
C16	QMIX : Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning	ICML 2018	calibration (12)
C21	Credit Assignment For Collective Multiagent RL With Global Rewards	NeurIPS 2018	calibration (12)
C22	Learning Temporal Point Processes via Reinforcement Learning	NeurIPS 2018	calibration (12)
C23	Q-learning with Nearest Neighbors	NeurIPS 2018	calibration (12), sans expérience
C24	Evolved Policy Gradients	NeurIPS 2018	calibration (12)
C25	Representation Balancing MDPs for Off-policy Policy Evaluation	NeurIPS 2018	calibration (12)
C26	Reinforcement Learning for Solving the Vehicle Routing Problem	NeurIPS 2018	calibration (12)
C17	Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning	ICML 2018	A2
C27	Scalable Coordinated Exploration in Concurrent Reinforcement Learning	NeurIPS 2018	A2
C28	On Oracle-Efficient PAC RL with Rich Observations	NeurIPS 2018	A2, sans expérience
C29	Multi-Agent Reinforcement Learning via Double Averaging Primal-Dual Optimization	NeurIPS 2018	A2
C210	Meta-Reinforcement Learning of Structured Exploration Strategies	NeurIPS 2018	A2
C18	Addressing Function Approximation Error in Actor-Critic Methods	ICML 2018	A2
C19	Time Limits in Reinforcement Learning	ICML 2018	A2
C110	GEP-PG : Decoupling Exploration and Exploitation in Deep Reinforcement Learning Algorithms	ICML 2018	A2
C211	Bayesian Control of Large MDPs with Unknown Dynamics in Data-Poor Environments	NeurIPS 2018	A2
C111	Fully Decentralized Multi-Agent Reinforcement Learning with Networked Agents	ICML 2018	A2
C112	Modeling Others using Oneself in Multi-Agent Reinforcement Learning	ICML 2018	A2
C212	Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion	NeurIPS 2018	A2
C213	Lifelong Inverse Reinforcement Learning	NeurIPS 2018	A2
C113	Stochastic Variance-Reduced Policy Gradient	ICML 2018	A2
C114	Policy Optimization as Wasserstein Gradient Flows	ICML 2018	A2
C31	N2N learning : Network to Network Compression via Policy Gradient Reinforcement Learning	ICLR 2018	A2
C32	Leveraging Grammar and Reinforcement Learning for Neural Program Synthesis	ICLR 2018	A2
C115	Importance Weighted Transfer of Samples in Reinforcement Learning	ICML 2018	A2
C214	Post : Device Placement with Cross-Entropy Minimization and Proximal Policy Optimization	NeurIPS 2018	A2
C116	Policy Optimization with Demonstrations	ICML 2018	A2
C117	End-to-end Active Object Tracking via Reinforcement Learning	ICML 2018	A2
C33	Go for a Walk and Arrive at the Answer : Reasoning Over Paths in Knowledge Bases using Reinforcement Learning	ICLR 2018	A2
C215	Genetic-Gated Networks for Deep Reinforcement Learning	NeurIPS 2018	A2
C34	Leave no Trace : Learning to Reset for Safe and Autonomous Reinforcement Learning	ICLR 2018	A2
C118	Discovering and Removing Exogenous State Variables and Rewards for Reinforcement Learning	ICML 2018	A2

suite page suivante

Tableau C.2, suite

Id.	Titre	Conférence	Rôle
C119	Efficient Bias-Span-Constrained Exploration-Exploitation in Reinforcement Learning	ICML 2018	A2
C35	Policy Optimization by Genetic Distillation	ICLR 2018	A2
C120	Automatic Goal Generation for Reinforcement Learning Agents	ICML 2018	A2
C216	Visual Reinforcement Learning with Imagined Goals	NeurIPS 2018	A2
C217	Reinforcement Learning of Theorem Proving	NeurIPS 2018	A2
C121	Mix & Match Agent Curricula for Reinforcement Learning	ICML 2018	A2
C36	Temporal Difference Models : Model-Free Deep RL for Model-Based Control	ICLR 2018	A2
C218	Simple random search of static linear policies is competitive for reinforcement learning	NeurIPS 2018	A2
C37	Reinforcement Learning Algorithm Selection	ICLR 2018	A2
C38	Neural Map : Structured Memory for Deep Reinforcement Learning	ICLR 2018	A2
C219	The Importance of Sampling inMeta-Reinforcement Learning	NeurIPS 2018	A2
C39	TreeQN and ATreeC : Differentiable Tree-Structured Models for Deep Reinforcement Learning	ICLR 2018	A2
C220	Verifiable Reinforcement Learning via Policy Extraction	NeurIPS 2018	A2

ANNEXE D

Les environnements logiciels

Cette annexe donne les versions des logiciels qui ont produit les résultats, la plateforme et la chaîne qui construit les documents. Les fichiers de `envs/` font foi. Le tableau D.2 en extrait les paquets clés.

