Technical report

Reinforcement Learning: Did We Listen to Joelle Pineau in 2018?

A survey of the papers from 2018 to 2026 and preregistered experiments, every number recomputed with one command

Warith Harchaoui
Deraison.ai · warithmetics@deraison.ai

21 November 2026

Code, data and protocols : `github.com/warith-harchaoui/pineau-rl`
Blog article : `deraison.ai/en/blog/listen-to-joelle-pineau`
BSD 3-Clause licence for the code

Deraison.ai

Abstract

The question. In 2018, at ICLR and then at NeurIPS, Joelle Pineau showed why the published results of deep reinforcement learning (RL) inspired little trust. We record sixteen flaws in her two talks and ask, for each one, whether the field listened.

The method. We use two instruments, each under a protocol written in advance whose every deviation is dated. The survey rereads papers with Pineau's reading grid, her criteria Q1 to Q8, seven more drawn from her ICLR talk (X1 to X7) and a second grid of our own (Y1 to Y12). Language-model agents read each paper in full and justify each decision with a quotation and its page. The verdict compares 50 classical RL papers from 2026 (language models excluded) with our own reading of 50 papers from 2018, using the same grids and the same readers. Twenty papers for each of four years between 2019 and 2024 date the change. The experiments rerun those of 2018 with PPO (in PyTorch and in JAX), SAC and CrossQ on four tasks of the MuJoCo physics simulator, with ten random seeds per program and per task, then twenty. A further strand fine-tunes two small language models with GRPO, a descendant of PPO.

Survey results. Our reading of 2018 departs from Pineau's figures: 55% of papers give all the hyperparameters of their method, where she counted 90%. The verdicts therefore compare our two readings. In classical RL, four practices changed. Linking the code rose from 31% to 74% ($+43$ points, $[+23, +58]$), showing the sensitivity to a setting from 44% to 80%, quantifying the compute, the machine time spent, from 27% to 58%, reporting an uncertainty from 63% to 88%. This last change is fragile: without thirteen papers read partly by another language model, it fails the Holm correction for multiple tests ($p_{\mathrm{Holm}} = 0.07$). All four rise slowly, with no jump after the 2019 NeurIPS checklist. The statistical test (6% then 12%), baselines described with all their settings (35% against 29%), evaluation on conditions held out of training (33% then 45%) and benchmark versions (23% then 26%) show no detectable change. Pineau was heard on what is published next to the result, much less on how the result itself is judged.

Experimental results. Among thirty PPO runs, two groups of five drawn at random fool Welch's t-test, which compares two means, in 1.2% of cases. Sorted groups fool it in 31%. Keeping the best three runs out of ten inflates the mean by 33%. Peter Henderson's script, which ignores the spread between random seeds, very probably explains the t of 9 from 2018, the statistic that declared two groups of runs of the same program different: applied to our TRPO runs, its t exceeds 9.09 in 38% of draws. At equal budget, with twenty random seeds, CrossQ leads on all four tasks. Physics modified by 20% sends the leader to last place on Hopper. On Qwen, with five random seeds, GRPO gains $+14$ points, with an interval from -3 to $+34$ that contains zero.

Code and data. Repository `github.com/warith-harchaoui/pineau-rl` (BSD 3-Clause licence): protocols and quoted readings (`study/`), training runs (`sim/`), statistics (`src/pineau_rl/`). `make verify` regenerates and checks every result.

Contents

Abstract **i**

1 Introduction **1**
1.1 Two talks in 2018 . . . 1
1.2 The question . . . 2
1.3 Two instruments . . . 2
1.4 Contributions . . . 2
1.5 Reading guide . . . 3

2 Context: reinforcement learning and the sixteen flaws **4**
2.1 The formal framework . . . 4
2.2 The four programs and TRPO . . . 5
2.3 The MuJoCo tasks . . . 6
2.4 What a random seed changes . . . 7
2.5 Evaluating an agent . . . 7
2.6 The sixteen flaws shown in 2018 . . . 8
2.6.1 The materials: what rerunning takes . . . 9
2.6.2 Tuning: hyperparameters and baselines . . . 10
2.6.3 Random seeds and statistics . . . 10
2.6.4 The reach of the conclusions . . . 11
2.7 Related work . . . 11
2.8 The vocabulary of language models . . . 12

3 The survey: method **14**
3.1 The protocol and its timeline . . . 14
3.2 Populations and draws . . . 17
3.3 The reading grids . . . 18
3.4 The readers . . . 22
3.5 The blind double readings . . . 23
3.6 The pilot and the harmonisation . . . 24
3.7 The checks of quotations and of negative decisions . . . 25
3.8 The calibration on 2018 and the verdict rule . . . 25
3.9 The statistics . . . 26
3.10 The robustness analyses . . . 27
3.11 Deviations from the protocol . . . 28

4 The survey: results **30**
4.1 The samples obtained . . . 30
4.2 The main comparison, with the same grid . . . 30
4.3 Rereading flaw by flaw . . . 34
4.4 What withstands the robustness analyses . . . 36
4.5 Our reading of 2018 against Pineau's figures . . . 38
4.6 The first draw and the language models . . . 40
4.7 The trajectory from 2018 to 2026 . . . 42
4.8 Agreement between readers . . . 44

4.9 The declarations of the papers . . . 47
4.10 Sensitivity to the title filter . . . 48
4.11 Synthesis . . . 48

5 The experiments: protocol and methods 49
5.1 A protocol written before the first training run . . . 49
5.2 Tools and versions . . . 50
5.3 The tasks . . . 50
5.4 The programs . . . 51
5.5 The budgets . . . 53
5.6 The random seeds . . . 54
5.7 Evaluation, held out . . . 55
5.8 The run format . . . 56
5.9 The statistics . . . 56
5.9.1 The interquartile mean . . . 56
5.9.2 The stratified bootstrap interval . . . 57
5.9.3 The probability of improvement . . . 57
5.9.4 The winner rule . . . 57
5.9.5 Student's and Welch's t-tests . . . 58
5.9.6 Three readings of the 2018 average t . . . 58
5.9.7 The statistic of Henderson's script . . . 58
5.9.8 Corrections for multiple comparisons . . . 59
5.9.9 Other tests . . . 59
5.9.10 Reproducible intervals . . . 59
5.10 The experiments, one by one . . . 60
5.10.1 E1: same code, different random seed . . . 61
5.10.2 B1: the puzzle of the t of 9 . . . 62
5.10.3 E2: does the ranking change from task to task? . . . 63
5.10.4 E3: two programs of the same algorithm . . . 63
5.10.5 E4: the best three out of ten . . . 63
5.10.6 E5: how many random seeds? . . . 63
5.10.7 E6: the tuned baseline . . . 64
5.10.8 E7: the modified physics . . . 64
5.10.9 B3: does the measure change the ranking? . . . 65
5.10.10 The consolidation: twenty random seeds and equal budget . . . 65
5.11 The queue and the compute . . . 66
5.12 What was written before the results, what was written after . . . 68

6 The experiments: results 70
6.1 E1: same code, another random seed . . . 70
6.2 B1: the puzzle of the t of 9 . . . 73
6.3 E2: the leading program changes from task to task . . . 76
6.4 E3: two programs of the same algorithm . . . 82
6.5 E4: the best three out of ten . . . 83
6.6 E5: how many random seeds? . . . 84
6.7 E6: a baseline that looked a little too good . . . 85
6.8 E7: the cost of a modified physics . . . 88
6.9 The physics modified by degrees . . . 91
6.10 B3: does the measure change the ranking? . . . 93
6.11 Ten or twenty random seeds, equal budget: what holds and what changes 97
6.12 What these experiments leave open . . . 97

7 The language-model strand **99**
7.1 Vocabulary . . . 99
7.2 Why this strand . . . 99
7.3 The GRPO algorithm, as we wrote it . . . 100
7.3.1 The task and the reward . . . 100
7.3.2 One training step . . . 100
7.3.3 Safeguards against a corrupted computation . . . 101
7.3.4 The measurement . . . 102
7.4 The protocol and its runs . . . 102
7.5 Results . . . 103
7.5.1 The reward drawn at random . . . 103
7.5.2 The true reward, in two stages . . . 103
7.5.3 Hardware and random seed, mixed up . . . 105
7.6 The cutoff trap . . . 107
7.6.1 The diagnosis . . . 107
7.6.2 What the twenty runs show . . . 107
7.7 What this strand establishes and what it does not . . . 109

8 Software and reproducibility **110**
8.1 Repository layout . . . 110
8.2 The `pineau_rl` package . . . 113
8.2.1 `pineau_rl.stats` . . . 113
8.2.2 `pineau_rl.bootstrap` . . . 114
8.2.3 `pineau_rl.runs` . . . 114
8.2.4 `pineau_rl.grid` . . . 114
8.2.5 `pineau_rl.fmt` . . . 115
8.3 File formats . . . 115
8.4 Every number checked against its file . . . 116
8.5 The targets of the `Makefile` . . . 116
8.6 Continuous integration . . . 117
8.7 Tests . . . 118
8.8 Frozen environments . . . 119
8.9 The trained policies . . . 119
8.10 Hardware and compute . . . 119
8.11 What is not in the repository . . . 120
8.12 Licence and citation . . . 120

9 Deviations from the protocols **121**
9.1 The principle . . . 121
9.2 What rests on our word . . . 121
9.3 The attempt ledgers . . . 122
9.4 The list of deviations . . . 122
9.5 Deviations of the survey . . . 125
9.6 Deviations of the simulations . . . 128
9.7 Deviations of the GRPO strand . . . 129
9.8 What these deviations say about the work . . . 130

10 Limitations and threats to validity **131**
10.1 Internal validity . . . 131
10.1.1 The survey readers are language-model agents . . . 131
10.1.2 Decisions taken after the results . . . 132

10.1.3 The simulation runs . . . 132
10.1.4 Hardware and random seed in the GRPO strand . . . 132
10.2 External validity . . . 132
10.2.1 The population of papers . . . 132
10.2.2 The simulations . . . 133
10.2.3 The budgets . . . 133
10.2.4 One platform . . . 133
10.2.5 The scale of the GRPO strand . . . 134
10.3 Construct validity . . . 134
10.4 Statistical conclusion validity . . . 134
10.5 What a new training run on another machine would test . . . 135

11 Conclusion **136**
11.1 The answer . . . 136
11.2 The main numbers . . . 136
11.3 Recommendations . . . 137
11.3.1 For authors . . . 137
11.3.2 For reviewers . . . 138
11.3.3 For conferences . . . 138
11.4 Future work . . . 139

A The complete reading grid **140**
A.1 What the reader does . . . 140
A.2 Pineau's grid (Q1 to Q8) . . . 140
A.3 Our additions in grid v1 (X1 to X7) . . . 141
A.4 Grid v2 (Y1 to Y12) . . . 142
A.5 Harmonisation . . . 144

B Supplementary tables **145**
B.1 RL on large language models in the survey . . . 145
B.2 Index of the generated tables . . . 145
B.3 The result files . . . 146

C The papers read **148**
C.1 The 2026 papers . . . 148
C.2 The 2018 papers . . . 152

D The software environments **155**
D.1 The platform . . . 155
D.2 The four environments . . . 155
D.3 Installation pitfalls . . . 155
D.4 The document toolchain . . . 157

Bibliography **158**

CHAPTER 1

Introduction

1.1 Two talks in 2018

Vancouver, then Montreal. In spring 2018, Joelle Pineau, a professor at McGill University in Montreal and head of Facebook's artificial intelligence research lab in that city, gives an invited talk at ICLR (*International Conference on Learning Representations*) in Vancouver (Pineau 2018a). The talk is titled 'Reproducibility, Reusability, and Robustness in Deep Reinforcement Learning'. In December, Pineau takes up and extends her argument at NeurIPS (*Neural Information Processing Systems*) in Montreal, as the Posner Lecture: 'Reproducible, Reusable, and Robust Reinforcement Learning' (Pineau 2018b). ICLR, NeurIPS and ICML (*International Conference on Machine Learning*) are the three main conferences of machine learning.

The starting question. When one team publishes a result, does another team that follows the recipe get the same one? Pineau starts from a survey by the journal *Nature*: of 1,576 researchers across all disciplines, more than 70% said they had already failed to reproduce a colleague's experiment, 52% saw a 'significant' reproducibility crisis and 38% a 'slight' one (Baker 2016). At a workshop in December 2017, she had put the same question to twenty machine learning specialists, 80% of whom saw a crisis, slight or significant (Pineau 2018a). She then quotes Buckheit and Donoho (1995), for whom a scientific article about computation is only the advertising of the research. The research itself is the complete software environment and the full set of instructions that produced the figures.

A growing field, costly errors. Pineau recalls that for nearly twenty years RL cited a single landmark success, the backgammon program TD-Gammon (Tesauro 1995), before AlphaGo (Silver et al. 2016). Around 2000, an RL workshop gathered thirty-five people. For 2016, she counts about thirteen thousand RL papers published in the year (Pineau 2018a). She adds that RL is approaching problems that matter: the design of medical treatments, the management of natural resources, of energy networks and of financial systems. Practitioners doubted too. In February 2018, Alex Irpan, of Google Brain, wrote that he answered no to anyone who asked him whether RL could solve their problem. He believed he was right at least 70% of the time (Irpan 2018). Pineau puts the address of this blog post on screen at NeurIPS.

The sixteen flaws. Pineau then shows, figure after figure, what RL lacks to deserve trust. Most of the figures come from her team's paper signed by Peter Henderson and co-authors (Henderson et al. 2018). We recorded sixteen flaws in the two talks, labelled F1 to F16 and sorted into four families: the materials needed to rerun an experiment (code, plan, software), the tuning of hyperparameters (the settings an experimenter fixes by hand) and of baselines (the existing methods a new one is compared with), random seeds (the integer from which a training run draws all its randomness) and statistics, the reach of the conclusions. The most famous is flaw F11: ten training

runs of the same program, split into two groups of five, give two curves that a test declares different, with $t = -9.09$ and $p = 0.0016$. Chapter 2 describes the sixteen flaws one by one (Table 2.2). Pineau goes beyond the diagnosis. She proposes a reproducibility checklist, which NeurIPS adopted in 2019 (Pineau et al. 2021). She also presents the ICLR 2018 reproducibility challenge, in which anyone could try to reproduce a submitted paper.

1.2 The question

Eight years have passed. RL has changed in size and in terrain: in 2025 and 2026, more than half of the RL papers in the first random draw of our survey apply what they call RL to large language models. The question of this report is simple: *did we listen to Joelle Pineau?* For each of the sixteen flaws, we ask whether the papers published between 2018 and 2026 still show it. We also measure what the flaw costs when the experiments of 2018 are rerun with the algorithms of 2026. We take her list as a grid and apply it to published papers, then to our own experiments, with the same severity.

1.3 Two instruments

The survey. The first instrument rereads papers with Pineau's reading grid (criteria Q1 to Q8), extended by seven criteria drawn from her ICLR talk (X1 to X7) and by a second grid of twelve criteria for the flaws she left unmeasured (Y1 to Y12). We read a first draw of 50 RL papers from 2025 and 2026, out of the 1,252 RL papers of ICLR 2026, ICML 2026 and NeurIPS 2025. We then read 50 classical RL papers from 2026 (robotics, games, control, language models excluded), 50 papers from 2018 and 20 papers per year for 2019, 2020, 2022 and 2024. Language-model agents read the papers. Each decision rests on an exact quotation and its page. The verdict compares classical RL in 2026 with our own reading of 2018, using the same grids and the same readers. Chapters 3 and 4 describe the method and the results.

The experiments. The second instrument reruns the experiments of 2018 under a protocol written before the first training run: four programs (PPO in PyTorch and in JAX, SAC, CrossQ) on four MuJoCo tasks, with ten random seeds per program and per task, then twenty. Thirty PPO runs and thirty TRPO runs on HalfCheetah complete the set, with one evaluation on random seeds never seen in training and another under modified physics. A further strand illustrates the same flaws on language models trained with GRPO. Chapters 5, 6 and 7 describe these experiments.

1.4 Contributions

1. **A same-grid measurement.** We apply Pineau's grid again to the papers of 2026 and calibrate it on our own reading of 50 papers from 2018. With the same grid on both sides, four practices changed in classical RL: linking the code (31% then 74%), showing the sensitivity to a setting (44% then 80%), quantifying the compute (27% then 58%), reporting an uncertainty (63% then 88%). Others show no detectable change, the statistical test among them (6% then 12%).

2. **A preregistered second grid.** Twelve criteria (Y1 to Y12), written with their decision rules before any coding, cover the performance measure, the tasks, the implementations, the versions, generalisation, sensitivity and recent statistical tools.
3. **A trajectory.** Four intermediate years read between 2018 and 2026 show the four practices rising slowly, with no jump right after the 2019 checklist.
4. **A battery of checks.** The survey comes with blind double readings, an audit of the quotations, a keyword search against the negative decisions, the published provenance of every reader, cuts by provenance, conference and reader, sensitivity analyses to the readers' model and to the title filter, a comparison with the checklists filled in by the authors.
5. **The experiments of 2018, rerun.** Two groups of five runs drawn at random fool a Welch test in only 1.2% of cases. Sorted, they fool it in 31%. Keeping the best three runs out of ten inflates the mean by 33%. The best program changes with the task, except at equal budget with twenty random seeds, where CrossQ leads on all four tasks. Physics modified by 20% moves the leader to last place on Hopper.
6. **An explanation of the t of 9.** The script Peter Henderson published in 2017 measures the spread between episodes and ignores the spread between random seeds. Applied to our TRPO runs, its t exceeds 9.09 in 38% of random draws.
7. **A bridge to language models.** We wrote our own GRPO and ran it on two small open models, with the true reward (the one that checks the answer) and with a coin-flip reward. With five random seeds, Qwen's mean gain with the true reward is $+14$ points, with an interval from -3 to $+34$ that contains zero.
8. **A repository that checks itself.** It holds the dated protocols, the readings with their quotations, the training results, the analysis scripts and checks that compare every number of the argued essay (`docs/conclusions-en.md`) and of the blog article with the result files (Chapter 8).

1.5 Reading guide

Chapter 2 sets out the formal framework of RL, describes the programs and the tasks, inventories the sixteen flaws and places the work among related work. Chapters 3 and 4 give the method and the results of the survey. Chapters 5 and 6 do the same for the MuJoCo experiments. Chapter 7 describes the GRPO strand. Chapter 8 describes the software, from training to documents. Chapter 9 lists every deviation from the three protocols. Chapter 10 gathers the limitations. Chapter 11 takes up each flaw with its verdict. The appendices give the complete rules of the grid (Appendix A), additional tables (Appendix B), the list of papers read (Appendix C) and the software environments (Appendix D).

A reader in a hurry can read the abstract, Table 2.2, Section 4.2 and Chapter 11. A reader who wants to redo a computation finds, in each caption, the result file that holds it.

CHAPTER 2

Context: reinforcement learning and the sixteen flaws

This chapter sets out the notions the rest of the report depends on. Section 2.1 gives the formal framework of reinforcement learning. Section 2.2 describes the four programs of our experiments and the TRPO algorithm, Section 2.3 the simulated tasks. Sections 2.4 and 2.5 state what a random seed fixes and how an agent is evaluated. Section 2.6 inventories the sixteen flaws Joelle Pineau showed in 2018. Section 2.7 places our work among related work. Section 2.8 fixes the vocabulary used for language models.

2.1 The formal framework

The Markov decision process. Reinforcement learning (RL) studies a program, the agent, that acts repeatedly in an environment. The standard framework is the Markov decision process (Puterman 1994; Sutton and Barto 2018), a 5-tuple

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

where $\mathcal{S}$ is the set of states, $\mathcal{A}$ the set of actions, $P(s' \mid s, a)$ the probability of moving to state s' after action a in state s, $r(s,a)$ the reward received and $\gamma \in [0,1)$ the discount factor, which weights a reward less the later it comes. A distribution ρ_0 draws the initial state. In the tasks of this report, $\mathcal{S}$ and $\mathcal{A}$ are subsets of $\mathbb{R}^d$: the state describes the positions and velocities of the joints of a simulated body, the action the torques applied to its motors.

Policy, episode and return. A policy $\pi_\theta(a \mid s)$ gives the distribution of the action in each state. In deep RL, the policy is a neural network with parameters θ. An episode is a sequence $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_T)$ obtained by drawing $s_0 \sim \rho_0$, then $a_t \sim \pi_\theta(\cdot \mid s_t)$ and $s_{t+1} \sim P(\cdot \mid s_t, a_t)$, until a terminal state or a maximum duration. The discounted return of an episode and the learning objective are

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

Learning seeks the θ that maximises J. The score reported in papers and in our experiments is almost always the undiscounted return $\sum_t r(s_t, a_t)$ of an episode, averaged over several evaluation episodes.

Value functions. The value of a state and the value of an action under the policy π are

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

The advantage $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$ measures how much more an action yields than the average action of the policy in that state. An actor-critic algorithm learns

both a policy (the actor) and an estimate of a value function (the critic). The critic's estimate reduces the variance of the actor's learning signal.

Policy gradient. For a policy differentiable in θ, the policy gradient theorem (Sutton and Barto 2018) gives

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

So-called *on-policy* methods, such as TRPO and PPO, estimate this expectation with episodes drawn from the current policy, then discard these episodes after the update. *Off-policy* methods, such as SAC and CrossQ, keep past transitions in a replay buffer and learn a critic Q from them. They reuse each transition several times. For the same number of environment steps, off-policy methods generally learn faster, at the price of heavier computation per step.

Hyperparameters. The parameters θ are learned. Other settings stay fixed by the experimenter: the learning rate, γ, the size of the networks, the batch size, the number of steps collected before each update. These are the hyperparameters. Flaws F4 to F7 of Section 2.6 concern them.

2.2 The four programs and TRPO

Our experiments use four programs: four specific implementations of an algorithm, each in a given library and version. The word 'algorithm' denotes the method described in a paper. The word 'program' denotes the code that runs it. Two programs of the same algorithm sometimes give different results, which is flaw F14. The pinned versions of each library are given in Table D.2. Chapter 5 details the budgets and the evaluation.

PPO, in PyTorch and in JAX. PPO (*Proximal Policy Optimization*) (Schulman et al. 2017) is an *on-policy* actor-critic algorithm. With the probability ratio $\rho_t(\theta) = \pi_\theta(a_t \mid s_t)/\pi_{\theta_{\text{old}}}(a_t \mid s_t)$ and an estimator $\hat{A}_t$ of the advantage, PPO maximises the clipped objective

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

to which a critic loss and an entropy bonus are added. Clipping removes the incentive for an update to move the new policy far from the old one. The program PPO PyTorch is the one in Stable-Baselines3 (Raffin et al. 2021). The program PPO JAX is the one in sbx, the JAX rewrite of Stable-Baselines (Bradbury et al. 2018; Raffin 2023). Both programs start from the same default values, as far as the two libraries allow. They collect experience in batches of 2,048 steps. PPO was the standard baseline of 2018. The protocol keeps it because it remains a common baseline in 2026 and because GRPO, the algorithm of our language-model strand (Chapter 7), derives from it.

SAC. SAC (*Soft Actor-Critic*) (Haarnoja et al. 2018) is an *off-policy* actor-critic algorithm. It maximises the return augmented by the entropy of the policy,

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

where the coefficient α sets the weight given to exploration and adapts during training. Two Q critics learned in parallel limit the overestimation of values. Delayed copies of

these critics, the target networks, stabilise the learning targets. The SAC program of our experiments is the one in sbx. SAC is a common *off-policy* baseline in continuous control.

CrossQ. CrossQ (Bhatt et al. 2024) descends from SAC. It removes the target networks and adds batch normalisation to the critics. The pairs (s, a) and (s', a') go through the critic together, so that the normalisation sees both distributions. Its authors report a sample efficiency, the score reached per environment step, at least equal to that of methods far more expensive in computation. The CrossQ program of our experiments is the one in sbx.

TRPO. TRPO (*Trust Region Policy Optimization*) (Schulman et al. 2015) is the algorithm of the most famous figure of 2018 (flaw F11). It maximises a surrogate objective under a Kullback-Leibler divergence constraint,

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

a problem solved by conjugate gradient followed by a line search. PPO is a simplification of TRPO. We use TRPO only in experiment B1, with the sb3-contrib 2.9.2 program and its default values, without any tuning (Chapter 5).

2.3 The MuJoCo tasks

Henderson and co-authors compared their algorithms on tasks of the MuJoCo simulator (*Multi-Joint dynamics with Contact*) (Henderson et al. 2018; Todorov et al. 2012). We take the same tasks in their 2026 version, the one in Gymnasium (Towers et al. 2024). Table 2.1 gives the dimensions of the state and action spaces, read in the pinned environment of our experiments. Each episode lasts at most 1,000 steps. Hopper and Walker2d stop earlier when the body falls.

Pineau judged these benchmarks ‘ridiculously simple’: states described by few numbers, deterministic transitions, a world weighing less than 100 kilobytes (Pineau 2018b). This criticism grounds flaw F16. We keep these tasks, since the aim is to rerun the experiments of 2018 as they were run.

Table 2.1. The four tasks of our experiments, in Gymnasium 1.1.1 and MuJoCo. The dimensions and the maximum length of an episode are read in the environment of our experiments (`sim/env`).

Task	state	action	Simulated body and reward
HalfCheetah-v5	17	6	articulated cheetah in two dimensions. Reward: forward velocity, minus a control cost
Hopper-v5	11	3	one-legged hopper. Reward: forward velocity, plus a bonus while the body stays upright
Walker2d-v5	17	6	two-legged walker. Reward: the same terms as Hopper
Swimmer-v5	8	2	three-segment swimmer in a viscous fluid. Reward: forward velocity

2.4 What a random seed changes

Definition. A training run draws many numbers at random. A computer produces this randomness from a starting integer, the random seed. In our programs, the random seed is passed to the library (`seed=SEED` in `sim/run.py`). The library derives from it the initialisation of the network weights, the draw of exploration actions, the order of the mini-batches and the sequence of initial states of the training environment. In this report, a *run* denotes a complete training with a given random seed.

What stays fixed. Same program, same random seed, same library versions and same machine: same result. Our logs confirm it for PPO PyTorch and PPO JAX: a run stopped then restarted recovers its evaluation points identically (`sim/SIM_PROTOCOL.md`, deviation 12 (e)). Changing the random seed changes all these draws. The final score of a given program then varies a great deal. Over our thirty runs of PPO PyTorch on HalfCheetah, the standard deviation is 34% of the mean (Chapter 6).

The statistical unit is the run. The random seed is a source of variance, in the same way as the draw of a sample in a survey. A comparison between two programs therefore rests on several independent runs per program, with an uncertainty computed over these runs. With one run per program, the comparison measures the luck of the draw. An episode, or a point on a curve, does not count as an independent observation, since it belongs to one run. Flaw F11 and experiment B1 show what happens when the episode is taken as the unit (Chapter 6).

2.5 Evaluating an agent

The score of a run. Evaluating an agent means playing episodes with its policy, without learning, then averaging their returns. Three choices remain open. A paper that leaves them unstated shows flaw F8.

- The measurement point: the score at the end of training, the best score reached along the curve, or the mean of the curve, which approximates the area under the curve divided by its length.
- The policy mode: stochastic as in training, or deterministic (the most probable action).
- The environment: the training one, or held-out conditions, such as other environment random seeds or modified physics (flaw F15).

Our programs are evaluated every 50,000 steps on 10 episodes, in deterministic mode, with environment random seeds $10{,}000 + i$, where i numbers the evaluation episodes. These random seeds never serve in training (Chapter 5).

Aggregating runs. With n runs of scores $x_{(1)} \le \cdots \le x_{(n)}$, the mean and the median react differently to extreme runs. Agarwal et al. (2021) recommend the interquartile mean (IQM), which discards the lowest quarter and the highest quarter of the runs, then averages the rest:

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

The IQM is the 25% trimmed mean of `scipy.stats.trim_mean`, which the rliable library calls: with ten runs, the IQM is the mean of the six middle runs. Its confidence interval comes from a stratified bootstrap. The runs are resampled with replacement within each task, the IQM is recomputed, then the 2.5% and 97.5% quantiles of the 2,000 replicates give the interval (`src/pineau_rl/bootstrap.py`).

Comparing two programs. The probability of improvement of X over Y is the probability that a run of X beats an independently drawn run of Y:

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

This probability equals the Mann-Whitney U statistic divided by $n_X n_Y$. The value 0.5 signals two indistinguishable programs. Our experiments declare a gap established when the 95% bootstrap interval of $P(X > Y)$ excludes 0.5. Welch's t-test (Welch 1947), which compares two means without assuming equal variances, serves only in experiments E1 and B1, since the 2018 figure used it.

2.6 The sixteen flaws shown in 2018

In her two talks of 2018, at ICLR and then at NeurIPS (Pineau 2018a,b), Pineau shows each flaw with an example, often a figure taken from her team's paper signed by Peter Henderson and co-authors (Henderson et al. 2018). We recorded sixteen flaws, in the order of the two talks, then sorted them into four families. Table 2.2 lists them with the slide or page that shows each one and the instrument of this report that measures it. The labels F1 to F16 serve only to find a flaw in the tables. Appendix A gives the decision rules of each reading criterion.

Table 2.2. The sixteen flaws Joelle Pineau showed in 2018 in deep reinforcement learning. For each one: the slide of her ICLR talk or the page of her NeurIPS document that shows it, then the reading criteria (Q, X, Y, Chapter 3) and the experiments (E, B, Chapter 5) that measure it in this report.

	Flaw	Where Pineau shows it	What measures it here
The materials: what rerunning takes			
F1	The paper published without its code	slide 6 (ICLR), page 29 (NeurIPS)	criterion Q5
F2	No plan for rerunning the experiment, as with AlphaGo	slides 9 to 12 (ICLR)	criteria Q3 and X7
F3	Hardware and software, which belong to the result	pages 34 and 35 (NeurIPS)	criteria X7 and Y10
Tuning: hyperparameters and baselines			
F4	The method's hyperparameters, given in part	page 29 (NeurIPS)	criterion Q3
F5	How the hyperparameters were chosen, left unsaid	page 29 (NeurIPS)	criterion Q6
F6	Settings that change everything and interact	slides 24 and 25 (ICLR)	criterion Y9

	Flaw	Where Pineau shows it	What measures it here
F7	Baselines tuned less carefully than the method	slide 25 (ICLR), page 29 (NeurIPS)	criteria Q4 and X4, experiment E6
Random seeds and statistics			
F8	A performance measure never defined	slides 26 and 27 (ICLR)	criteria Y1 and Y2, experiment B3
F9	Too few runs: three or five, never ten in her table	slide 28 (ICLR)	criterion X1, experiment E5
F10	Only the best runs out of N published	slides 29 and 30 (ICLR)	criterion X3, experiments E1 and E4
F11	Same code, another random seed: two ‘algorithms’	slide 31 (ICLR), page 22 (NeurIPS)	experiments E1 and B1
F12	No statistical test	page 29 (NeurIPS)	criteria Q8, X2, Y11 and Y12
The reach of the conclusions			
F13	The best algorithm changes with the task	slides 20 and 21 (ICLR)	criteria Y3 and Y4, experiment E2
F14	Two implementations of the same algorithm that diverge	slides 22 and 23 (ICLR)	criterion Y5, experiment E3
F15	Evaluation in the training conditions	pages 36 to 51 (NeurIPS)	criteria Q7 and Y8, experiment E7
F16	Benchmarks too simple, easy to learn by heart	pages 43 to 50 (NeurIPS)	criteria X6 and Y7

2.6.1 The materials: what rerunning takes

F1, the paper published without its code. An RL paper describes its method in a few pages. The program that produced the results runs to thousands of lines, full of choices the prose never mentions. Pineau takes up the line of Buckheit and Donoho (1995) quoted in Chapter 1, in which the two authors condensed an idea of Jon Claerbout: the paper is the advertising, the research is the code and the instructions that produced the figures. Of the 50 RL papers from 2018 she read, 55% link to their code.

F2, no plan for rerunning the experiment. Code falls short without the exact settings and an idea of the compute required. Pineau’s example is AlphaGo (Silver et al. 2016). The paper is detailed, with a long appendix. Her slide still asks whether that is enough: a rerun would also need the expert games, the code, the hyperparameters, the computers and the evaluation protocol. A discovery that no outside lab has the means to redo stays a claim.

F3, hardware and software. The same code launched on two machines sometimes gives two results. Graphics cards round computations differently from processors. A new version of a library changes a default value. Pineau’s checklist asks for ‘the computing infrastructure used’ and for the code ‘including all dependencies’.

2.6.2 Tuning: hyperparameters and baselines

F4, the method's hyperparameters, given in part. One setting left out blocks reproduction: the reader takes the default value of their library and gets something else. Pineau counts 90% of 2018 papers giving all the hyperparameters of their method.

F5, how the hyperparameters were chosen, left unsaid. A final value says nothing of how it was found. A team that tried a hundred combinations and kept the best one has optimised on its benchmark. The reader needs the search budget. Only 20% of 2018 papers state it.

F6, settings that change everything. Henderson and co-authors change one setting at a time. For PPO on HalfCheetah, going from a network of two layers of 64 neurons to a network of 400 then 300 neurons drops the score from 2,201 to $-1{,}180$. The settings interact: the best network for one algorithm stops being the best for another. Engstrom et al. (2020) and then Andrychowicz et al. (2021) confirmed this sensitivity at large scale.

F7, baselines tuned less carefully than the method. The author of a new method tunes it with care. The same author often takes the baseline with its default values. The comparison then pits a tuned method against an untuned one. Pineau counts 60% of 2018 papers giving the hyperparameters of their baselines. Eimer et al. (2023) call for the same tuning budget for both.

2.6.3 Random seeds and statistics

F8, a performance measure never defined. The final score, the mean over training and the best score reached do not always rank methods in the same order (Section 2.5). A paper that states neither which measure it uses nor what its error bars represent compares with no other paper.

F9, too few runs. Pineau shows a table of papers from 2015 to 2017: they launch three or five random seeds, none of them ten. With three runs, a mean depends heavily on the luck of one run. Colas et al. (2018) compute the number of random seeds needed to detect a given gap.

F10, the best runs out of N. A team launches ten runs, keeps the best ones and publishes their mean. Every number is true, yet the published mean exceeds what the method delivers on average. The spread also looks smaller than it is. Pineau illustrates the practice with ten curves of which she keeps the best three.

F11, same code, another random seed. Henderson and co-authors train TRPO ten times on HalfCheetah, changing only the random seed. They form two groups of five and plot the two mean curves, which stand clearly apart. A test declares the curves different ($t = -9.09$, $p = 0.0016$). On stage, Pineau specifies that the groups are not sorted. Chapters 5 and 6 return at length to this number (experiments E1 and B1).

F12, no statistical test. When two methods differ by a few points, a test or an interval tells whether the gap exceeds what the luck of the random seeds produces by itself. Only 5% of 2018 papers apply a test to their comparisons.

2.6.4 The reach of the conclusions

F13, the best algorithm changes with the task. Henderson and co-authors rank four algorithms on four tasks. The algorithm that leads on HalfCheetah stops leading on Hopper or Swimmer. A general conclusion drawn from one or two tasks generalises a special case.

F14, two implementations that diverge. Three implementations of TRPO, with the same network, finish on HalfCheetah at 1,939, 834 and 205. Comparing a method with 'TRPO' without saying which implementation was taken amounts to comparing it with an anonymous program.

F15, evaluation in the training conditions. In supervised learning, a model is evaluated on data it has never seen. In RL, the agent is often evaluated in the same world it trained in. Pineau shows a sentence she puts up for debate, according to which RL would be the only case of machine learning where testing on the training set is acceptable. She proposes separating the training and test random seeds, then the tasks (Zhang et al. 2018a,b). Only 10% of 2018 papers evaluate on held-out conditions.

F16, benchmarks too simple. A large enough agent memorises the right sequence of actions on a deterministic benchmark without understanding anything about the world (Machado et al. 2018). Pineau also asks for the version of each benchmark, so that a result can be rerun in the same world.

The remedies of 2018. Against the sixteen flaws, Pineau first sets a reproducibility checklist (pages 31 to 33 at NeurIPS). For each algorithm, it asks for a clear description, a complexity analysis, the link to the code and its dependencies. For each theoretical result, it asks for the statement, the assumptions and the proof. For each figure, it asks for the data collection, the split between training, validation and test, the excluded data, the range and selection method of the hyperparameters, the exact number of runs, the definition of the statistics and error bars, the hardware used. She also presents the ICLR 2018 reproducibility challenge (slides 37 to 43 at ICLR), in which anyone could try to reproduce a submitted paper. NeurIPS adopted the checklist in 2019 (Pineau et al. 2021).

2.7 Related work

On 9 November 2026, we searched arXiv, OpenReview and the conference proceedings. To our knowledge, no work reapplies Pineau's grid to recent RL papers while calibrating it on 2018. None reruns the protocol of Henderson and co-authors with the algorithms of 2026 under a protocol written in advance. Related work falls into five groups.

The 2018 experiment. Henderson et al. (2018) ground most of flaws F6 to F14: sensitivity to hyperparameters, to implementations, to random seeds, to tasks. Islam et al. (2017) preceded them on the same tasks. Lynnerup et al. (2020) rerun algorithms on a real robot arm. Bjorck et al. (2022) show, for control from images, that part of the variance between random seeds comes from runs that fail early through numerical instability.

Statistical method. Colas et al. (2018) run a power analysis of the number of random seeds. Jordan et al. (2020) discuss the choice of the performance measure, then Jordan et al. (2024) the cost of a rigorous comparison. Agarwal et al. (2021) contribute the IQM, stratified bootstrap intervals, performance profiles and the probability of improvement, gathered in the rliable library. They show that several improvements claimed on Atari games dissolve into the uncertainty once the intervals are computed. Patterson et al. (2024) turn these results into a guide. Eimer et al. (2023) call for an equal tuning budget for baselines. These works say what to do. Our survey measures whether papers do it.

Checklists and reproducibility challenges. Pineau et al. (2021) describe the NeurIPS 2019 reproducibility programme: the mandatory checklist, the reproducibility challenge and the code submission policy. The share of accepted papers with code rises from under 50% to 74.4%. Magnusson et al. (2023) measure the effect of the checklist of natural language processing conferences.

Audits of the literature. Gundersen and Kjensmo (2018) score 400 papers from the IJCAI and AAAI conferences on how they document methods, data and experiments. Coakley et al. (2026) track 56,800 papers from five artificial intelligence conferences from 2014 to 2024: the share that releases code and data rises from 11% to 64%. The rise begins before the checklists. In RL, the closest audit is the one by Gorsane et al. (2022): 75 papers on cooperative multi-agent RL, from 2016 to 2022, of which about a third quantify no uncertainty. None of these audits targets single-agent RL or the 2018 grid.

Language models. Hochlehnert et al. (2025) re-evaluate reasoning models and find standard deviations of 5 to 15 percentage points from one sampling random seed to the next on small benchmarks such as AIME 2024. Chandak et al. (2025) report, in a blog post that was not peer-reviewed, under-evaluated starting models in seven papers, the modern equivalent of flaw F7. Shao et al. (2025) show that a randomly drawn reward improves Qwen models in mathematics, with no such gain for Llama or OLMo models. Yue et al. (2025) show that the pass@k measure reverses a ranking depending on k, an instance of flaw F8.

2.8 The vocabulary of language models

The vocabulary of RL has spread to the generative AI models that write text. Some of its algorithms are borrowed, with only a loose resemblance to the original setting, to fine-tune the answers of a pretrained language model, with no agent acting in a world. This report fixes its words as follows.

- **Classical RL.** RL in the sense of Section 2.1: an agent, often trained from scratch, acts in an environment whose states evolve, in robotics, games or control. Offline RL belongs to it. Classical RL is the `classic` family of criterion X5.
- **RL on large models.** A label of convenience for the papers that call RL what they apply to a pretrained generative model conditioned on language (language model, vision-language model, text-to-image model). It is the `llm` family of criterion X5 (Chapter 3). The label takes up the papers' own word. It passes no judgement on the nature of what they do.