D.1 La plateforme

Tous les essais et toutes les analyses publiées viennent d'un Mac à puce Apple M2 Max, sous macOS 27.2, pour l'architecture arm64, avec Python 3.11.18. Les simulations ont tourné sur le processeur central, un fil d'exécution par essai. Les essais GRPO ont tourné sur le processeur graphique de la puce (interface MPS de PyTorch) ou sur le processeur central, selon l'essai (chapitre 7). Les environnements d'origine viennent de conda (`conda create -p sim/env python=3.11`), puis pip a installé chaque paquet. Un environnement `venv` ordinaire reçoit les mêmes paquets.

D.2 Les quatre environnements

Chaque fichier est le `pip freeze` d'un environnement, pris le 15 novembre 2026, avec la version de Python et la plateforme en tête. Les fichiers `*.lock` donnent les mêmes listes avec l'empreinte SHA-256 de chaque paquet, pour une installation par `pip install --require-hashes`. Les quatre fichiers `*.txt` reprennent les listes que citent les protocoles, `sim/requirements.lock` et ses voisins, réunies en un seul dossier.

TABLE D.1. Les environnements figés du projet (`envs/README.md`).

Fichier	Environnement	Ce qu'il fait tourner
`train.txt`	`sim/env`	les entraînements (`sim/run.py`) et les réévaluations (`sim/dose.py`)
`analysis.txt`	`sim/env-an`	chaque script d'analyse, `docs/check_numbers.py`, l'intégration continue
`zoo.txt`	`sim/env-zoo`	le pipeline officiel de RL Baselines3 Zoo de l'expérience E6
`llm.txt`	`sim/env-llm`	GRPO (`sim/llm/grpo.py`) et les figures de `docs/`
`test.txt`	sur `analysis.txt`	les tests unitaires (`tests/`)

D.3 Les pièges de l'installation

- **Recréer un environnement.** Les chemins comptent, puisque les scripts et les lanceurs appellent `sim/env/bin/python` et ses voisins :

TABLE D.2. Versions épinglées des paquets principaux dans les quatre environnements Python du projet, toutes sur Python 3.11.18, macOS 27.2 sur puce Apple M2 Max (arm64). Chaque fichier `envs/<nom>.txt` est le `pip freeze` de l'environnement, relevé le 15 novembre 2026 ; `envs/<nom>.lock` en est le jumeau verrouillé par empreintes SHA-256 (mêmes versions). `train` : entraînements (`sim/run.py`, `sim/dose.py`) ; `analysis` : tous les scripts d'analyse et l'intégration continue ; `zoo` : le pipeline officiel de RL Baselines3 Zoo (E6) ; `llm` : GRPO (`sim/llm/grpo.py`). Un point : paquet absent de l'environnement. Les tests unitaires ajoutent pytest 9.2.0 (`envs/test.txt`). Le protocole annonçait Stable-Baselines3 2.6.0 ; tous les entraînements ont tourné avec 2.9.2 (écart 14 (c) de `sim/SIM_PROTOCOL.md`). Les révisions des modèles de langue téléchargés ne sont pas épinglées. Source : `envs/*.txt`.