- **RLHF.** Learning from human feedback (*reinforcement learning from human feedback*) (Christiano et al. 2017; Ouyang et al. 2022). Annotators rank answers, a reward model learns their preferences, then the language model is fine-tuned to maximise this score. The acronym contains the letters RL. This report nonetheless keeps RLHF out of RL: the optimised policy produces a text in response to a prompt, then receives one score, with no environment whose state evolves under its actions.
- **RLVR and GRPO.** RLVR (*reinforcement learning with verifiable rewards*) replaces the reward model with a program that checks the answer: the final answer to a mathematics problem compared with the correct one, code run against tests (DeepSeek-AI 2025). GRPO (*Group Relative Policy Optimization*) (Shao et al. 2024) is the most widespread algorithm of this family. It derives from PPO: it draws a group of answers for each prompt and replaces the critic with the reward of each answer, standardised within the group. Chapter 7 describes our GRPO strand.

In this report, large language models are generative AI models. The readers of our survey are agents built on such a model (Chapter 3). There, the word ‘agent’ denotes a program that reads a PDF and fills in a grid, unrelated to the agent of a Markov decision process.

CHAPTER 3

The survey: method

The first instrument of this report rereads published papers with Pineau's reading grid, extended by seven criteria drawn from her ICLR talk and by a second grid of our own. This chapter describes the method in full: the timeline and the protocol (Section 3.1), the populations and the draws (Section 3.2), the grids and their decision rules (Section 3.3), the readers (Section 3.4), the double readings (Section 3.5), the harmonisation (Section 3.6), the checks of quotations and of negative decisions (Section 3.7), the calibration on 2018 and the verdict rule (Section 3.8), then the statistics (Section 3.9) and the robustness analyses (Section 3.10). Section 3.11 summarises the deviations from the protocol. Chapter 4 gives the results.

3.1 The protocol and its timeline

A protocol written before the reading. The survey protocol (`study/PROTOCOL.md`) is dated 8 November 2026, before the first paper was read. It fixes the question, the populations, the draws, the grid, the decision rules, the double reading and the analysis. Every later change is entered at the bottom of the file, in a numbered list of 'deviations'. Each deviation is dated and states whether it was decided before or after the result it concerns was known. The file counts 29 of them as of 21 November 2026. The instruction sheet given to the readers (`study/GRILLE.md`) summarises the protocol. The plan of the second grid (`study/planification_v2.md`), written on 9 November 2026 before any coding, was made public as it stood on 15 November 2026.

The stages. Figure 3.1 shows the complete flow, from the populations to the final counts. The stages followed one another as below.

1. **8 November.** Protocol, first draw of 50 papers from 2025 and 2026, calibration on 12 papers from 2018, reading with grid v1 (Q1 to Q8, X1 to X7) by seven readers, one of them a second reader.
2. **9 November.** Grid v2 (Y1 to Y12) and its decision rules, written before any coding (deviations 11 and 14). Enlargement of the two samples, written before any draw (deviation 12). Pilot (deviations 16 and 17), enlarged reading, audit of the raw readings, harmonisation (deviations 18 and 19), then first percentages. Same-grid verdict rule (deviation 20).
3. **10 and 11 November.** Robustness analyses (deviation 21), checklist declarations (A5, deviation 22), sensitivity to the title filter (A6, deviation 24), trajectory from 2019 to 2024 (A4, deviations 25 and 26).
4. **15 November.** Keyword search against the negative decisions (deviation 27), provenance of the readers (deviation 28), sensitivity to the change of model (deviation 29).

Every paper read comes from a draw written in advance

The survey, part one: from the published RL papers to the three samples

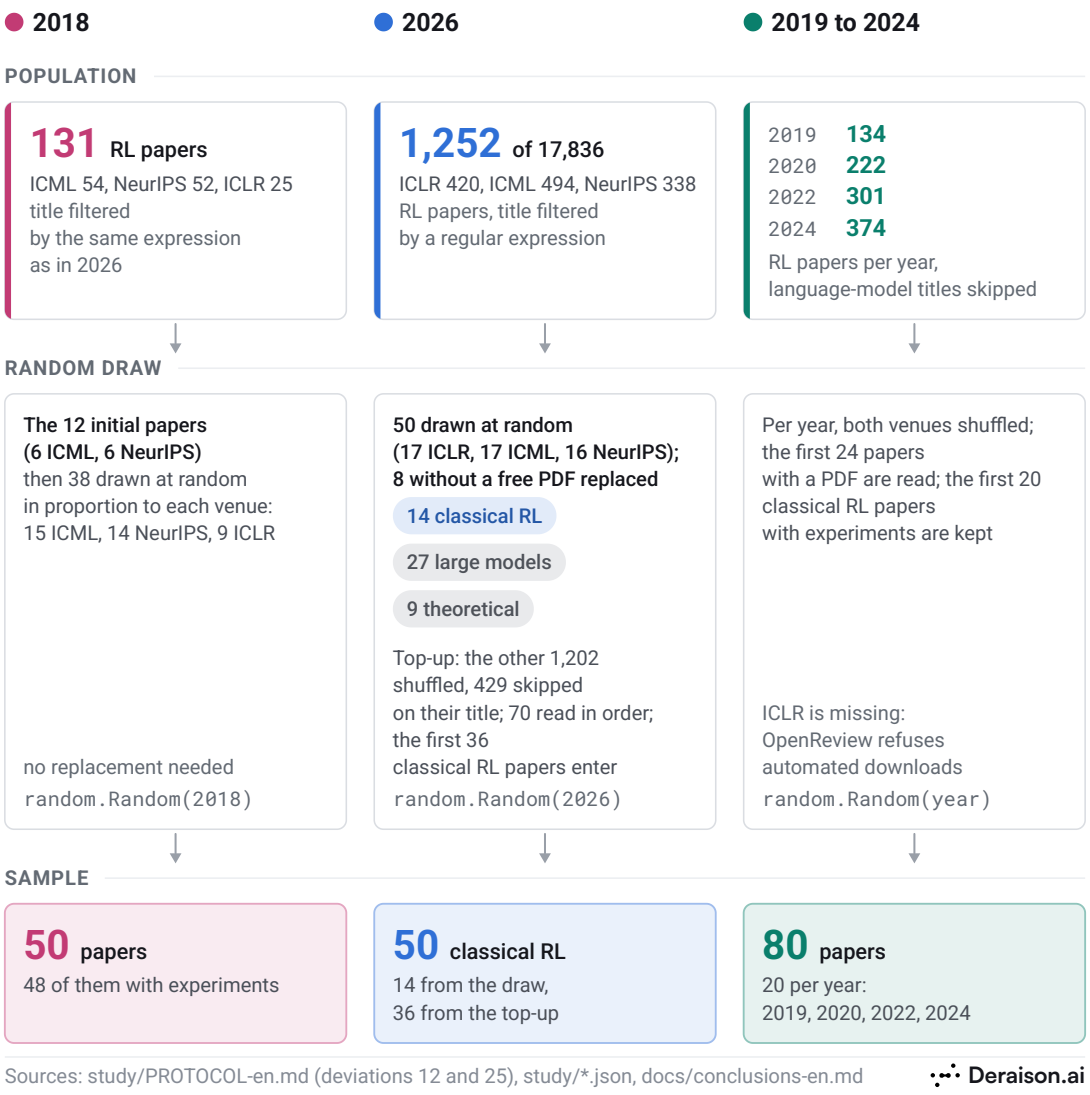


Figure 3.1. The survey, first part: from the populations to the samples. The RL papers of ICML, NeurIPS and ICLR 2018, those of ICLR 2026, ICML 2026 and NeurIPS 2025, those of ICML and NeurIPS in 2019, 2020, 2022 and 2024. Every sample comes from a draw whose random seed is written in the protocol: 50 papers from 2018 (48 with experiments), 50 classical RL papers from 2026, 20 papers per year for 2019, 2020, 2022 and 2024. The first 2026 draw is kept as context.



Figure 3.2. The survey, second part: from reading to comparisons. Every decision of the readers quotes a sentence of the paper with its page. Three sets of blind double readings, the harmonisation of the rules, the quote audit and the search for evidence against every 'no' test these decisions before the final comparisons.

3.2 Populations and draws

The definition of an RL paper. A paper belongs to the population if it is accepted to the main track of a conference and if its title matches, ignoring case, the regular expression

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

The same expression serves for every year. The three drawing scripts copy it verbatim: `study/sample_v2.py`, `study/trajectory_sample.py` and `study/title_filter.py`. A filter on the title misses the RL papers whose title does not say so. Analysis A6 measures this loss (Section 3.10).

The 2026 papers. The three most recent conferences as of 8 November 2026 are ICLR 2026, ICML 2026 and NeurIPS 2025. Their public lists hold 17,836 papers, 1,252 of them on RL: 420 at ICLR, 494 at ICML, 338 at NeurIPS. The list of RL papers is `study/rl_population_2026.json`. The first draw takes 17 papers from ICLR, 17 from ICML and 16 from NeurIPS with `random.Random(2018)`, in that order, by `sample()` on the list of each conference. The script `study/draw_2026.py` redoes this draw and checks that it recovers the same papers in the same order. We took the PDFs from the official proceedings for ICLR and from arXiv for ICML and NeurIPS, since OpenReview refuses automated downloads. An arXiv paper is accepted when its title is at least 85% similar to the one in the list (`difflib.SequenceMatcher`). Eight papers with no freely accessible PDF are replaced, in order, by a `random.Random(2026)` draw from the rest of the same conference (`study/replacements_2026.json`). The final sample is `study/final_2026.json`.

The initial calibration on 2018. To check that our readers roughly recover Pineau's figures, the protocol adds 6 RL papers from ICML 2018 and 6 from NeurIPS 2018, with the same filter and the same random seed (`study/calib_2018.json`). The separate 2018 appendices are read: the PMLR supplementary material for ICML, the proceedings archives for NeurIPS (deviation 8).

Enlargement A2: 50 papers from 2018. With 12 papers from 2018 and 14 classical RL papers from 2026, the statistical audit of the first draw (`study/audit_stats.py`) lacks the power to detect a gap of 30 points. Deviation 12, written on 9 November 2026 before any draw and before any reading of the added papers, enlarges both samples. For 2018, the population brings together the three conferences Pineau names: 131 RL papers, 54 at ICML, 52 at NeurIPS and 25 at ICLR (`study/rl_population_2018.json`). The ICLR 2018 list, 336 papers of the main track, orals and posters together, was taken from the official programme of the conference, since the OpenReview and dblp interfaces refused automated requests on 9 November 2026. The 50 papers are allocated to conferences in proportion to their size, by the largest-remainder method: 21 for ICML, 20 for NeurIPS, 9 for ICLR. The 12 initial papers stay. The 38 others (15, 14 and 9 per conference) are taken in the order of a `random.Random(2018)` shuffle of the rest of the population. A paper with no freely accessible PDF would be replaced by the next one in that order. No replacement was needed (`study/replacements_2018.json`). The PMLR and NeurIPS proceedings supplied the PDFs, an Internet Archive copy of them when the proceedings site did not respond, or arXiv by title. For ICLR 2018, OpenReview refusing automated downloads, the PDF is the Internet Archive copy of

the same OpenReview address, or else the arXiv version. The sample is `study/calib_2018_v2.json`.

Enlargement A3: 50 classical RL papers from 2026. The starting population is that of the 1,252 papers minus the 50 of the first draw, or 1,202 papers, shuffled with `random.Random(2026)`. A screen on the title alone, fixed in deviation 12, skips the titles that contain 'LLM', 'language model', 'reasoning', 'RLHF', 'RLVR', 'GRPO', 'preference', 'alignment', 'reward model', 'chain-of-thought', 'math' or 'agentic'. The screen sets aside 429 papers (`study/classic_screened_out_2026.json`) and leaves an ordered queue of 773 papers (`study/classic_queue_2026.json`). The readers code the family of each paper in the queue (criterion X5), in order. Reading stops once 36 `classic` papers are coded. The 9 ICLR 2026 papers at the head of the queue were unreachable on 9 November 2026, the proceedings site being down: they are skipped. The queue is frozen so that the selection does not depend on the reading date (deviation 17). In all, 70 papers of the queue were read: 36 enter the classical RL sample, 18 belong to large language models and 3 to theory. The other 13 were read after the last paper retained (table C.1). The first 36 `classic` papers in the order of the draw form the sample. The 18 `llm` papers and the 3 `theory` papers read before the thirty-sixth `classic` one stay coded and reported apart. They form no sample, since the title screen set aside many others like them. The 2026 classical RL sample brings together the 14 `classic` papers of the first draw and these 36 papers.

Trajectory A4: 2019, 2020, 2022 and 2024. Deviation 25, written on 10 November 2026 before any draw, adds four years to date the change. The population of each year brings together the RL papers of the main proceedings of ICML (PMLR volumes 97, 119, 162 and 235) and of NeurIPS. ICLR is missing, as OpenReview refuses automated downloads. After the title screen of deviation 12, the population counts 134 papers in 2019, 222 in 2020, 301 in 2022 and 374 in 2024 (`study/trajectory_queue.json`). The reading order is a `random.Random(<year>)` shuffle of the two conferences pooled. The readers read the first 24 papers of each year that have a PDF. The sample takes, per year, the first 20 `classic` papers with experiments, in order. In 2019 and 2020, the first 24 papers held only 19 and 18 of them: the next three papers of each year were read, as planned. Eight readers (M1 to M8) each read twelve papers taken from all years, so that no reader is confounded with a year. The NeurIPS 2019 to 2022 appendices are zip archives, which the first download had left aside. The raw readings of the first pass are recorded in the repository (commit `0d3519a`). Each reader then rereads their papers with the appendix and changes the decisions the appendix justifies.

Analyses A5 and A6. Analysis A5 (deviation 22) takes the 30 NeurIPS 2025 papers already read, without a draw: 16 from the first draw and 14 from the A3 queue. Analysis A6 (deviation 24) draws 400 papers, with `random.Random(2029)`, among the papers of the three 2026 lists whose title does not match the expression.

3.3 The reading grids

Two versions of the grid. Table 3.2 lists the 27 criteria. Grid v1 brings together Pineau's eight criteria (Q1 to Q8), as she presents them at NeurIPS (page 29), then seven criteria drawn from her ICLR talk (X1 to X7). Grid v2 (Y1 to Y12) covers the flaws that v1 leaves unmeasured. It was written on 9 November 2026 (deviation 11), with its decision rules

Table 3.1. The samples of the survey. For each: the population, the drawing rule with its random seed, what was read and what enters the analysis. Sources: the JSON files of `study/` named in the text.

Sample	Population	Draw	Analysed
First draw 2026	1,252 RL papers of ICLR 2026, ICML 2026, NeurIPS 2025	`Random(2018)`: 17, 17 and 16, then replacements by `Random(2026)`	50 papers, 45 with experiments
Initial calibration 2018	RL papers of ICML and NeurIPS 2018	`Random(2018)`: 6 and 6	12 papers
A2, 2018	131 RL papers of ICML, NeurIPS and ICLR 2018	quotas 21, 20 and 9, `Random(2018)` shuffle	50 papers, 48 with experiments
A3, classical RL 2026	1,202 remaining papers, 773 after the title screen	`Random(2026)` shuffle, stop at 36 `classic`	70 read, 14 + 36 = 50 retained
A4, 2019 to 2024	134, 222, 301 and 374 papers after the screen	`Random(<year>)` shuffle	20 papers per year
A5, checklists	30 NeurIPS 2025 papers already read	none	30 papers
A6, title filter	2026 papers outside the filter	`Random(2029)`: 400	400 abstracts, 20 papers read

(deviation 14), before any v2 coding. Y11 and Y12 were added the same day, after the search for related work, still before any coding. Appendix A reproduces the rules in full.

Table 3.2. The reading criteria. The 'Values' column gives the permitted answers. A `yes/no/na` criterion takes `na` when it does not apply. The last column gives the flaw of Table 2.2 that the criterion measures, or its role.

Code	Criterion	Values	Flaw or role
Grid v1: Pineau's criteria			
Q1	The paper has experiments	yes, no	scope
Q2	It uses neural networks	yes, no, na	population
Q3	All of the method's hyperparameters are given	yes, no, na	F2, F4
Q4	All of the baselines' hyperparameters are given	yes, no, na	F7
Q5	The code is linked	yes, no, na	F1
Q6	How the hyperparameters were chosen is described	yes, no, na	F5
Q7	The evaluation uses held-out conditions	yes, no, na	F15
Q8	A statistical test is applied to a comparison	yes, no, na	F12
Grid v1: our additions drawn from the ICLR talk			
X1	Number of independent training runs behind the main results	integer	F9
X2	An uncertainty is reported on the main results	yes, no, na	F12

Code	Criterion	Values	Flaw or role
X3	The best runs are selected	yes, no, na	F10
X4	The paper says it tuned the baselines with the same care	yes, no, na	F7
X5	Family: `llm`, `classic` or `theory`	one of the three	stratum
X6	For `llm`: number of base-model families	integer	F16
X7	The compute used is quantified	yes, no, na	F2, F3
Grid v2			
Y1	The performance measure is defined	yes, no, na	F8
Y2	The error bars are defined	yes, no, na	F8
Y3	Number of tasks behind the main conclusion	integer	F13
Y4	A task where the method does not win is shown	yes, no, na	F13
Y5	The implementation of the baselines is identified	yes, no, na	F14
Y6	Exclusions are stated, or their absence	yes, no, na	checklist
Y7	The version of the environments or benchmarks is given	yes, no, na	F16
Y8	A generalisation test outside the training conditions	yes, no, na	F15
Y9	The sensitivity to at least one hyperparameter is shown	yes, no, na	F6
Y10	The software dependencies are listed with their versions	yes, no, na	F3
Y11	The type of uncertainty is named	yes, no, na	F12
Y12	A statistical tool later than 2018 is used	yes, no, na	F12

Decision rules of grid v1. The protocol fixes the following rules, completed by deviations 1 to 6.

- **Q3 and Q4.** `yes` if a table or a paragraph gives the hyperparameters of all the main methods. 'Default values of [library]' without listing them counts as `yes` only if the library version is given. Q4 is `na` when no baseline is compared.
- **Q5.** A link to code in the paper. 'On request', 'will be released' and 'will be made public at <URL>' count as `no`. A project page counts as `yes` only if it leads to code, which was checked by visiting each page (deviations 6 and 19).
- **Q6.** The selection method is stated: grid, random search, budget, criterion. 'Tuned' and nothing more counts as `no`.
- **Q7.** Evaluation on tasks, environments, environment random seeds or datasets unseen in training and in tuning. For `llm`, benchmarks distinct from the RL

training data count as yes. A theory paper with toy experiments counts as no (deviation 4).

- **Q8.** A test applied to a comparison: t, Welch, Mann-Whitney, bootstrap with an interval, rliable IQM with stratified intervals. Error bars with no test count as no for Q8 and as yes for X2.
- **X1.** The smallest number of independent training runs stated for a main table or curve. For llm, evaluation samples (pass@k, '16 samples') do not count.
- **X3.** yes also when the paper's tables show that the main result is the best of several identical training runs, even if the text does not say so (deviation 5).
- **X4.** yes requires a described tuning of the baselines or an announced equal tuning budget. 'Same configuration as the method', with no tuning described for anyone, counts as no (deviation 1).
- **X5.** The family is read from the model that the RL trains (deviation 18): llm if the optimised policy is a pretrained generative model conditioned on language, classic otherwise, even in the service of a large model. theory is reserved for papers with no experiment or with toy experiments only.
- **X7.** yes requires at least one quantity: hours, GPU-hours, FLOP, or a number of GPUs with a duration. The name of the hardware, with no quantity, counts as no (deviation 2).

Decision rules of grid v2. Deviation 14 fixes the rules, deviation 17 refines them after the pilot, deviation 19 after the audit of the raw readings.

- **Scope.** If Q1 is no, all the other criteria are na. Y2 and Y11 are na if and only if X2 is no. Y4 is na if Y3 is 1. Y5 is na if and only if Q4 is na.
- **Y1.** The paper states what the reported number contains and at what point it is taken: final return, mean over the last N episodes, best point, area under the curve, pass@k with k and the number of samples. 'Return' or 'accuracy' and nothing more counts as no. A curve whose evaluation points are stated counts as a measurement point. For a quantity other than the score of a policy, the defined quantity suffices (deviation 19).
- **Y2.** The paper states what the band or the interval represents. Bars present but undefined count as no.
- **Y3.** An integer, the number of distinct training settings behind the main conclusion. Two difficulty levels of the same game, or the quality variants of an offline dataset, count as one. A meta-training counts as one. A setting present only in an appendix does not count. The analysis turns Y3 into yes when the integer is at least 2.
- **Y4.** A task-by-task result shows at least one task where the method does not win.
- **Y5.** For each main baseline, the origin of the code is stated: repository, named library, 'reimplemented by us', numbers copied with their source, released model designated by its exact name. An implied reimplementation does not count.
- **Y6.** The runs, random seeds or data removed are stated, or their absence is stated. Silence counts as no.

- **Y7.** The version of each main environment or benchmark: Gymnasium suffix, suite version, name that fixes a single published set. ‘MuJoCo’ without a version counts as no. ‘MATH-500’ counts as yes, bare ‘MATH’ as no.
- **Y8.** At least one evaluation on conditions absent from training and tuning. Stricter than Q7: for `llm`, a different benchmark counts only if it is presented as an out-of-distribution test. New training random seeds are not held-out conditions.
- **Y9.** At least one hyperparameter of the method is swept with the result shown. A component ablation does not count.
- **Y10.** The versions of the main libraries are given, or an environment file is cited. The repository is not visited, unless the paper points explicitly to the file.
- **Y11.** The type of uncertainty is named somewhere, with a `kind` field (`std`, `se`, `ci`, `iqr`, `minmax`, `other`).
- **Y12.** At least one tool among the IQM, the stratified bootstrap, rliable, performance profiles, the probability of improvement. A plain bootstrap that yields a reported interval counts.

What a share measures. For a criterion and a sample, the share is the number of yes divided by the number of yes and no. Papers with no experiment (Q1 no) and na values leave the denominator (`tally()` in `src/pineau_rl/grid.py`). A denominator smaller than the sample size therefore signals papers where the criterion does not apply, for instance a paper with no compared baseline for Q4. Pineau reports 100% experimental papers: her denominators are the 50 papers.

3.4 The readers

Language-model agents. The reading is entrusted to language-model agents, Claude Code subagents each launched with a written instruction, a list of identifiers and a folder to write in. The folder `study/readers/` gives the provenance of each agent, reconstructed on 15 November 2026 from the session transcripts (deviation 28): the exact instruction (`prompts/<reader>.md`), the model recorded in the transcript, the dates, the papers written and the way the PDFs were read (`readers.csv`). The 38 reading folders each match one agent and its papers, without discrepancy (`folder_check.csv`). All readers ran on the model `claude-opus-5-5`, except two that finished on `claude-opus-5` after a content filter of the provider interrupted an answer. The two are C2 (papers ML14 to ML17) and M4 (nine A4 papers), thirteen papers in all.

The work of a reader. Each reader reads the whole paper, appendices included. The reader converts the PDF to text with `pdftotext` and often looks at the page images. The reader writes one JSON file per paper, in `study/codes/<reader>/` for grid v1 and `study/codes_v2/<reader>/` for v2. For each criterion, the file gives the value, the page and an exact quotation of the paper copied word for word. If nothing justifies a yes, the quotation is ‘absent after reading the experiments, appendices and links sections’ and the page is `null`. The reader records hesitations in a `doubts` field. A reader never reads another reader’s codes, nor the results, nor the documents of `docs/`. For a paper already read in v1, the v2 reader receives only the v1 values of Q1, Q4 and X2, which fix the scope rules.

The mechanical check. The script `study/check_codes.py` checks each v2 file before any percentage. It checks the schema (criteria present, permitted values, integers, `kind` field of Y11) and the scope rules (`validate()` in `src/pineau_rl/grid.py`). It then searches for each quotation in the text of the PDF and of its appendix. The reader's notes in square brackets are removed. A quotation stitched from pieces joined by '[...]' is cut into its pieces. A piece of fewer than 12 letters is too short to test. The comparison uses letters and digits only, so that spaces, hyphenation, ligatures and mathematical symbols do not count. A piece not found verbatim is searched for approximately: it passes if 90% of its characters are found in a window of the text. A failing file goes back to its reader. A second failure on the same criterion is decided from the PDF and recorded in `study/overrides_v2.json`.

Three facts the protocol did not state. The reconstruction of deviation 28 established three facts the protocol did not state. Readers C2 and M4 finished on the model `claude-opus-5`. The primary reader M7 opened the file K002 of reader K1 as a format example. K002 is not M7's and enters no double reading. The grid v1 readings (readers A to G) took place on 8 November in a session opened in a folder other than the repository. The second readers G, G2, R2 and R3 opened no file of another reader.

3.5 The blind double readings

Four second readers. A second reader rereads papers without seeing the first reading or knowing who made it.

- **G (grid v1) and G2 (grid v2).** The first 10 papers of the first draw in identifier order after a `random.Random(7)` shuffle.
- **R2 (both grids).** 10 papers drawn by `random.Random(7)` among the papers added by A2 and A3: 5 from 2018 and 5 from 2026.
- **R3 (both grids).** 10 papers drawn by `random.Random(2030)` among the 80 of A4, reread with the appendices (deviation 26).

The quantity compared. Agreement is computed on the raw readings, before any harmonisation: harmonisation applies to both readings and would inflate agreement. For each criterion, `study/agree_v2.py` keeps the papers where both readers answered `yes` or `no` (Y3 transformed as in the analysis), then computes the share of identical answers and Cohen's kappa when both readers vary. A criterion below 70% agreement is called fragile. The agreement of grid v1 on the first draw is computed by `study/audit_stats.py`, which also counts the `na` values.

The human rereading, planned then abandoned. Deviation 13 planned for a human, the author, to reread 10 papers from 2026 alone on seven criteria, without seeing the agents' readings. The papers had been drawn by `random.Random(2027)`. Deviation 23 abandons this rereading on 10 November 2026, before it started. The folder `study/relecture_humaine/` remains as a trace of the plan. No human reread the papers blind. Each decision remains checkable through its quotation and its page.

3.6 The pilot and the harmonisation

The pilot. Before the primary readers, a pilot reader codes three papers (deviation 16): LR08 (`llm`) and ML15 (`classic`), chosen by `random.Random(11)` among the 40 papers from 2026 outside the double reading, then C218, the first of the 38 new papers from 2018 after a `random.Random(11)` shuffle. The pilot serves only to find ambiguous rules and to test `check_codes.py`. Its files (`codes_v2/pilot/`) enter no analysis. The three papers are recoded by their regular reader. Deviation 17 writes down the resulting clarifications, before the main coding and before any v2 percentage.

The harmonisation of grid v1. Deviations 1 to 6 were written during the coding of the first draw, before aggregation. They are applied from the quotations, in `study/overrides.json`. The check of the quotations not found showed on 9 November 2026 that these rules had been applied only to the 2026 papers (deviation 15). The comparison between years therefore mixed two slightly different grids. The same rules were applied to the 12 papers from 2018, with no new rule. For X4, C11, C13, C14 and C16 switch to `no`. For Q7, C14 switches from `na` to `no`. For X7, C14 switches to `no`. The script `study/analyze_final.py` now regenerates `study/results_final.json`, until then written without a script. This correction was decided after seeing the percentages, which the protocol declares.

The harmonisation of grid v2. After the enlarged reading, an audit rereads all the v2 files of the primary readers, criterion by criterion, without computing any share (`study/harmonisation_audit.md`, 170 files). It finds codes contrary to the rules and ten cases that no rule covered. Deviation 19, written on 9 November 2026 before any percentage, rules on these cases for both years. Its decisions are applied from the quotations, each with its reason, in `study/overrides_v2.json`: 80 changes, of which Y1 21, Y3 5 (and Y4 5 through the scope rule), Y5 10, Y6 3, Y7 19, Y8 6, Y9 9 and Q5 2. The project pages given as a link to the code were visited on 9 November 2026. Five lead to the code, two do not. The link of K090 leads to an expired anonymous mirror (HTTP 410). The link of C19 leads to a library released after the paper, without its scripts. The author of the protocol ruled on two cases, NE16 and ML14, all of whose experiments are in the appendix, so that the rule on settings present only in an appendix does not apply to them. Deviation 18 refines X5 before the classical RL sample is frozen: applied to the 120 papers read, the rule changes one code, K006, from `llm` to `classic`.

Three criteria with no new rule. For X3, X4 and X7, the audit found borderline cases, such as compute reduced to inference time or failed random seeds set aside. We added no new rule. A rule written at that point would have applied only to the papers of the enlarged reading, which deviation 15 forbids: the two years would again have been read with two different grids. These cases stay coded as the reader read them.

The final value of a criterion. The function `value()` of `src/pineau_rl/grid.py` computes the value that enters the analysis. It takes the v2 reading of the primary reader when that reading contains the criterion, with `overrides_v2.json`. Otherwise it takes the v1 reading, with `overrides.json`. The readings of the second readers and of the pilot never enter the analysis.

3.7 The checks of quotations and of negative decisions

The audit of the grid v1 quotations. The first pass of the automatic check on the v1 files left 89 quotations not found. An audit agent checked all of them in the PDFs (`study/v1_quotes_audit-en.md`). None is invented: 83 are present word for word, 1 is a faithful paraphrase, 5 have a value contrary to the rules. The failures had three causes: French comments that the reader had added to an exact English quotation, the two-column layout that `pdftotext` interleaves, three NeurIPS 2018 appendices missing from the `study/pdf/` folder. The five values contrary to the rules led to deviation 15.

The keyword search against the 'no' decisions. A `yes` decision rests on a quotation and a page. A `no` decision rests on nothing. Yet the 'changed' verdicts rest mostly on the `no` decisions of 2018. Deviation 27, written on 15 November 2026 after the results and before any search, checks every final `no` decision of the two samples of the main comparison against the text of the paper, on the seven criteria Q4, Q5, Q7, Q8, X2, X7 and Y9. The script `study/falsify_no.py` extracts the text of the paper and of its appendix, removes the references section, then searches for patterns fixed in the deviation and never modified afterwards. For example, the pattern of Q5 is

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

Each match is kept with 300 characters of context. An arbiter, an agent that read neither sample and sees neither the paper nor the year, says according to the rules of `GRILLE.md` whether the excerpt suffices to turn the decision into `yes`, or whether it remains doubtful. The batches the arbiter saw and its raw answers are in `study/falsify_batches/`. The script `study/falsify_stats.py` then redoes the main comparison with the turned decisions, without changing the test, as the deviation announces. It also redoes the comparison with the turned decisions plus the doubtful ones, a bound the deviation did not announce.

3.8 The calibration on 2018 and the verdict rule

The reason for rereading 2018. Pineau describes neither her draw nor how she decided doubtful cases. Comparing our 2026 readings with her figures would mix two changes: the change in practice and the change of grid. The calibration measures the second. From the first 12 papers on, our grid looked stricter than hers on Q3 and more lenient on Q7 (deviation 7). The protocol then planned to conclude that a practice had changed only if it held both against her figures and against our own reading of 2018.

The same-grid verdict rule. With 50 papers from 2018, the secondary comparison of deviation 12 confirmed that our grid reads 2018 differently from Pineau (Chapter 4). Deviation 20, decided on 9 November 2026 after these first results, draws from it the rule of the report: a verdict rests only on the main comparison of deviation 12, our 50 classical RL papers from 2026 against our reading of 50 papers from 2018, with the same grids and the same readers. This comparison had been fixed before any reading of the added papers. The comparisons with Pineau's figures stay on display as context. The change of rule was decided after seeing these results, which the protocol declares. A criterion 'changed' when its gap passes the 5% threshold after the Holm correction.

3.9 The statistics

All the statistical functions are in `src/pineau_rl/stats.py`. They call SciPy and statsmodels and are deterministic. Six scripts use them: `study/extension_stats.py` (main comparison and secondary comparisons), `study/trajectory.py` (A4), `study/agree_v2.py` and `study/audit_stats.py` (agreement), `study/robustness.py` (robustness) and `study/falsify_stats.py` (negative decisions). In what follows, k is the number of yes, n the denominator and $\hat{p} = k/n$.

Wilson interval. The 95% interval of a share is that of Wilson (1927), suited to small samples and to shares close to 0 or 1. With $z = \Phi^{-1}(0.975)$,

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

Newcombe interval for a difference. Pineau's 50 papers also form a sample. Treating her shares as exact, the 55% of papers with code for instance, would overstate the certainty of the verdicts. The gap $d = \hat{p}_a - \hat{p}_b$ between two independent samples therefore receives the hybrid interval of Newcombe (1998) (his method 10). With the Wilson intervals (l_a, u_a) and (l_b, u_b) of each share,

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

The computation is that of `confint_proportions_2indep(method="newcomb")` in statsmodels. For one of Pineau's figures, the share enters the computation as $k = \text{round}(50\,p)$ out of $n = 50$.

Fisher's exact test. The p-value of a comparison is that of the exact test of Fisher (1922) on the table $\left(\begin{smallmatrix} k_a & n_a - k_a \\ k_b & n_b - k_b \end{smallmatrix}\right)$. Under the null hypothesis and with fixed margins, k_a follows a hypergeometric distribution. The two-sided p-value sums the probabilities of the tables as probable as the observed one or less probable (`scipy.stats.fisher_exact`).

Holm correction. Testing several criteria at once multiplies the chances of crossing the threshold by chance. The correction of Holm (1979) controls the probability of at least one false positive in a family of m tests. With the sorted p-values $p_{(1)} \le \cdots \le p_{(m)}$, the adjusted value is

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

The families are fixed in the scripts. The main comparison has two: the 11 tested v1 criteria (Q2 to Q8, X2, X3, X4, X7) and, separately, the 12 v2 criteria. The secondary comparisons, 2026 against Pineau and our 2018 against Pineau, each take as their family the six criteria Q3 to Q8 for which Pineau gives a comparable figure. For the trajectory, each family brings together the five tested criteria (Q5, Y9, X7, X2, Q8).

Cochran-Armitage trend test. For trajectory A4, the test of Cochran (1954) and Armitage (1955) looks for a linear trend of the share over the years. With $K = \sum_i k_i$, $N = \sum_i n_i$, $\bar{p} = K/N$, the scores x_i equal to the years (2018, 2019, 2020, 2022, 2024, 2026) and $\bar{x} = \sum_i x_i n_i / N$,

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

The two-sided p-value is read from the normal distribution. Two Newcombe gaps complete this test, each with a Fisher test: 2019 and 2020 pooled against 2018 (the immediate effect of the checklist), then 2022 and 2024 pooled against 2019 and 2020 (the later effect).

Cohen's kappa. The raw agreement between two readers depends on the frequency of the answers: two readers who almost always answer no often agree by chance. The kappa of Cohen (1960) removes this share:

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

where p_o is the share of identical answers and $\pi_a(c)$ the share of answers c given by reader a. The kappa is undefined when $p_e = 1$. The script `agree_v2.py` does not compute it when one of the two readers always gives the same answer. On ten papers, one or two disagreements drag the kappa down towards 0.5 or 0.6 when a reader answers almost always the same way. The kappa is read together with the share of agreement and the number of papers.

Cochran-Mantel-Haenszel test and Spearman correlation. For the robustness analyses, a Cochran-Mantel-Haenszel test (Mantel and Haenszel 1959) combines the 2×2 tables of each conference (ICML, NeurIPS, ICLR), with a Haldane correction of 0.5 per cell and a continuity correction (`statsmodels.stats.contingency_tables.StratifiedTable`). The Spearman correlation between the cost or risk scores and the observed gaps receives a permutation p-value: 10,000 permutations of the gaps, NumPy generator with random seed 2026.

3.10 The robustness analyses

All these analyses are exploratory. None changes a verdict of deviation 20. They only say whether a verdict is fragile.

Four cuts of the verdicts (deviation 21, a). For the four criteria that changed (Q5, Y9, X7, X2) and the two intervals that avoid zero without passing Holm (Q6, Y12), `study/robustness.py` redoes the comparison in four ways.

- **(a1) Provenance.** Classical RL in 2026 is split by provenance: the 14 papers of the first draw, with no title screen, against the 36 of the A3 queue, with the screen. The main comparison is redone with each provenance alone.
- **(a2) PDF source.** The 2026 papers are split by the source of the PDF read, arXiv or the conference site.
- **(a3) Conference.** A Cochran-Mantel-Haenszel test stratifies by conference, to rule out a composition effect.
- **(a4) Reader.** The comparison is redone removing each primary reader in turn.

A verdict is called fragile if the gap changes sign in a cut, or if the recomputed Newcombe interval contains zero while each side keeps at least 35 papers. Cut (a2) first read a damaged local copy of a file. Redone on the repository file, it sets 45 papers read on arXiv against 5 read on a conference site and settles nothing (deviation 21, c).

The cost and the risk of each practice (deviation 21, b). An earlier version of our conclusions claimed that Pineau had been heard where listening to her cost little. That ranking came from us, after seeing the gaps. To put it to the test, three independent raters each score the 17 practice criteria of the main comparison on two scales from 1 to 5: the cost C for an author who already has the main results, then the risk R that the practice weakens or reverses the conclusion. The raters are three language-model agents of three different sizes, with no access to the results or to the repository, who receive only the title and the rule of each criterion (`study/blind_cost/`). The test is the Spearman correlation between the median of the three scores and the observed gap, with X3 reversed because a lower share is progress for it. A clear negative correlation would support the earlier claim.

Without the thirteen papers read partly by another model (deviation 29). The thirteen papers finished by `claude-opus-5` are removed, then the main comparison and the trajectory are redone with the same tests (environment variable `EXCLUDE`, outputs `extension_stats_sans_opus5.json` and `trajectory_sans_opus5.json`). A verdict changes if a gap or a trend moves from one side of the threshold to the other after the Holm correction.

The title filter (A6, deviation 24). For each of the 400 papers drawn outside the filter, the abstract is read on the official page. A paper is 'hidden RL' if its abstract matches the same regular expression as the titles. The script `study/title_filter.py` gives the share of hidden RL with its Wilson interval, the estimated number of papers missed (this share times the size of the population outside the filter) and their fraction of the total. The first 20 hidden-RL papers in the order of the draw are read with both grids, then compared descriptively with the 2026 sample on the four criteria that changed (`study/title_filter_compare.py`). Four of them had no freely accessible PDF and were replaced by the next ones in order.

The declarations of the papers (A5, deviation 22). NeurIPS 2025 papers attach the checklist filled in by their authors. The script `study/checklist.py` retrieves the official proceedings version of each of the 30 papers by its title, then extracts the answer and the justification of four questions: 'Open access to data and code', 'Experimental setting/details', 'Experiment statistical significance', 'Experiments compute resources'. The comparison, fixed in advance, sets them against our harmonised codes: Q5, Q3, X2 for an uncertainty and Q8 for a test, X7. The measure is the share of papers that declare 'Yes' while our code is no, with its Wilson interval. For each disagreement on code, the script looks for a link to code in the proceedings version, because our codes often come from the arXiv version.

3.11 Deviations from the protocol

The survey protocol counts 29 deviations. Table 9.1 gives all of them with their date and their position relative to the results, next to those of the two other protocols (Chapter 9). They fall into five groups.

- **Rules clarified before any percentage**: deviations 1 to 6 (grid v1), 14, 16 to 19 (grid v2, pilot, X5 family, harmonisation). They remove ambiguities without targeting a number.

- **Instruments added before their own computation**: deviations 11 and 12 (grid v2, enlargement), 13 (human rereading, abandoned by deviation 23), 22, 24, 25 and 26 (A5, A6, A4 and its second reading).
- **Decisions taken after seeing results**: deviations 7 (initial calibration), 15 (2018 harmonisation caught up), 20 (same-grid verdict rule), 21, 27, 28 and 29 (analyses answering a critical review, written before their own computation). The protocol declares them as such.
- **Incidents with no effect on the data**: deviations 9 (contact address removed from the download header) and 10 (working scripts overwritten in a shared temporary folder, codes intact).
- **Reading facts**: deviation 8 (appendices of 2018 and 2026).