Paquet *environnement*	`train.txt` `sim/env`	`analysis.txt` `sim/env-an`	`zoo.txt` `sim/env-zoo`	`llm.txt` `sim/env-llm`
Python	3.11.18	3.11.18	3.11.18	3.11.18
Stable-Baselines3	2.9.2	·	2.9.2	·
sb3-contrib	2.9.2	·	2.9.2	·
SBX	0.28.0	·	0.28.0	·
RL Baselines3 Zoo	·	·	2.9.2	·
Gymnasium	1.1.1	·	1.1.1	·
MuJoCo	4.0.0	·	4.0.0	·
PyTorch	2.15.0	·	2.15.0	2.15.0
JAX	0.10.2	·	0.10.2	·
jaxlib	0.10.2	·	0.10.2	·
Flax	0.12.8	·	0.12.8	·
Optax	0.2.8	·	0.2.8	·
Transformers	·	·	·	5.22.2
Accelerate	·	·	·	1.15.0
NumPy	2.4.6	2.4.6	2.4.6	2.4.6
SciPy	1.17.1	1.17.1	1.17.1	1.17.1
statsmodels	0.15.0	0.15.0	0.15.0	·
rliable	1.2.0	1.2.0	1.2.0	·
pandas	3.1.1	2.3.3	3.1.1	·
Matplotlib	3.11.2	3.11.2	3.11.2	·
Paquets épinglés	110	25	140	36

```
python3.11 -m venv sim/env-an
sim/env-an/bin/pip install -r envs/analysis.txt -r envs/test.txt
make verify
```

Les trois autres suivent le même modèle. `sim/env-zoo` est un `venv` construit sur le Python de `sim/env`.

- **rliable 1.2.0** n'est distribué que sous forme d'archive source. pip le construit en quelques secondes : c'est du Python pur.
- **Sur Linux**, `analysis.txt` s'installe tel quel. L'intégration continue le fait sur Ubuntu. Les derniers chiffres d'un bootstrap bougent avec le processeur : la comparaison octet par octet est exigée sur macOS arm64, la plateforme des résultats. Linux est comparé à 10^{-12} près en relatif (section 8.6).
- **Entraîner sur Linux** fonctionne avec `train.txt`, mais les essais du dépôt viennent de macOS arm64. Un nouvel essai ailleurs donne d'autres courbes pour la même graine aléatoire : MuJoCo, JAX et PyTorch arrondissent différemment d'un processeur à l'autre (section 10.5).

- **GRPO** télécharge `Qwen/Qwen2.5-0.5B-Instruct` et `unsloth/Llama-3.2-1B-Instruct` depuis Hugging Face. Depuis l'écart 15 (c) du volet GRPO, `grpo.py` fixe la révision exacte de chaque modèle, celle de nos essais. `docs/figures/build.py` télécharge le découpeur en jetons de Qwen.

D.4 La chaîne des documents

Hors des environnements Python, la construction des PDF et des figures emploie les outils du tableau D.3. Ces outils dessinent et composent sans calculer aucun nombre. `make check` et `make verify` n'ont besoin que de `analysis.txt`.

Table D.3. Les outils de construction des documents.

Outil	Version	Employé par
pandoc	3.6.1	`docs/build.sh`
md2star	3.13.5 (PyPI)	`docs/build.sh` (DOCX et un premier PDF)
TeX Live 2026 : LuaHBTeX, latexmk	1.24.0, 4.88	`docs/build.sh`, `report/build.sh`
sprezzature-figures	2.14.1 (2.14.0 sur PyPI)	`docs/figures/build.py`, `article/figures/build.py`
rsvg-convert (librsvg)	2.61.3	conversion des SVG en PDF et en PNG
git	2.50.0	`make verify`

Le présent rapport se construit par `make report`, qui lance `report/build.sh` : les tableaux depuis les fichiers JSON (`report/tools/tables.py`), les figures (`report/tools/figures.py`), puis deux passes de LuaLaTeX avec biber, en français (`rapport.tex`) et en anglais (`report-en.tex`).

Bibliographie

Agarwal, R. et al. (2021). « Deep Reinforcement Learning at the Edge of the Statistical Precipice ». In : *Advances in Neural Information Processing Systems*. T. 34. arXiv : 2108.13264.

Andrychowicz, M. et al. (2021). « What Matters for On-Policy Deep Actor-Critic Methods ? A Large-Scale Study ». In : *International Conference on Learning Representations*. arXiv : 2006.05990.

Armitage, P. (1955). « Tests for Linear Trends in Proportions and Frequencies ». In : *Biometrics* 11.3, p. 375-386. doi : 10.2307/3001775.

Baker, M. (2016). « 1,500 scientists lift the lid on reproducibility ». In : *Nature* 533.7604, p. 452-454. doi : 10.1038/533452a.

Bhatt, A. et al. (2024). « CrossQ : Batch Normalization in Deep Reinforcement Learning for Greater Sample Efficiency and Simplicity ». In : *International Conference on Learning Representations*. arXiv : 1902.05605.