CHAPTER 4

The survey: results

This chapter gives all the results of the survey. Section 4.1 describes the samples obtained. Section 4.2 gives the main comparison with the same grid on both sides, criterion by criterion, then Section 4.3 rereads it flaw by flaw. Section 4.4 says what withstands the robustness analyses. Section 4.5 compares our reading of 2018 with Pineau's figures. Section 4.6 describes the first 2026 draw and the papers on language models. Section 4.7 dates the change from 2019 to 2024. Section 4.8 measures the agreement between readers. Sections 4.9 and 4.10 give analyses A5 and A6. Every number is taken from the JSON files of `study/` named in each caption.

4.1 The samples obtained

The first 2026 draw. The 50 papers drawn from the 1,252 RL papers of 2025 and 2026 break down as follows: 27 apply what they call RL to large language models (`llm` family), 14 belong to classical RL (`classic`) and 9 are theoretical (`theory`), 5 of which have no experiment at all. The percentages are computed over the 45 papers that have experiments.

Our reading of 2018. The 50 papers from 2018 (21 from ICML, 20 from NeurIPS, 9 from ICLR) include 48 papers with experiments. Of these 48 papers, one (C14) compares an exact method with no hyperparameter: Q3 and Q6 are `na` for it, hence a denominator of 47 for these two criteria. Q4 and Y5 are `na` when a paper compares no baseline.

Classical RL in 2026. The 50 papers bring together the 14 `classic` papers of the first draw and the first 36 `classic` papers of the A3 queue (`extension_stats.json`, field `samples`). All 50 have experiments. Seventy papers of the queue were read: reaching the thirty-sixth `classic` paper took 57 readings, 18 `llm` papers and 3 `theory` papers among them despite the title screen. Thirteen more papers were read after it.

'50 against 48'. The blog article and our figures speak of '50 classical RL papers from 2026' against '50 papers from 2018'. Both samples do hold 50 papers read. The 2018 percentages are nevertheless computed over 48 papers, sometimes 47, because two 2018 papers have no experiment. Those of 2026 are computed over 50 papers, sometimes 49 when a criterion does not apply. The exact denominator of each share appears in the tables of this chapter. The label '50 against 50' thus describes the samples drawn, the label '48 against 50' the samples analysed for most criteria.

4.2 The main comparison, with the same grid

The two samples compared. The main comparison sets our 50 classical RL papers from 2026 against our reading of 50 papers from 2018, with the same grids, the same har-

monised rules and the same readers (Section 3.8). Table 4.1 gives the criteria of grid v1, Table 4.2 those of grid v2. For each criterion, the tables give the share in 2018 and in 2026 with their Wilson intervals, the gap with its Newcombe interval, the p-value of Fisher's exact test and the value adjusted by the Holm correction, applied separately to the 11 v1 criteria and to the 12 v2 criteria. Figure 4.1 draws these gaps criterion by criterion. Figure 4.2 takes up the eight practices of the blog article.

Four practices changed. Four gaps pass the Holm correction.

- **Code linked (Q5)** goes from 31% (15 of 48) to 74% (37 of 50): $+43$ points, $[+23,\ +58]$, $p_{\text{Holm}} = 0.0004$.
- **Sensitivity to a hyperparameter shown (Y9)** goes from 44% (21 of 48) to 80% (40 of 50): $+36$ points, $[+17,\ +52]$, $p_{\text{Holm}} = 0.004$.
- **Compute quantified (X7)** goes from 27% (13 of 48) to 58% (29 of 50): $+31$ points, $[+11,\ +47]$, $p_{\text{Holm}} = 0.02$.
- **Uncertainty reported (X2)** goes from 63% (30 of 48) to 88% (44 of 50): $+26$ points, $[+8,\ +41]$, $p_{\text{Holm}} = 0.04$. This verdict is the most fragile (Section 4.4).

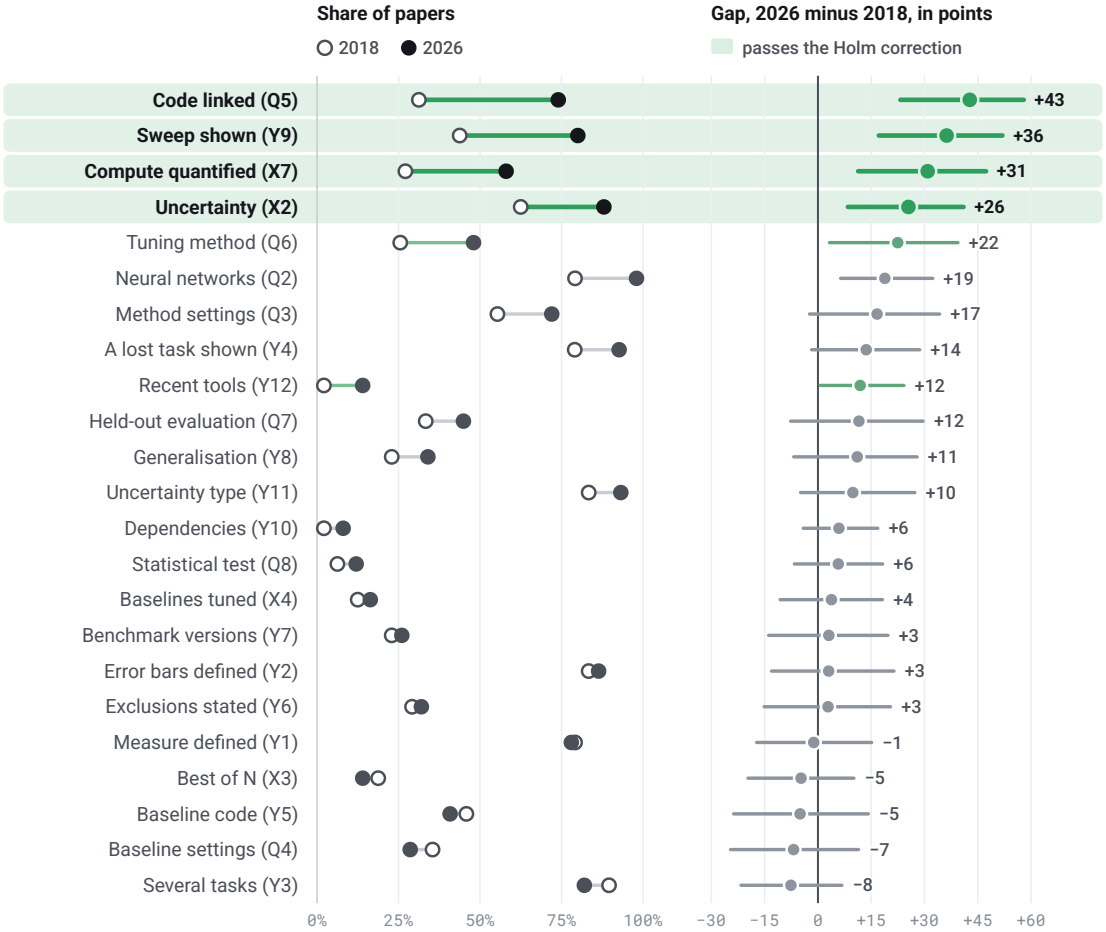
A fifth criterion also passes the correction: the use of neural networks (Q2), from 79% (38 of 48) to 98% (49 of 50), $+19$ points, $[+6,\ +32]$, $p_{\text{Holm}} = 0.03$. Q2 describes the

Table 4.1. Pineau's grid (criteria Q2 to Q8) and the X criteria drawn from her ICLR 2018 talk, read with the same grid and the same readers on two samples: our reading of 50 papers from 2018 (ICML, NeurIPS, ICLR) and 50 classical RL papers from 2026 (14 from the first draw, 36 from the A3 queue). This is the main comparison, the only one a verdict rests on (deviation 20 of `study/PROTOCOL.md`). k/n: papers that meet the criterion out of the papers where it applies (papers with experiments, Q1 not 'no'); the percentage carries its 95% Wilson interval. Gap: classical RL 2026 minus 2018, in percentage points, with the 95% Newcombe interval; p: two-sided Fisher exact test; Holm p: Holm correction over the eleven criteria of grid v1. **In bold**, the criteria whose gap passes the Holm correction (Holm $p < 0.05$). For X3, a lower share is progress; Q2 (neural networks) describes the population studied, not a reproducibility practice. Pineau's own figures are in Table 4.4. Source: `study/extension_stats.json` (script `study/extension_stats.py`, written before any reading of the added papers).

		Our reading of 2018		Classical RL 2026		Gap		
Code	Criterion	k/n	% [Wilson]	k/n	% [Wilson]	[Newcombe]	p	Holm p
Q2	**Neural networks used**	**38/48**	**79 [66, 88]**	**49/50**	**98 [90, 100]**	**+19 [+6, +32]**	**0.003**	**0.03**
Q3	All of the method's hyperparameters	26/47	55 [41, 69]	36/50	72 [58, 83]	+17 [−2, +34]	0.10	0.58
Q4	All of the baselines' hyperparameters	17/48	35 [23, 50]	14/49	29 [18, 42]	−7 [−25, +11]	0.52	1
Q5	**Code linked**	**15/48**	**31 [20, 45]**	**37/50**	**74 [60, 84]**	**+43 [+23, +58]**	$\mathbf{4 \times 10^{-5}}$	**0.0004**
Q6	Hyperparameter selection described	12/47	26 [15, 40]	24/50	48 [35, 61]	+22 [+3, +39]	0.03	0.24
Q7	Evaluation on held-out conditions	16/48	33 [22, 47]	22/49	45 [32, 59]	+12 [−8, +30]	0.30	1
Q8	Statistical test applied	3/48	6 [2, 17]	6/50	12 [6, 24]	+6 [−7, +18]	0.49	1
X2	**Uncertainty reported**	**30/48**	**63 [48, 75]**	**44/50**	**88 [76, 94]**	**+26 [+8, +41]**	**0.005**	**0.04**
X3	Only the best runs kept	9/48	19 [10, 32]	7/50	14 [7, 26]	−5 [−20, +10]	0.59	1
X4	Baselines tuned with the same care	6/48	13 [6, 25]	8/49	16 [9, 29]	+4 [−11, +18]	0.77	1
X7	**Compute quantified**	**13/48**	**27 [17, 41]**	**29/50**	**58 [44, 71]**	**+31 [+11, +47]**	**0.002**	**0.02**

Read with the same grid, four practices clearly rose between 2018 and 2026

Share of papers meeting each criterion, read with the same grids by the same readers: 48 papers of 2018, 50 of classical RL in 2026. On the right, the gap and its 95% Newcombe interval.



Interval clear of zero without passing the Holm correction: Q6 and Y12 (pale green). For X3, a fall is progress. Q2 describes the method, not the reporting.

Data: study/extension_stats.json, Newcombe difference, Fisher's exact test, Holm correction

Deraison.ai

Figure 4.1. All the reading criteria, grid v1 (Q, X) and grid v2 (Y), in our reading of 2018 (48 papers with experiments) and in our 50 classical RL papers from 2026, read with the same grids by the same readers. For each criterion, the two dots give the two shares, the bar the 95% Newcombe interval of their gap. The criteria whose gap passes the Holm correction are highlighted. For X3, a lower share is progress. Data: `study/extension_stats.json`.

Table 4.2. Grid v2 (criteria Y1 to Y12, deviations 11 and 14 of `study/PROTOCOL.md`, written on 9 November 2026 before any v2 reading), on the same samples as Table 4.1: 50 papers from 2018 read by us and 50 classical RL papers from 2026. Same columns: k/n over the papers where the criterion applies, share with its 95% Wilson interval, gap in points with its 95% Newcombe interval, Fisher exact test, Holm correction over the twelve criteria of grid v2 (a family separate from grid v1). Y3 is 'yes' when the main conclusion rests on at least two tasks; Y2 and Y11 apply only to papers that report an uncertainty, Y4 only to papers with several tasks. **In bold**, the criteria whose gap passes the Holm correction: Y9 alone. Harmonised values (`study/overrides_v2.json`). Source: `study/extension_stats.json`.

		Our reading of 2018		Classical RL 2026		Gap		
Code	Criterion	k/n	% [Wilson]	k/n	% [Wilson]	[Newcombe]	p	Holm p
Y1	Performance measure defined	38/48	79 [66, 88]	39/50	78 [65, 87]	−1 [−17, +15]	1	1
Y2	Error bars defined	25/30	83 [66, 93]	38/44	86 [73, 94]	+3 [−13, +21]	0.75	1
Y3	Several tasks (at least 2)	43/48	90 [78, 95]	41/50	82 [69, 90]	−8 [−22, +7]	0.39	1
Y4	A task where the method loses shown	34/43	79 [65, 89]	38/41	93 [81, 97]	+14 [−2, +29]	0.12	1
Y5	Origin of the baselines' code	22/48	46 [33, 60]	20/49	41 [28, 55]	−5 [−24, +14]	0.68	1
Y6	Exclusions stated	14/48	29 [18, 43]	16/50	32 [21, 46]	+3 [−15, +20]	0.83	1
Y7	Benchmark versions given	11/48	23 [13, 37]	13/50	26 [16, 40]	+3 [−14, +20]	0.82	1
Y8	Generalisation test	11/48	23 [13, 37]	17/50	34 [22, 48]	+11 [−7, +28]	0.27	1
Y9	**Sensitivity to a hyperparameter**	**21/48**	**44 [31, 58]**	**40/50**	**80 [67, 89]**	**+36 [+17, +52]**	**0.0003**	**0.004**
Y10	Library versions	1/48	2 [0, 11]	4/50	8 [3, 19]	+6 [−4, +17]	0.36	1
Y11	Type of uncertainty named	25/30	83 [66, 93]	41/44	93 [82, 98]	+10 [−5, +27]	0.26	1
Y12	Post-2018 statistical tool	1/48	2 [0, 11]	7/50	14 [7, 26]	+12 [+1, +24]	0.06	0.66

population. The gap says that classical RL in 2026 is almost entirely deep RL, which it was not quite in 2018. The criterion measures no flaw.

Two intervals that avoid zero without passing the correction. How the hyperparameters were chosen (Q6) goes from 26% (12 of 47) to 48% (24 of 50): $+22$ points, $[+3, +39]$, $p = 0.03$ before correction, $p_{\text{Holm}} = 0.24$. Statistical tools later than 2018 (Y12) go from 2% (1 of 48) to 14% (7 of 50): $+12$ points, $[+1, +24]$, $p_{\text{Holm}} = 0.66$. Taken alone, these two intervals avoid zero. With 11 and 12 criteria tested at once, they do not pass the adjusted threshold. We therefore detect no change after correction, without being able to rule one out.

No detectable change for the other criteria. The 16 other criteria have intervals that contain zero. Among those that measure a flaw: the method's hyperparameters (Q3, 55% then 72%), the baselines' hyperparameters (Q4, 35% then 29%), baselines tuned with the same care (X4, 13% then 16%), the measure defined (Y1, 79% then 78%), the best runs kept (X3, 19% then 14%), the statistical test (Q8, 6% then 12%), the origin of the baselines' code (Y5, 46% then 41%), held-out evaluation (Q7, 33% then 45%), the generalisation test (Y8, 23% then 34%), the benchmark versions (Y7, 23% then 26%), the library versions (Y10, 2% then 8%).

The power of the survey. Each Newcombe interval spans twenty to forty points. With 50 papers on each side, a change of ten points can still go unnoticed. 'No detectable change' does not mean 'no change'. For Q3, for instance, the observed gap is $+17$ points,

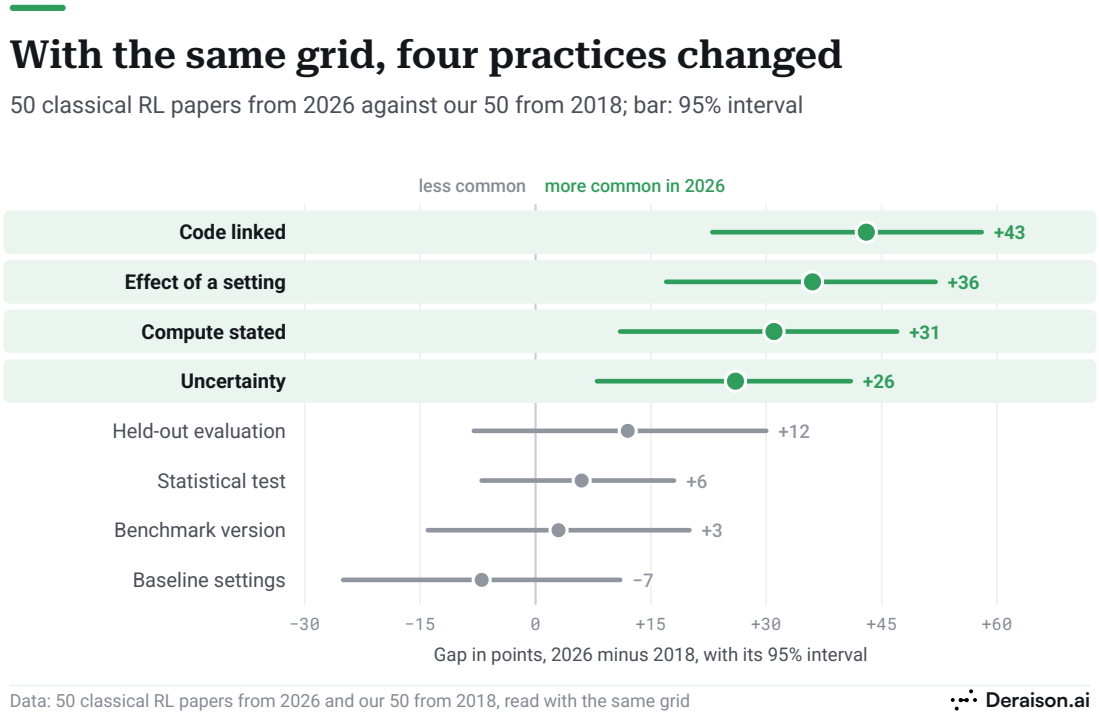


Figure 4.2. Gap, in percentage points, between 50 classical RL papers from 2026 and 50 papers from 2018 (48 with experiments), read with the same grids by the same readers, for the eight practices of the blog article. The dot gives the observed gap, the bar its 95% Newcombe interval. The four practices in green pass the Holm correction: linking the code (Q5), showing the effect of a setting (Y9), quantifying the compute (X7), reporting an uncertainty (X2). The four others, the statistical test (Q8) among them, have intervals that contain zero. Data: `study/extension_stats.json`.

with an interval from -2 to $+34$: the data are compatible with a rise of thirty points as well as with stagnation.

4.3 Rereading flaw by flaw

This section takes up the results in the order of the flaws of Table 2.2. The 2018 and 2026 figures are those of the main comparison, unless the first draw is mentioned.

F1, the code. Linking the code goes from 31% to 74% of papers. The flaw is heeded. The official report of the NeurIPS 2019 reproducibility programme points the same way: the share of accepted papers with code rose from under 50% to 74.4% (Pineau et al. 2021). Across the whole first draw, large language models included, the share is 67% (30 of 45). One classical RL paper in four still publishes its result without its code.

F2 and F3, the plan for rerunning and the hardware. The compute is quantified in 58% of 2026 papers against 27% in 2018. All of the method's hyperparameters are given in 72% of papers against 55%, with no detectable change ($p_{\text{Holm}} = 0.58$). Library versions are still rare: 4 classical RL papers out of 50 give them in 2026, against 1 out of 48 in 2018 (Y10). Across the whole first draw, one paper out of 45 gives them. Both flaws are partly heeded: compute is quantified, software versions are left unstated.

F4 and F5, the method's hyperparameters and how they were chosen. The level of Q3 shows no detectable change. How the hyperparameters were chosen is described in 48% of 2026 papers against 26% in 2018, a gap that does not pass the correction. Even in 2026, fewer than one paper in two says how it found its settings.

F6, the sensitivity to settings. At least one hyperparameter of the method is swept, with the result shown, in 80% of 2026 papers against 44% in 2018. This flaw is heeded too. The first draw, large models included, gives a similar level: 32 papers out of 45.

F7, the baselines. All the baselines' hyperparameters are given in 29% of 2026 papers (14 of 49), against 35% in 2018 (17 of 48). The papers that say they tuned their baselines with the same care as their method go from 13% to 16%. The practice is rare in both years, with no detectable change. Against Pineau's 60%, the decline looked sharp (Section 4.5). This apparent decline comes partly from our grid, which finds 35% in 2018, partly from the papers on language models, which rarely describe their baselines: the whole first draw falls to 18% (8 of 44).

F8, the performance measure. The measure is defined in 78% of 2026 papers against 79% in 2018 (Y1). When error bars appear, their meaning is defined in 38 papers out of 44 in 2026 against 25 out of 30 in 2018 (Y2). The measure is usually defined, in both years, with no detectable change. Y1 is our most fragile criterion: on the first draw, two readers agreed on only 60% of papers before its rules were clarified (Section 4.8).

F9, the number of runs. Of the 41 experimental RL papers of the first draw (`llm` and `classic` families), 20 do not say how many independent training runs support their results (X1). Four have only one. The median of the others is five, the most frequent value in the table Pineau showed in 2018. The flaw is not heeded. Chapter 6 measures with experiment E5 what five random seeds can settle.

F10, the best runs. The selection of the best runs shows in 14% of 2026 papers against 19% in 2018 (X3), with no detectable change. In the first draw, 2 papers out of 45 let it show. A paper that keeps its best runs without saying or showing it escapes our reading: these shares are lower bounds.

F12, the uncertainty and the test. The flaw splits in two. Reporting an uncertainty has become common: 88% of 2026 papers against 63% in 2018. The type of uncertainty is named in 41 of the 44 papers from 2026 that report one, against 25 of 30 in 2018 (Y11). The statistical test shows no detectable change: 12% of 2026 papers against 6% in 2018 (Q8). The tools that appeared after 2018, such as the IQM or the rliable library, go from 2% to 14% (Y12), without passing the correction. Papers draw error bars, but they rarely test.

F13 and F14, the tasks and the implementations. The main conclusion rests on several tasks in 41 papers out of 50 in 2026, against 43 out of 48 in 2018 (Y3). Among the papers with several tasks, 38 out of 41 show at least one task where their method does not win, against 34 out of 43 in 2018 (Y4). The origin of the baselines' code is stated in 41% of 2026 papers against 46% in 2018 (Y5). The three criteria show no detectable change. The first two practices were already common in 2018, the third is still a minority practice. In the first draw, only 25 papers out of 45 have several tasks: many papers on language models train on one dataset.

F15, held-out evaluation. With the same grid on both sides, classical RL goes from 33% to 45% for Q7 and from 23% to 34% for the stricter generalisation test (Y8). Both intervals contain zero. The strong apparent rise against Pineau's 10% comes from the grid and from the change of terrain (Sections 4.5 and 4.6).

F16, the benchmarks. The version of each benchmark is given in 26% of 2026 papers against 23% in 2018 (Y7), with no detectable change. In the first draw, 16 papers out of the 27 of the `llm` family try only one family of language models (X6).

4.4 What withstands the robustness analyses

Table 4.3 gathers each alternative analysis and states, for each verdict, whether the verdict survives it. The paragraphs below give the detail.

The provenance of the 2026 papers (a1). Within 2026, the 14 papers of the first draw and the 36 of the A3 queue differ on none of the four criteria (Fisher, $p = 0.73$ for Q5, 1 for Y9 and X7, 0.17 for X2). Redone with each provenance alone against 2018, the comparison keeps its sign and an interval that excludes zero in all eight cases. For Q5: $+47$ points $[+18, +65]$ with the first draw, $+41$ $[+20, +57]$ with the queue. For X2, the interval of the queue alone comes close to zero: $+21$ points $[+1, +38]$.

Composition by conference (a3). The Cochran-Mantel-Haenszel test, stratified by conference, keeps all four verdicts: $p = 5 \times 10^{-5}$ for Q5, 0.001 for Y9, 0.005 for X7, 0.006 for X2. It supports neither Q6 ($p = 0.06$) nor Y12 ($p = 0.15$).

The PDF source (a2). This cut sets 45 papers from 2026 read on arXiv against 5 read on a conference site. It settles nothing. The report does not rely on it. The difference of source does limit the comparison: an arXiv version is sometimes longer or more recent than the proceedings version, while the 2018 papers were read in the proceedings.

One reader removed at a time (a4). Removing each of the sixteen primary readers in turn makes none of the four verdicts fragile. The weakest lower bound of the Newcombe interval is $+16$ points for Q5 (without H1 or without H3), $+10$ for Y9 (without H2), $+7$ for X7 (without E, K2 or K5) and $+4$ for X2 (without H4). Q6 becomes fragile without reader H2 ($+18$ points, $[-3, +36]$). Y12 is fragile in 8 of the 16 cuts.

Without the thirteen papers read partly by another model. Removing the thirteen papers finished by `claude-opus-5` (deviation 29) leaves 48 classical RL papers in 2026 and the 50 papers of 2018. Code (Q5, $p_{\text{Holm}} = 0.0001$), sensitivity (Y9, 0.008) and compute (X7, 0.04) still pass the correction. Reported uncertainty no longer passes it: $+25$ points, $[+8, +41]$, $p_{\text{Holm}} = 0.07$. Neither does Q2 ($p_{\text{Holm}} = 0.07$). The trends of the trajectory do not change (Section 4.7).

The search against the 'no' decisions. On the seven criteria checked, the two samples hold 377 final no decisions. For 94 of them, the patterns fixed in advance find no match in the text. The other 283 went to the arbiter. The arbiter turns 10 of them: 2 in 2018 (C19 for Q5, C120 for Y9) and 8 in 2026 (K058, K063 and LR03 for Q4, K002 for Q7, ML06 for Q8, K056, LR03 and LR09 for X7). The arbiter judges 24 of them doubtful. Redone

Table 4.3. Robustness checks of the survey, all exploratory (deviations 21, 27 and 29 of `study/PROTOCOL.md`): they say whether a verdict is fragile, without changing any. Columns: the four criteria that changed (Q5, Y9, X7, X2) and the two whose interval avoids zero without passing Holm (Q6, Y12). Gaps in points (classical RL 2026 minus our reading of 2018), 95% Newcombe interval, Fisher test, Holm over the criterion's grid family as in the main comparison. (a1) classical RL 2026 split by provenance (first draw with no title screen, A3 queue with the screen), then Fisher p between the two provenances within 2026; (a2) Fisher p between 2026 papers read on arXiv (45) and on a conference site (5), a cut too unbalanced to settle anything (deviation 21 c); (a3) Cochran-Mantel-Haenszel test stratified by conference (Haldane correction); (a4) comparison redone without each primary reader in turn: range of the gaps and lowest lower bound; a criterion is fragile when the gap changes sign or the interval contains zero with at least 35 papers on each side. Bottom part: without the thirteen papers read partly by `claude-opus-5` (deviation 29); with the 10 'no' decisions a blind arbiter turned after the keyword search, then with the 24 decisions judged doubtful as well, out of 377 'no' decisions examined (deviation 27). **In bold**, gaps whose interval excludes zero and p below 0.05 (except the homogeneity p of (a1) and (a2), where a high p is expected). The thesis 'heeded where it cost little' is rejected by the blind test (b): Spearman correlation between the observed gap and the median cost scored by three raters +0.16 (permutation p 0.52), with the risk −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`.

Check	Q5	Y9	X7	X2	Q6	Y12
Main comparison: gap [CI]	**+43 [+23, +58]**	**+36 [+17, +52]**	**+31 [+11, +47]**	**+26 [+8, +41]**	**+22 [+3, +39]**	**+12 [+1, +24]**
Holm p	**0.0004**	**0.004**	**0.02**	**0.04**	0.24	0.66
(a1) first draw only (14)	**+47 [+18, +65]**	**+35 [+5, +54]**	**+30 [+2, +54]**	**+38 [+13, +52]**	+17 [−8, +44]	+12 [−1, +38]
(a1) A3 queue only (36)	**+41 [+19, +57]**	**+37 [+16, +53]**	**+31 [+10, +49]**	**+21 [+1, +37]**	**+24 [+4, +43]**	**+12 [0, +27]**
(a1) first draw against A3: p	0.73	1	1	0.17	0.76	1
(a2) arXiv against site: p	1	1	0.15	1	1	1
(a3) Mantel-Haenszel: odds ratio	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) one reader removed: gaps	+37 to +50	+30 to +43	+27 to +37	+21 to +31	+18 to +25	+9 to +14
(a4) lowest CI bound	**+16**	**+10**	**+7**	**+4**	−3	−2
Fragile under (a)	no	no	no	no	yes (1)	yes (8)
Without the 13 opus-5 papers	**+46 [+26, +61]**	**+35 [+16, +51]**	**+31 [+11, +48]**	**+25 [+8, +41]**	**+22 [+3, +40]**	**+13 [+1, +25]**
Holm p	**0.0001**	**0.008**	**0.04**	0.07	0.23	0.65
Refuted 'no' turned: Holm p	**0.001**	**0.008**	**0.003**	**0.04**	0.24	0.66
doubtful ones too: Holm p	**0.002**	0.05	**0.005**	0.07	0.24	0.66

with the turned decisions, the main comparison keeps all its verdicts (`study/falsify_no.json`). The gap in quantified compute grows, to $+37$ points $[+17, +53]$. The gap in the baselines' hyperparameters falls to -1 point. If the 24 doubtful decisions are turned as well, reported uncertainty and sensitivity move just above the threshold ($p_{\text{Holm}} = 0.07$ and 0.054). Deviation 27 did not announce this second bound.

Robustness summary. Code and compute pass the threshold in all our checks. Sensitivity to settings passes it in all but the bound of the doubtful decisions, where it stops at $p_{\text{Holm}} = 0.054$. Reported uncertainty fails in two checks, without the thirteen papers and with the bound of the doubtful decisions, with $p_{\text{Holm}} = 0.07$ each time. The report keeps it among the four changes, with this reservation.

The cost and the risk of each practice. The three raters agree with one another: their scores correlate from 0.78 to 0.91 for cost, from 0.90 to 0.93 for risk (Spearman, per pair). Their scores do not predict the observed gaps. The Spearman correlation is $+0.16$ for cost ($p = 0.52$ by permutation) and -0.04 for risk ($p = 0.89$). Sweeping a hyperparameter (Y9), scored costly (median 4), progressed. The versions of benchmarks (Y7) and of dependencies (Y10), scored nearly free (median 1), did not move. The reading 'heeded where it cost little' is withdrawn.

4.5 Our reading of 2018 against Pineau's figures

The gap. The secondary comparison of deviation 12 sets our reading of 2018 against Pineau's figures, on the six criteria Q3 to Q8 for which she gives a comparable one, with a Holm correction over these six criteria (`extension_stats.json`, field `ours2018_vs_pineau`).

- **Q3, the method's hyperparameters**: 55% (26 of 47) in our reading, 90% in Pineau's. Gap of -35 points, $[-50, -17]$, $p_{\text{Holm}} = 0.001$.
- **Q7, held-out evaluation**: 33% (16 of 48) in our reading, 10% in Pineau's. Gap of $+23$ points, $[+7, +39]$, $p_{\text{Holm}} = 0.03$.
- **Q4, the baselines' hyperparameters**: 35% against 60%. Gap of -25 points, $[-42, -5]$, $p_{\text{Holm}} = 0.06$.
- **Q5, code linked**: 31% against 55%. Gap of -25 points, $[-42, -5]$, $p_{\text{Holm}} = 0.06$.
- **Q6 and Q8**: 26% against 20% and 6% against 5%, with no detectable gap.

The first 12 papers of the calibration already gave the same sign: Q3 at 6 of 10, Q7 at 4 of 11 (`results_final.json`). Our grid is stricter than Pineau's on what the paper gives, more lenient on what counts as held out.

Why our reading differs. Pineau's draw and her way of settling doubtful cases remain unknown: the slide announces 50 RL papers from 2018 published at NeurIPS, ICML and ICLR, without saying how they were chosen or how each criterion was decided. We therefore cannot attribute the gap to one cause. Three mechanisms make it plausible, none of them demonstrated.

- **Stricter written rules on what is given.** Our rules for Q3 and Q4 require the hyperparameters of all the main methods. 'Default values' without a list count

Table 4.4. Pineau's figures (2018), as context and never as a verdict: her shares on 50 RL papers from 2018, for the six criteria she quantifies (published percentage, `study/results_final.json`; in parentheses the k/n used by the tests, her percentage times 50 and rounded, that is 56% for Q5 and 4% for Q8), set against our reading of 50 papers from 2018 with our grid, then against the 45 experimental papers of the first 2026 draw, large language models included. The first row of each criterion measures the gap between our grid and hers on the same year: our reading is stricter on Q3 and broader on Q7, which led us to base verdicts only on the same-grid comparison (deviation 20 of `study/PROTOCOL.md`, Table 4.1). The second mixes the change in practice, the change of grid and the change of population. Gap in points with the 95% Newcombe interval, Fisher exact test, Holm correction over Q3 to Q8 within each family of comparisons. **In bold**, what passes the Holm correction. Pineau's draw remains unknown. Sources: `study/extension_stats.json`, `study/results_final.json`.

Code	Criterion, then comparison	k/n	%	Pineau 2018	Gap [Newcombe]	p	Holm p
Q3	All of the method's hyperparameters						
	our reading of 2018	26/47	55	90 (45/50)	**−35 [−50, −17]**	**0.0002**	**0.001**
	first 2026 draw	30/45	67		**−23 [−39, −7]**	**0.006**	**0.02**
Q4	All of the baselines' hyperparameters						
	our reading of 2018	17/48	35	60 (30/50)	−25 [−42, −5]	0.02	0.06
	first 2026 draw	8/44	18		**−42 [−57, −22]**	**5×10^{-5}**	**0.0002**
Q5	Code linked						
	our reading of 2018	15/48	31	55 (28/50)	−25 [−42, −5]	0.02	0.06
	first 2026 draw	30/45	67		+11 [−9, +29]	0.30	0.91
Q6	Hyperparameter selection described						
	our reading of 2018	12/47	26	20 (10/50)	+6 [−11, +22]	0.63	1
	first 2026 draw	12/45	27		+7 [−10, +23]	0.47	0.95
Q7	Evaluation on held-out conditions						
	our reading of 2018	16/48	33	10 (5/50)	**+23 [+7, +39]**	**0.006**	**0.03**
	first 2026 draw	31/44	70		**+60 [+42, +73]**	**1×10^{-9}**	**6×10^{-9}**
Q8	Statistical test applied						
	our reading of 2018	3/48	6	5 (2/50)	+2 [−8, +13]	0.67	1
	first 2026 draw	2/45	4		0 [−10, +11]	1	1

only if the library version is given. For Q5, 'will be released' and a project page that does not lead to the code count as no. A reader who accepts a partial table or a promise of release finds higher shares.

- **A more lenient rule on what is held out.** Our Q7 counts environment random seeds unseen in training. Pineau probably had a sharper separation in mind, since her NeurIPS roadmap asks for separate tasks and for richer simulators.
- **Another draw.** Our 2018 sample is drawn at random from the 131 papers whose title matches our expression. Pineau's sample may contain papers our filter does not see, or exclude some.

After correction, the gaps for Q4 and Q5 remain compatible with chance ($p_{\text{Holm}} = 0.06$). Those for Q3 and Q7 do not.

The verdict rule that follows. Comparing our 2026 papers with Pineau's figures would mix the change in practice with the change of grid. Deviation 20 draws the verdict rule from this (Section 3.8). Pineau's figures stay on display as context, in Section 4.6.

4.6 The first draw and the language models

The first draw against Pineau. Figure 4.3 compares the 45 papers with experiments of the first draw, large language models included, with Pineau's figures. Figure 4.4 draws the gaps with their Newcombe intervals, Figure 4.5 the 2026 shares with their Wilson intervals. Three intervals stand clearly away from zero. The method's hyperparameters decline, from 90% to 67% (-23 points, $[-39, -7]$, $p_{\text{Holm}} = 0.02$). Those of the baselines decline, from 60% to 18% (-42 points, $[-57, -22]$, $p_{\text{Holm}} = 0.0002$). Held-out evaluation rises, from 10% to 70% ($+60$ points, $[+42, +73]$). Code goes from 55% to 67% with no detectable gap. The statistical test goes from 5% to 4%. These gaps mix two changes, the change in practice and the change of grid. They serve as context, never as a verdict.

The papers on language models. Table 4.5 sets, within the first draw, the 27 papers of the `llm` family against the 14 classical RL papers. The differences are large. Held-out evaluation is almost universal in the `llm` family (26 of 27) and a minority practice in the classical RL of the first draw (5 of 13): collections of mathematics problems separate training and test problems by construction. Uncertainty is rare in the `llm` family (4 of 27) and universal in the classical RL of the first draw (14 of 14). The baselines' hyperparameters are all given in 5 `llm` papers out of 27. Sixteen `llm` papers out of 27 try only one family of base models. The apparent rise of held-out evaluation against Pineau comes almost entirely from these papers. So does the apparent decline of uncertainty in the first draw.

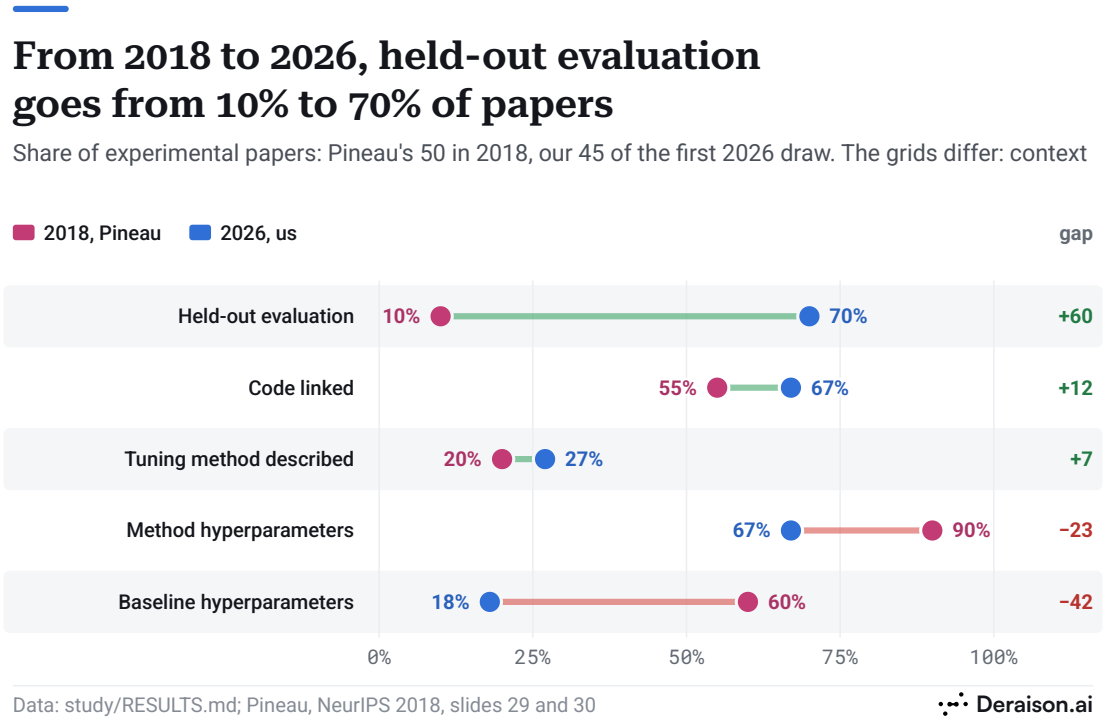


Figure 4.3. Pineau's grid, applied in 2018 and in 2026, as context. Each row is a reading criterion. The 2018 dot gives the share of the 50 papers read by Pineau that meet the criterion. The 2026 dot gives the share among the 45 papers with experiments of our first draw from ICLR 2026, ICML 2026 and NeurIPS 2025, large language models included. The two dots do not come from the same grid: our reading of 2018 finds 55% for the method's hyperparameters, where Pineau found 90%. The statistical test, left out of the drawing, goes from 5% to 4%. Data: `study/audit_stats.json`.