Bjorck, J., C. P. Gomes et K. Q. Weinberger (2022). « Is High Variance Unavoidable in RL ? A Case Study in Continuous Control ». In : *International Conference on Learning Representations*. arXiv : 2110.11222.

Bradbury, J. et al. (2018). *JAX : composable transformations of Python+NumPy programs*. GitHub repository. url : https://github.com/jax-ml/jax.

Buckheit, J. B. et D. L. Donoho (1995). « WaveLab and Reproducible Research ». In : *Wavelets and Statistics*. T. 103. Lecture Notes in Statistics. Springer, p. 55-81. doi : 10.1007/978-1-4612-2544-7_5.

Chandak, N., S. Goel et A. Prabhu (2025). *Incorrect Baseline Evaluations Call into Question Recent LLM-RL Claims*. Blog post, 29 May 2025, https://safe-lip-9a8.notion.site/Incorrect-Baseline-Evaluations-Call-into-Question-Recent-LLM-RL-Claims-2012f1fbf0ee8094ab8ded1953c15a37, relayed on LessWrong.

Christiano, P. F. et al. (2017). « Deep Reinforcement Learning from Human Preferences ». In : *Advances in Neural Information Processing Systems*. T. 30. arXiv : 1706.03741.

Coakley, K. L. et al. (2026). « The Shift Toward Open and Reproducible AI Research ». In : *Communications in AI & Computing* 1, p. 24. doi : 10.1038/s44488-026-00015-x. arXiv : 2606.16974.

Cochran, W. G. (1954). « Some Methods for Strengthening the Common χ^2 Tests ». In : *Biometrics* 10.4, p. 417-451. doi : 10.2307/3001616.

Cohen, J. (1960). « A Coefficient of Agreement for Nominal Scales ». In : *Educational and Psychological Measurement* 20.1, p. 37-46. doi : 10.1177/001316446002000104.

Colas, C., O. Sigaud et P.-Y. Oudeyer (2018). *How Many Random Seeds ? Statistical Power Analysis in Deep Reinforcement Learning Experiments*. arXiv : 1806.08295.

DeepSeek-AI (2025). « DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning ». In : *Nature* 645, p. 633-638. doi : 10.1038/s41586-025-09422-z.

Dunn, O. J. (1961). « Multiple Comparisons Among Means ». In : *Journal of the American Statistical Association* 56.293, p. 52-64. doi : 10.1080/01621459.1961.10482090.

Efron, B. (1979). « Bootstrap Methods : Another Look at the Jackknife ». In : *The Annals of Statistics* 7.1, p. 1-26. doi : 10.1214/aos/1176344552.

Eimer, T., M. Lindauer et R. Raileanu (2023). « Hyperparameters in Reinforcement Learning and How To Tune Them ». In : *Proceedings of the 40th International Conference on Machine Learning*. T. 202. PMLR, p. 9104-9149. arXiv : 2306.01324.

Engstrom, L. et al. (2020). « Implementation Matters in Deep Policy Gradients : A Case Study on PPO and TRPO ». In : *International Conference on Learning Representations*. arXiv : 2005.12729.

Fisher, R. A. (1922). « On the Interpretation of χ^2 from Contingency Tables, and the Calculation of P ». In : *Journal of the Royal Statistical Society* 85.1, p. 87-94. doi : 10.2307/2340521.

Gorsane, R. et al. (2022). « Towards a Standardised Performance Evaluation Protocol for Cooperative MARL ». In : *Advances in Neural Information Processing Systems*. T. 35. arXiv : 2209.10485.

Grattafiori, A. et al. (2024). *The Llama 3 Herd of Models*. arXiv : 2407.21783.

Gundersen, O. E. et S. Kjensmo (2018). « State of the Art : Reproducibility in Artificial Intelligence ». In : *Proceedings of the AAAI Conference on Artificial Intelligence*. T. 32. 1, p. 1644-1651. doi : 10.1609/aaai.v32i1.11503.

Haarnoja, T. et al. (2018). « Soft Actor-Critic : Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor ». In : *Proceedings of the 35th International Conference on Machine Learning*. T. 80. PMLR, p. 1861-1870.

Henderson, P. (2017). *from_same_dist.py, in the repository ReproducibilityInContinuousPolicyGradientMethods*. GitHub, commit of 17 June 2017. Companion code of Islam, Henderson et al. (2017). Accessed 9 November 2026. url : https://github.com/Breakend/ReproducibilityInContinuousPolicyGradientMethods/blob/master/from_same_dist.py.