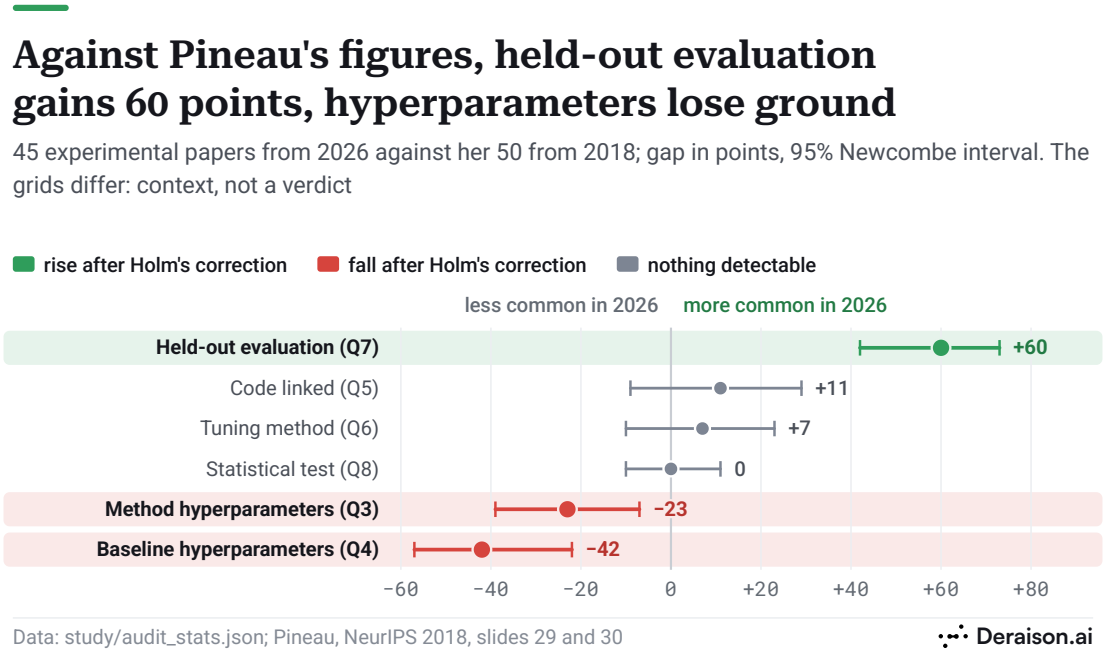


Figure 4.4. Gap between the share of the 45 papers with experiments of our first 2026 draw and that of the 50 papers from 2018 read by Pineau, criterion by criterion, in percentage points. The dot is the observed gap, the bar its 95% Newcombe interval, which accounts for the uncertainty of both samples. Above zero, the practice is more frequent in 2026. The gap mixes the change in practice with the difference between our grid and Pineau's: it serves as context. The verdicts come from the same-grid comparison. Data: `study/audit_stats.json`.

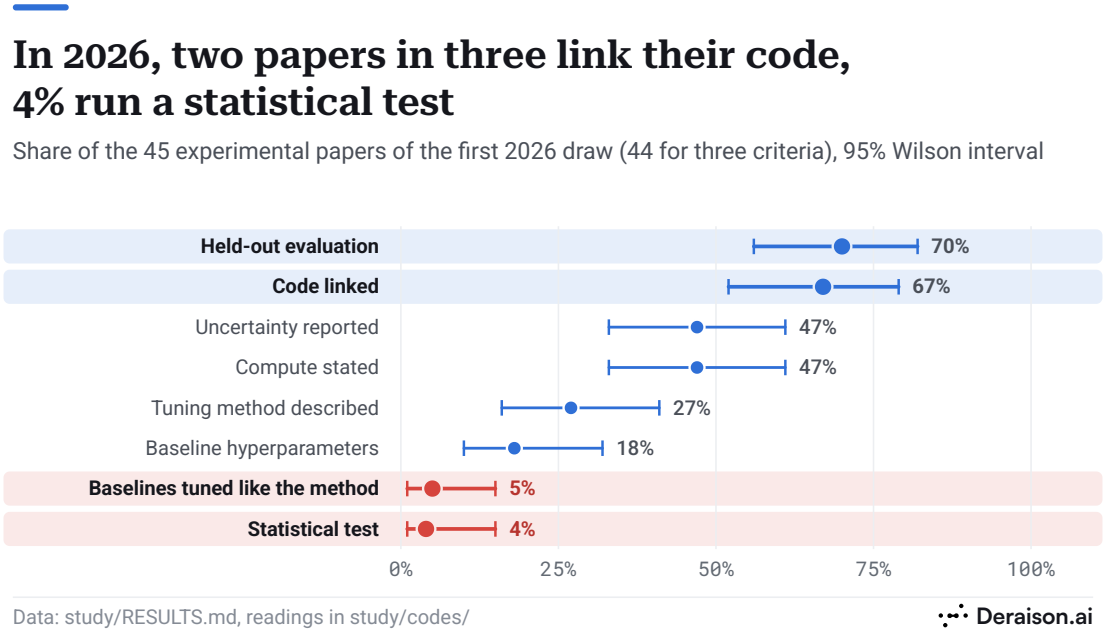


Figure 4.5. Share of the 45 papers with experiments of the first 2026 draw that meet each criterion, with its 95% Wilson interval. The draw mixes classical RL with papers on large language models. With a sample of this size, a gap of less than fifteen points from 2018 cannot be told apart from the luck of the draw. Data: `study/results_final.json`.

Table 4.5. RL applied to large language models (family `llm` of criterion X5) against classical RL, in the first random draw of 50 papers from 2026 (ICLR 2026, ICML 2026, NeurIPS 2025): 27 `llm` papers, 14 `classic`, 9 `theory` (not shown, counted in 'All'). k/n over the papers with experiments where the criterion applies, share in percent in parentheses; 95% Wilson interval for the 'All' column, the only one the script computes. Only grid v1 has this split by family in the results (`study/audit_stats.json`, harmonisation of `study/overrides.json`): grid v2 is not broken down by family, and no test between families was preregistered; the table is descriptive. The last two columns recall the classical sample enlarged to 50 papers (Table 4.1). The 18 `llm` papers and 3 `theory` papers read in the A3 queue stay apart: the title screen had already removed many of them, so they do not form a sample. Sources: `study/audit_stats.json`, `study/extension_stats.json`.

		First 2026 draw (grid v1)				Classical RL 2026, enlarged	
Code	Criterion	RL on large models	Classical RL	All	% all [Wilson]	k/n	%
Q3	All of the method's hyperparameters	17/27 (63)	11/14 (79)	30/45	67 [52, 79]	36/50	72
Q4	All of the baselines' hyperparameters	5/27 (19)	2/14 (14)	8/44	18 [10, 32]	14/49	29
Q5	Code linked	19/27 (70)	11/14 (79)	30/45	67 [52, 79]	37/50	74
Q6	Hyperparameter selection described	6/27 (22)	6/14 (43)	12/45	27 [16, 41]	24/50	48
Q7	Evaluation on held-out conditions	26/27 (96)	5/13 (38)	31/44	70 [56, 82]	22/49	45
Q8	Statistical test applied	2/27 (7)	0/14 (0)	2/45	4 [1, 15]	6/50	12
X2	Uncertainty reported	4/27 (15)	14/14 (100)	21/45	47 [33, 61]	44/50	88
X3	Only the best runs kept	1/27 (4)	1/14 (7)	2/45	4 [1, 15]	7/50	14
X4	Baselines tuned with the same care	0/27 (0)	2/14 (14)	2/44	5 [1, 15]	8/49	16
X7	Compute quantified	12/27 (44)	8/14 (57)	21/45	47 [33, 61]	29/50	58

A change of population. The `llm` family did not exist in 2018. A gap between the first 2026 draw and 2018 that runs through these papers first measures the arrival of a new population, with its own habits. It says nothing about the change in practice within a population that stayed comparable. The verdicts of this report therefore hold for classical RL alone.

4.7 The trajectory from 2018 to 2026

The shares year by year. Table 4.6 and Figure 4.6 give, for each practice, the share of classical RL papers that follow it in 2018 (48 papers), in 2019, 2020, 2022 and 2024 (20 papers per year, ICML and NeurIPS) and in 2026 (50 papers). Code linked goes from 31% in 2018 to 60%, 50%, 70% and 75% from 2019 to 2024, then 74% in 2026. Sensitivity to a hyperparameter is at 44%, then 35%, 45%, 60%, 85% and 80%. Compute quantified is at 27%, then 30%, 40%, 40%, 55% and 58%. Reported uncertainty is at 63%, then 80%, 85%, 95%, 100% and 88%. The statistical test is at 6%, then 5%, 10%, 10%, 25% and 12%. With 20 papers, a share is uncertain by about twenty points either way.

The trends. The Cochran-Armitage test, with the years as scores, gives a clear trend for the four practices after the Holm correction over the five tested criteria: $z = 4.29$ for code ($p_{\text{Holm}} = 7 \times 10^{-5}$), $z = 4.78$ for sensitivity (9×10^{-6}), $z = 3.44$ for compute (0.002), $z = 3.39$ for uncertainty (0.002). The statistical test shows no trend ($z = 1.53$, $p = 0.13$).

Table 4.6. Trajectory of classical RL from 2018 to 2026 (deviation 25 of `study/PROTOCOL.md`, written before any draw): share of papers that meet each criterion, in percent, by year. 2018: our reading of the papers with experiments; 2019, 2020, 2022 and 2024: the first 20 experimental classical RL papers per year, drawn at random from ICML and NeurIPS (ICLR is missing, its site refusing automated downloads); 2026: the classical sample of the main comparison. The number in parentheses gives the denominator when it differs from the column size (criterion not applicable to some papers). With 20 papers, a share carries an uncertainty of about twenty points either way. For the five tested criteria (Q5, Y9, X7, X2, Q8): Cochran-Armitage trend test over the six years (scores equal to the years), two-sided p and Holm-corrected p over the five; the six other rows are descriptive. Bottom part: immediate gap (2019 and 2020 pooled against 2018) and later gap (2022 and 2024 against 2019 and 2020), in points, 95% Newcombe interval, Holm over the five criteria within each family. **In bold**, what passes the Holm correction. Source: `study/trajectory.json` (`study/trajectory.py`).

Code	Criterion	2018	2019	2020	2022	2024	2026	trend p	Holm p
	n	48	20	20	20	20	50		
Q5	**Code linked**	**31**	**60**	**50**	**70**	**75**	**74**	**2×10^{-5}**	**7×10^{-5}**
Y9	**Sensitivity to a hyperparameter**	**44**	**35**	**45**	**60**	**85**	**80**	**2×10^{-6}**	**9×10^{-6}**
X7	**Compute quantified**	**27**	**30**	**40**	**40**	**55**	**58**	**0.0006**	**0.002**
X2	**Uncertainty reported**	**63**	**80**	**85**	**95**	**100**	**88**	**0.0007**	**0.002**
Q8	Statistical test applied	6	5	10	10	25	12	0.13	0.13
Q3	All of the method's hyperparameters	55 (47)	45	55	70	55	72		
Q4	All of the baselines' hyperparameters	35	16 (19)	45	25	40	29 (49)		
Q6	Hyperparameter selection described	26 (47)	30	40	40	35	48		
Q7	Evaluation on held-out conditions	33	25	20	20	55	45 (49)		
X4	Baselines tuned with the same care	13	26 (19)	35	15	25	16 (49)		
Y7	Benchmark versions given	23	25	20	40	20	26		

Gaps between periods (Newcombe, Fisher, Holm over the five tested criteria)

Code	Criterion	2019-2020 against 2018 gap [CI]	Holm p	2022-2024 against 2019-2020 gap [CI]	Holm p
Q5	Code linked	+24 [+3, +42]	0.16	+17 [−3, +36]	0.49
Y9	Sensitivity to a hyperparameter	−4 [−23, +16]	1	**+32 [+11, +50]**	**0.03**
X7	Compute quantified	+8 [−11, +27]	1	+13 [−9, +32]	0.62
X2	Uncertainty reported	+20 [+1, +37]	0.23	+15 [+1, +30]	0.23
Q8	Statistical test applied	+1 [−10, +14]	1	+10 [−5, +25]	0.62

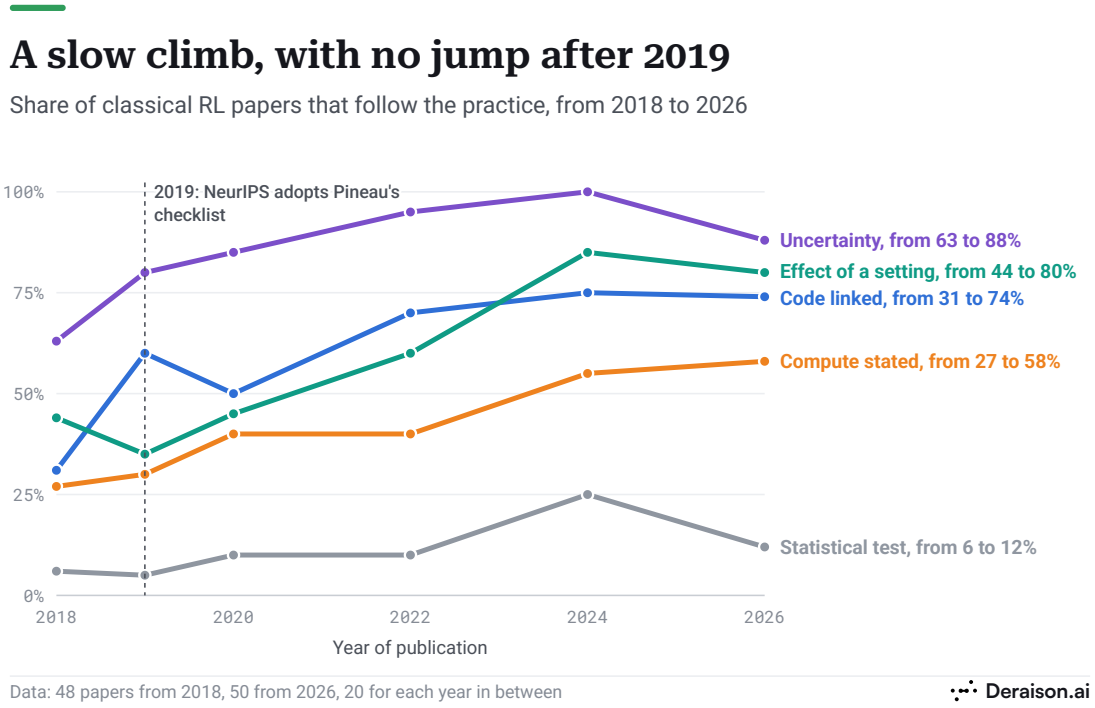


Figure 4.6. Share of classical RL papers that follow each practice, year by year: 48 papers from 2018 (those with experiments), 20 per year for 2019, 2020, 2022 and 2024 drawn from ICML and NeurIPS, 50 from 2026. The dotted vertical line marks the NeurIPS checklist (2019). Code linked, sensitivity to a hyperparameter, compute quantified and reported uncertainty climb steadily. The statistical test, in grey, shows no trend. With 20 papers, a share is uncertain by about twenty points either way. Data: `study/trajectory.json`.

No jump right after 2019. From 2018 to 2019 and 2020 pooled, no gap passes the correction. Code rises by $+24$ points, $[+3, +42]$, with $p = 0.03$ before correction and $p_{\text{Holm}} = 0.16$ after. Uncertainty rises by $+20$ points, $[+1, +37]$, $p_{\text{Holm}} = 0.23$. Sensitivity does not rise (-4 points). From 2019 and 2020 pooled to 2022 and 2024 pooled, only sensitivity rises clearly, from 40% to 73%, with an interval from $+11$ to $+50$ points and $p_{\text{Holm}} = 0.03$. These series describe a slow rise that began before the checklist or with it. They do not say what caused it. Coakley et al. (2026) likewise see the share of released code rise before the checklists, across five artificial intelligence conferences.

Without the thirteen papers. With the papers finished by `claude-opus-5` removed, the four trends still pass the correction ($p_{\text{Holm}} < 0.004$) and the statistical test still shows none. The later rise of sensitivity keeps $p_{\text{Holm}} = 0.02$ (`trajectory_sans_opus5.json`).

The statistical test's peak in 2024. The statistical test reaches 25% of papers in 2024 (5 of 20), then falls back to 12% in 2026 (6 of 50). With 20 papers, this peak stays within the margin of uncertainty. We do not read it as a fact.

4.8 Agreement between readers

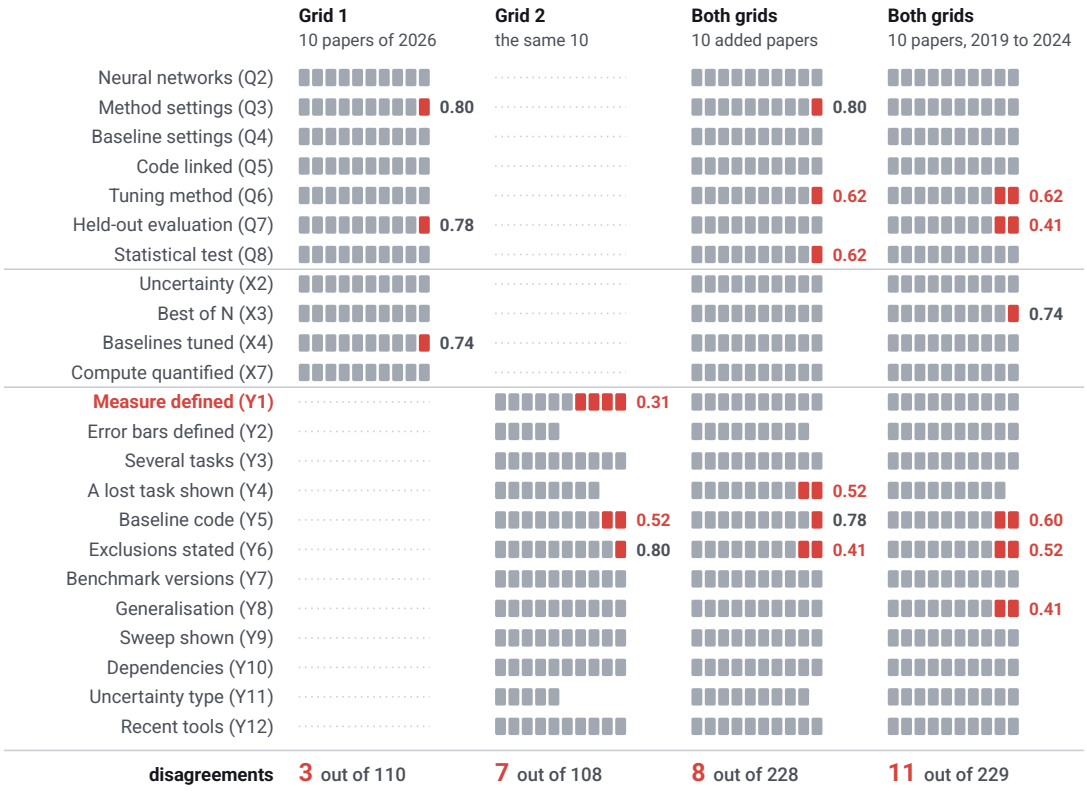
Table 4.7 and Figure 4.7 give the agreement of the blind double readings, criterion by criterion, on the raw readings before harmonisation.

Table 4.7. Agreement between two blind readings by language model agents, on the raw readings, before any harmonisation. G: ten papers of the first 2026 draw reread with grid v1 (127 identical decisions out of 130, `study/audit_stats.json`); G2: the same ten papers reread with grid v2; R2: ten papers drawn at random from the added papers (`random.Random(7)`, deviation 12), reread with both grids; R3: ten papers of the 2019 to 2024 trajectory (`random.Random(2030)`, deviation 26). Agree: identical decisions over the papers where both readers answered yes or no (Y3 turned into yes from two tasks); κ: Cohen's kappa, blank when one of the two readers does not vary (trivial agreement). An empty cell outside κ: criterion absent from that double reading. **In bold**, the only criterion under 70% agreement, flagged as fragile by the script: Y1, performance measure, before its rules were made more precise (deviations 17 and 19). All readers are copies of the same model (deviation 28): agreement measures that model's stability, not that of two independent readers. Sources: `study/agree_v2.json`, `study/audit_stats.json`.