Henderson, P. et al. (2018). « Deep Reinforcement Learning That Matters ». In : *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, p. 3207-3214. doi : 10.1609/aaai.v32i1.11694. arXiv : 1709.06560.

Hochlehnert, A. et al. (2025). « A Sober Look at Progress in Language Model Reasoning : Pitfalls and Paths to Reproducibility ». In : *Conference on Language Modeling (COLM)*. arXiv : 2504.07086.

Holm, S. (1979). « A Simple Sequentially Rejective Multiple Test Procedure ». In : *Scandinavian Journal of Statistics* 6.2, p. 65-70.

Irpan, A. (fév. 2018). *Deep Reinforcement Learning Doesn't Work Yet*. Blog post, 14 February 2018, https://www.alexirpan.com/2018/02/14/rl-hard.html.

Islam, R. et al. (2017). *Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control*. Reproducibility in Machine Learning Workshop, ICML. arXiv : 1708.04133.

Jordan, S. M. et al. (2020). « Evaluating the Performance of Reinforcement Learning Algorithms ». In : *Proceedings of the 37th International Conference on Machine Learning*. T. 119. PMLR, p. 4962-4973. arXiv : 2006.16958.

Jordan, S. M. et al. (2024). « Position : Benchmarking is Limited in Reinforcement Learning Research ». In : *Proceedings of the 41st International Conference on Machine Learning*. T. 235. PMLR, p. 22551-22569. arXiv : 2406.16241.

Kingma, D. P. et J. Ba (2015). « Adam : A Method for Stochastic Optimization ». In : *International Conference on Learning Representations (ICLR)*. arXiv : 1412.6980.

Lynnerup, N. A. et al. (2020). « A Survey on Reproducibility by Evaluating Deep Reinforcement Learning Algorithms on Real-World Robots ». In : *Proceedings of the Conference on Robot Learning*. T. 100. PMLR, p. 466-489. arXiv : 1909.03772.

Machado, M. C. et al. (2018). « Revisiting the Arcade Learning Environment : Evaluation Protocols and Open Problems for General Agents ». In : *Journal of Artificial Intelligence Research* 61, p. 523-562. arXiv : 1709.06009.

Magnusson, I., N. A. Smith et J. Dodge (2023). *Reproducibility in NLP : What Have We Learned from the Checklist ?* arXiv : 2306.09562.

Mann, H. B. et D. R. Whitney (1947). « On a Test of Whether One of Two Random Variables Is Stochastically Larger than the Other ». In : *The Annals of Mathematical Statistics* 18.1, p. 50-60. doi : 10.1214/aoms/1177730491.

Mantel, N. et W. Haenszel (1959). « Statistical Aspects of the Analysis of Data From Retrospective Studies of Disease ». In : *Journal of the National Cancer Institute* 22.4, p. 719-748. doi : 10.1093/jnci/22.4.719.

Meta AI (sept. 2024). *Llama 3.2 : Revolutionizing Edge AI and Vision with Open, Customizable Models*. https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/. 25 September 2024.

Newcombe, R. G. (1998). « Interval Estimation for the Difference Between Independent Proportions : Comparison of Eleven Methods ». In : *Statistics in Medicine* 17.8, p. 873-890. doi : 10.1002/(SICI)1097-0258(19980430)17:8<873::AID-SIM779>3.0.CO;2-I.

Ouyang, L. et al. (2022). « Training language models to follow instructions with human feedback ». In : *Advances in Neural Information Processing Systems*. arXiv : 2203.02155.

Paszke, A. et al. (2019). « PyTorch : An Imperative Style, High-Performance Deep Learning Library ». In : *Advances in Neural Information Processing Systems*. T. 32. arXiv : 1912.01703.

Patterson, A. et al. (2024). « Empirical Design in Reinforcement Learning ». In : *Journal of Machine Learning Research* 25.318, p. 1-63. arXiv : 2304.01315.

Pineau, J. (mai 2018a). *Reproducibility, Reusability, and Robustness in Deep Reinforcement Learning*. Invited talk, Sixth International Conference on Learning Representations (ICLR), Vancouver. url : https://iclr.cc/virtual/2018/invited-talk/346.

Pineau, J. (déc. 2018b). *Reproducible, Reusable, and Robust Reinforcement Learning*. Invited talk (Posner Lecture), Thirty-second Conference on Neural Information Processing Systems (NeurIPS), Montréal, 5 December 2018. url : https://neurips.cc/virtual/2018/invited-talk/12486.

Pineau, J. et al. (2021). « Improving Reproducibility in Machine Learning Research (A Report from the NeurIPS 2019 Reproducibility Program) ». In : *Journal of Machine Learning Research* 22.164, p. 1-20.

Puterman, M. L. (1994). *Markov Decision Processes : Discrete Stochastic Dynamic Programming*. New York : Wiley. doi : 10.1002/9780470316887.

Qwen Team (2025). *Qwen2.5 Technical Report*. arXiv : 2412.15115.

Raffin, A. (2020). *RL Baselines3 Zoo*. https://github.com/DLR-RM/rl-baselines3-zoo.

Raffin, A. (2023). *SBX : Stable Baselines Jax*. GitHub repository. url : https://github.com/araffin/sbx.

Raffin, A. et al. (2021). « Stable-Baselines3 : Reliable Reinforcement Learning Implementations ». In : *Journal of Machine Learning Research* 22.268, p. 1-8.

Schulman, J. et al. (2015). « Trust Region Policy Optimization ». In : *Proceedings of the 32nd International Conference on Machine Learning*. T. 37. PMLR, p. 1889-1897. arXiv : 1502.05477.

Schulman, J. et al. (2017). *Proximal Policy Optimization Algorithms*. arXiv : 1707.06347.

Shadish, W. R., T. D. Cook et D. T. Campbell (2002). *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*. Boston : Houghton Mifflin.

Shao, R. et al. (2025). *Spurious Rewards : Rethinking Training Signals in RLVR*. arXiv : 2506.10947.

Shao, Z. et al. (2024). *DeepSeekMath : Pushing the Limits of Mathematical Reasoning in Open Language Models*. arXiv : 2402.03300.

Shapiro, S. S. et M. B. Wilk (1965). « An Analysis of Variance Test for Normality (Complete Samples) ». In : *Biometrika* 52.3–4, p. 591-611. doi : 10.1093/biomet/52.3-4.591.
Silver, D. et al. (2016). « Mastering the game of Go with deep neural networks and tree search ». In : *Nature* 529.7587, p. 484-489. doi : 10.1038/nature16961.
Student (1908). « The Probable Error of a Mean ». In : *Biometrika* 6.1, p. 1-25. doi : 10.2307/2331554.
Sutton, R. S. et A. G. Barto (2018). *Reinforcement Learning : An Introduction*. 2e éd. Cambridge, MA : MIT Press.
Tesauro, G. (1995). « Temporal Difference Learning and TD-Gammon ». In : *Communications of the ACM* 38.3, p. 58-68. doi : 10.1145/203330.203343.
Todorov, E., T. Erez et Y. Tassa (2012). « MuJoCo : A physics engine for model-based control ». In : *IEEE/RSJ International Conference on Intelligent Robots and Systems*, p. 5026-5033. doi : 10.1109/IROS.2012.6386109.
Towers, M. et al. (2024). *Gymnasium : A Standard Interface for Reinforcement Learning Environments*. Accepted at NeurIPS Datasets and Benchmarks 2025. arXiv : 2407.17032.
Welch, B. L. (1947). « The Generalization of 'Student's' Problem when Several Different Population Variances are Involved ». In : *Biometrika* 34.1/2, p. 28-35. doi : 10.2307/2332510.
Wilson, E. B. (1927). « Probable Inference, the Law of Succession, and Statistical Inference ». In : *Journal of the American Statistical Association* 22.158, p. 209-212. doi : 10.1080/01621459.1927.10502953.
Yue, Y. et al. (2025). « Does Reinforcement Learning Really Incentivize Reasoning Capacity in LLMs Beyond the Base Model ? » In : *Advances in Neural Information Processing Systems*. arXiv : 2504.13837.
Zhang, A., N. Ballas et J. Pineau (2018a). *A Dissection of Overfitting and Generalization in Continuous Reinforcement Learning*. arXiv : 1806.07937.
Zhang, A., Y. Wu et J. Pineau (2018b). *Natural Environment Benchmarks for Reinforcement Learning*. arXiv : 1811.06032.