		G 2026 draw, v1		G2 2026 draw, v2		R2 added papers		R3 trajectory	
Code	Criterion	agree	κ	agree	κ	agree	κ	agree	κ
Q1	The paper has experiments	10/10							
Q2	Neural networks used	10/10	1.00			10/10	1.00	10/10	
Q3	All of the method's hyperparameters	9/10	0.80			9/10	0.80	10/10	1.00
Q4	All of the baselines' hyperparameters	10/10	1.00			10/10	1.00	10/10	1.00
Q5	Code linked	10/10	1.00			10/10	1.00	10/10	1.00
Q6	Hyperparameter selection described	10/10	1.00			9/10	0.62	8/10	0.62
Q7	Evaluation on held-out conditions	9/10	0.78			10/10	1.00	8/10	0.41
Q8	Statistical test applied	10/10	1.00			9/10	0.62	10/10	1.00
X2	Uncertainty reported	10/10	1.00			10/10	1.00	10/10	
X3	Only the best runs kept	10/10				10/10		9/10	0.74
X4	Baselines tuned with the same care	9/10	0.74			10/10		10/10	1.00
X5	Family (llm, classic, theory)	10/10	1.00						
X7	Compute quantified	10/10	1.00			10/10	1.00	10/10	1.00
Y1	**Performance measure defined**			**6/10**	**0.31**	**10/10**	**1.00**	**10/10**	**1.00**
Y2	Error bars defined			5/5	1.00	9/9	1.00	10/10	1.00
Y3	Several tasks (at least 2)			10/10	1.00	10/10		10/10	1.00
Y4	A task where the method loses shown			8/8	1.00	8/10	0.52	9/9	1.00
Y5	Origin of the baselines' code			8/10	0.52	9/10	0.78	8/10	0.60
Y6	Exclusions stated			9/10	0.80	8/10	0.41	8/10	0.52
Y7	Benchmark versions given			10/10	1.00	10/10	1.00	10/10	1.00
Y8	Generalisation test			10/10	1.00	10/10	1.00	8/10	0.41
Y9	Sensitivity to a hyperparameter			10/10	1.00	10/10	1.00	10/10	1.00
Y10	Library versions			10/10		10/10	1.00	10/10	
Y11	Type of uncertainty named			5/5	1.00	9/9	1.00	10/10	1.00
Y12	Post-2018 statistical tool			10/10		10/10	1.00	10/10	1.00

Read twice blind, 646 decisions out of 675 agree

Four times, a second agent rereads ten papers without seeing the first reading. One square per paper read twice: grey when the two readings agree, red when they differ. The number is Cohen's kappa.



Dotted: criterion absent from that grid. Fewer than ten squares: the criterion does not apply to every paper. Kappa in red: under 0.70. Name in red: the only row under 70% agreement, the performance measure (Y1) read with grid 2.

Data: study/audit_stats.json (grid 1), study/agree_v2.json (grid 2), raw readings before harmonisation; kappa intervals not computed

Deraison.ai

Figure 4.7. Agreement between the blind double readings, criterion by criterion, on the raw readings, before harmonisation: share of identical decisions and Cohen's kappa when both readers vary. Four pairs of readings of ten papers each: G (grid v1) and G2 (grid v2) on the first 2026 draw, R2 on the added papers of 2018 and 2026, R3 on the trajectory from 2019 to 2024. The 70% agreement threshold, below which a criterion is called fragile, is marked. Data: `study/agree_v2.json`, `study/audit_stats.json`.

Grid v1 on the first draw. The second reader G and the primary readers agree on 127 decisions out of 130 (98%), that is, 13 criteria on 10 papers. The three disagreements concern Q3, Q7 and X4, with kappas of 0.80, 0.78 and 0.74. The kappa is 1 for the other criteria on which both readers vary.

Grid v2 on the first draw. On the same ten papers, G2 and the primary readers agree completely on 9 criteria out of 12. Agreement falls to 90% on exclusions (Y6, kappa 0.80), to 80% on the implementation of the baselines (Y5, kappa 0.52) and to 60% on the performance measure (Y1, kappa 0.31). Y1 is the only fragile criterion in all the double readings. Deviations 17 and 19 then clarified its rules.

The added papers. R2 rereads ten papers drawn from the additions of 2018 and 2026 with both grids. Agreement reaches at least 80% on each of the 23 criteria and is complete on 17 of them. The lowest agreements are those of Y4 (80%, kappa 0.52) and Y6 (80%, kappa 0.41).

The trajectory. R3 rereads ten papers of A4. Agreement reaches at least 80% on each of the 23 criteria and is complete on 17 of them, the five tested criteria of the trajectory among them (Q5, Y9, X7, X2, Q8). The lowest kappas are those of Q7 and Y8 (80%, kappa 0.41).

The stability of one model. All the readers are copies of the same model, apart from the thirteen papers finished on `claude-opus-5`. The agreement of the double readings therefore measures how stably this one model reads. It does not measure the agreement of two independent readers. A human might have read the same passages differently. The quotations and pages in `study/codes/` and `study/codes_v2/` let anyone check each decision.

4.9 The declarations of the papers

Statistical significance. The NeurIPS checklist question on statistical significance settles for error bars 'or other appropriate information'. Of the 30 NeurIPS 2025 papers read, 18 answer 'Yes' to it. None of the 18 applies a test: the checklist does not ask for the test and the papers do not run one. One of the 18 reports no uncertainty either. The others report an uncertainty and stop there (`study/checklist.json`).

Code, settings and compute. On code, 5 papers out of 22 answer 'Yes' without a link to their code in the version we read, or 23% [10, 43]. Four have no link even in the proceedings version. Three write in their own justification that the code will be released later. On settings, 6 papers out of 22 answer 'Yes' without giving all the hyperparameters of their method according to our rule. On compute, 6 papers out of 21 answer 'Yes' without quantifying any compute time, which the question explicitly asks for. One paper has no readable checklist in the proceedings. A ticked box does not guarantee the practice it names.

4.10 Sensitivity to the title filter

The RL papers the filter misses. Of the 400 papers drawn outside the title filter, 25 have an abstract that matches the regular expression: 6.2%, with a Wilson interval from 4.3% to 9.1% (`study/title_filter.json`). Scaled to the population outside the filter, this share gives about 1,037 missed RL papers, from 708 to 1,503, or 45% of all RL papers according to this criterion (from 36% to 55%). This figure is an upper bound: the rule is mechanical and also counts false positives, such as a paper where 'RL' stands for representation learning. Most of the papers found deal with large language models.

The effect of these papers. Read with both grids, the first 20 hidden-RL papers resemble the first 2026 draw: code linked in 60% of them (67% in the draw), an uncertainty in 50% (47%). Thirteen deal with large language models. The 7 classical RL papers are too few to test our verdicts. The filter misses many papers, above all on language models, with no sign that these papers follow different practices.

4.11 Synthesis

With the same grid on both sides, in classical RL, four practices changed between 2018 and 2026: linking the code, showing the sensitivity to a hyperparameter, quantifying the compute, reporting an uncertainty. Code and compute withstand every cut, sensitivity nearly every one. Reported uncertainty fails in two checks, with $p_{\mathrm{Holm}} = 0.07$. All four rise slowly from 2018 to 2024, with no jump right after the 2019 checklist. The statistical test, baselines described and tuned with care, held-out evaluation, benchmark versions and library versions show no detectable change. Pineau was heard on what is published next to the result, much less on how the result itself is judged. Our reading of 2018 differs from Pineau's on the method's hyperparameters and on held-out evaluation, hence the same-grid verdict rule. Chapter 10 gathers the limitations of these results. Chapter 11 sets them against the experiments.

CHAPTER 5

The experiments: protocol and methods

The survey of chapters 3 and 4 measures what papers publish. The experiments put the flaws of 2018 to the test on real training runs, made for this report. They return to the ground of Henderson et al. (2018): simulated bodies in the MuJoCo physics simulator, bodies that reinforcement learning algorithms learn to move forward. They do so with the algorithms of 2026 and with the rules Joelle Pineau asked for: enough random seeds, confidence intervals, baselines tuned with the same care, evaluation held out, code and data published.

This chapter fixes everything that comes before a result: the tools and their versions, the programs and their settings, the tasks, the budgets, the random seeds, the evaluation protocol, the file format, the statistics and the design of each experiment. It describes what the code does, line by line where that helps. It also flags every place where the code departs from the written protocol. The results come in chapter 6. Chapter 8 draws the software chain that links the runs to the documents.

5.1 A protocol written before the first training run

The protocol of the experiments is the file `sim/SIM_PROTOCOL.md`. Its first version is dated 8 November 2026. It was written before any training. It fixes the questions, the algorithms, the tasks, the budgets, the number of random seeds, the evaluation and the analyses. It also lays down three rules of conduct:

- the random seeds are fixed before launching, from 0 to 29;
- no run is removed or relaunched because of its score;
- a run that crashes is relaunched once with the same random seed, then the incident is recorded.

Every later change enters a numbered, dated list of ‘deviations’, together with a note of what we had already seen when we decided it. The protocol counts twenty of them as of 21 November 2026. Chapter 9 gives all of them, with those of the survey and of the language-model study. This chapter cites a deviation by its number when it changes a method. Section 5.12 sorts the decisions by whether they precede or follow the results they affect.

Throughout the report, a *run* is one complete training of one program on one task with a given random seed. The *main* results are the ones the protocol announced: ten random seeds per pair of program and task. The additions of deviation 12 (twenty random seeds, equal budget, replication) are given next to them. Every divergence is stated.

5.2 Tools and versions

Four frozen Python environments. Four Python 3.11.18 environments were used, all on a Mac with an Apple M2 Max chip (macOS 27.2, arm64). Each is frozen by a `pip freeze` taken on 15 November 2026, in `envs/` (table D.2, chapter D):

- `sim/env` serves training and re-evaluation (`sim/run.py`, `sim/dose.py`). It holds 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` computes every analysis: NumPy 2.4.6, SciPy 1.17.1, statsmodels 0.15.0, rliable 1.2.0 (Agarwal et al. 2021);
- `sim/env-zoo` runs the official script of RL Baselines3 Zoo 2.9.2 (Raffin 2020) for experiment E6, with the same libraries as `sim/env`;
- `sim/env-llm` serves the language-model study (chapter 7).

What each tool contributes. Gymnasium, the successor of OpenAI Gym maintained by the Farama Foundation, defines the interface of the tasks: `reset` returns a first observation, `step` applies an action and returns the next observation, the reward and two end-of-episode flags. MuJoCo computes the physics. It has been free and open since 2022, whereas it required a paid licence in 2018. Stable-Baselines3 provides PPO written in PyTorch. SBX ('Stable-Baselines Jax') rewrites PPO, SAC and CrossQ in JAX with the same interface. sb3-contrib provides TRPO. rliable computes interquartile means, their bootstrap intervals and probabilities of improvement.

A version deviation. The protocol announced Stable-Baselines3 2.6.0. Every training run used version 2.9.2, installed on 8 November at 10:11, before the first training run (deviation 14, item c). The versions in `envs/train.txt` are the ones that actually ran.

Reproducing on another machine. On a given machine, a run is determined by its random seed: each training runs on a single thread (`OMP_NUM_THREADS=1`, `torch.set_num_threads(1)`, single-thread XLA options for JAX). On another processor, MuJoCo, JAX and PyTorch round differently, so the same random seed gives a different curve there. A rerun elsewhere tests the statistics of the report. It recovers no curve identically.

5.3 The tasks

The four tasks are those of Henderson et al. (2018), in their current Gymnasium version: HalfCheetah-v5, Hopper-v5, Walker2d-v5 and Swimmer-v5. In each, MuJoCo simulates an articulated body. At every step the agent observes a vector of positions and velocities, then chooses a continuous action: the torque applied at each joint. The reward pays mostly for forward velocity, minus a cost that grows with motor effort. For Hopper and Walker2d, it adds a bonus at every step where the body stays upright. An episode stops after 1,000 steps at most.

- HalfCheetah is a planar cheetah with six joints. It never falls, so its episodes always last 1,000 steps.
- Hopper is a one-legged hopper. The episode stops when it falls, which makes scores very uneven from one run to another.
- Walker2d is a planar two-legged walker, also stopped when it falls.
- Swimmer is a three-segment swimmer in a viscous fluid. Its scores are far lower than those of the other three tasks.

The 'score' of an episode is its return: the sum of the rewards received from its first step to its last. Scores from different tasks are not comparable. Every comparison in this report is made within one task.

Experiment E6 also uses Hopper-v4, the previous version. The PPO settings that RL Baselines3 Zoo publishes for Hopper are filed under this task name in the collection's `hyperparams/ppo.yml`: they were tuned on that version.

5.4 The programs

Our experiments compare four *programs* (four ready-to-use code bases) that implement three *algorithms*. A fifth program, TRPO, serves in one experiment only (B1). Table 5.1 gives their settings.

PPO, twice. PPO (*Proximal Policy Optimization*, Schulman et al. 2017) was the reference algorithm of 2018. It is still a common point of comparison in 2026. It is also the direct ancestor of GRPO, the algorithm of the language-model study. PPO is an 'on-policy' algorithm: it collects a batch of transitions with the current policy, uses it for a few gradient passes, then discards it. Let $r_t(\theta) = \pi_\theta(a_t \mid s_t) / \pi_{\theta_{\text{old}}}(a_t \mid s_t)$ be the ratio of the probabilities of the action under the new and the old policy. Let $\hat{A}_t$ be the advantage estimated by GAE (*generalized advantage estimation*): how much more the action taken returned than the value function predicted for that state. PPO maximises the clipped objective

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

with $\epsilon = 0.2$, plus a value-function error term. Two programs implement PPO:

- PPO PyTorch, the `PPO` class of Stable-Baselines3 (`ppo_sb3` in the code);
- PPO JAX, the `PPO` class of SBX (`ppo_sbx` in the code).

Both share the same default values for every hyperparameter of the algorithm. They differ in weight initialisation: Stable-Baselines3 initialises its layers with an orthogonal matrix (`ortho_init=True`), while SBX keeps the default initialisation of Flax (`ortho_init=False`). They also differ in the numerical library, its random generators and its rounding. Experiment E3 measures what these differences cost.

SAC. SAC (*Soft Actor-Critic*, Haarnoja et al. 2018) has been the 'off-policy' standard of continuous control since 2019. It keeps every transition it has seen in a replay buffer and learns at every step from a batch drawn from that buffer. It maximises the return augmented with the entropy of the policy, $\sum_t \mathbb{E}\big[r_t + \alpha\,\mathcal{H}(\pi(\cdot \mid s_t))\big]$, with a coefficient α tuned automatically towards a target entropy. Two critic networks estimate the value

of actions. Slow copies of these networks, called target networks, stabilise learning. The program is the `SAC` class of SBX (`sac`).

CrossQ. CrossQ (Bhatt et al. 2024), published at ICLR 2024, targets data efficiency, the score reached per step of interaction with the environment. A descendant of SAC, it removes the target networks and applies batch normalisation in the critic and the actor. It also widens the critic. The program is the `CrossQ` class of SBX (`crossq`). Its critic has two layers of 1,024 units, instead of the paper's 2,048: SBX makes this choice to compute faster. We kept it.

TRPO, for B1 only. TRPO (*Trust Region Policy Optimization*, Schulman et al. 2015) is the algorithm of figure 5 of Henderson et al. (2018), the figure whose t of 9 experiment B1 tries to explain (section 5.10.2). It maximises the advantage weighted by the probability ratio under a constraint on the mean Kullback-Leibler divergence between the old and the new policy, $\hat{\mathbb{E}}_t\left[\mathrm{KL}(\pi_{\theta_{\text{old}}} \| \pi_\theta)\right] \leq \delta$, solved by conjugate gradient followed by a line search. The program is the `TRPO` class of sb3-contrib (`trpo`), with all its default values.

TD3, planned then dropped. The protocol planned TD3 if the budget allowed. The budget did not allow it: TD3 was not launched (deviation 1).

Why the default values. The four programs of the main grid keep the default values of their library. None was tuned. This choice applies Pineau's request on baselines to the letter (flaw F7, baselines tuned less carefully than the method): each program receives the same tuning effort, here none. The choice has a cost. A program well served by its default values is favoured. The ranking of E2 holds for these settings. Experiment E6 measures, on one case, what a published tuning changes. The only command that builds a program of the grid, in `sim/run.py`, is `cls("MlpPolicy", TASK, seed=SEED, verbose=0)`, with no other argument.

The published Zoo settings, for E6. Experiment E6 pits PPO PyTorch with its default values against PPO PyTorch with the settings RL Baselines3 Zoo publishes for Hopper-v4. `sim/run.py` copies them into the constant `ZOO_PPO_HOPPER`, identical to those of `rl_zoo3/hyperparams/ppo.yml` in version 2.9.2 (deviation 5):

- batches of 512 steps, minibatches of 32, 5 passes, $\gamma = 0.999$, $\lambda = 0.99$ for GAE;
- learning rate $9.808{,}28 \times 10^{-5}$, entropy coefficient 0.002,295,19, value coefficient 0.835,671, gradient norm clipped at 0.7, clipping 0.2;
- networks of two layers of 256 units for the policy and the value, ReLU activation, no orthogonal initialisation, initial log standard deviation of the actions at -2;
- normalisation of observations and rewards by a running mean and standard deviation (`VecNormalize`).

Reward normalisation divides each reward by the standard deviation of a running discounted return. In the Zoo recipe, the discount factor of that return is the algorithm's, 0.999. Our first pass (`ppo_sb3_zoo`, option `--zoo`) left the default value of `VecNormalize`, 0.99. The second pass (`ppo_sb3_zoofix`, option `--zoofix`) passes 0.999 (deviation 2). Both are kept.

Table 5.1. The settings of the five programs, read in the code of the installed libraries (Stable-Baselines3 2.9.2, sb3-contrib 2.9.2, SBX 0.28.0). They are the default values: no program was tuned. An empty cell means the setting does not exist for that algorithm. The networks are multilayer perceptrons, given by the widths of their hidden layers.

Setting	PPO PyTorch	PPO JAX	SAC	CrossQ	TRPO
Learning rate	3×10^{-4}	3×10^{-4}	3×10^{-4}	1×10^{-3}	1×10^{-3} (critic)
Discount factor γ	0.99	0.99	0.99	0.99	0.99
Steps collected per update	2,048	2,048	1	1	2,048
Batch size	64	64	256	256	128
Passes over each batch of transitions	10	10			
GAE parameter λ	0.95	0.95			0.95
Clipping ϵ	0.2	0.2			
Divergence constraint δ					0.01
Entropy coefficient	0	0	automatic	automatic	
Replay buffer			1,000,000	1,000,000	
Steps before learning starts			100	100	
Target network update τ			0.005	no target network	
Critic updates per actor update			1	3	10
Policy network	64, 64, tanh	64, 64, tanh	256, 256, ReLU	256, 256, ReLU	64, 64, tanh
Value or critic network	64, 64, tanh	64, 64, tanh	2 times 256, 256	2 times 1,024, 1,024, batch-normalised	64, 64, tanh
Orthogonal initialisation	yes	no			yes

5.5 The budgets

What the protocol fixes. PPO PyTorch, PPO JAX and TRPO train for one million steps. SAC and CrossQ train for 200,000 steps. A step is one transition of the environment: an observation, an action, a reward.

Why two budgets. The two families of algorithms are not compared in the same regime. An on-policy algorithm discards its data after each update, so it needs many steps. One million is the 2018 order of magnitude for these tasks. An off-policy algorithm reuses its data at every step: papers from 2024 to 2026 compare SAC and CrossQ in a 'low-data' regime. The protocol set 300,000 steps. Deviation 1 brought them down to 200,000, before the first training run, to keep the compute within a shared machine: we then measured 19 ms per step for these two algorithms.

The exact number of steps. `sim/run.py` trains in blocks of 50,000 steps and evaluates after each block. The call is `model.learn(50_000, reset_num_timesteps=False)`, repeated until the budget is reached. PPO and TRPO collect their transitions in batches of 2,048 steps and stop only at the end of a batch, so each block lasts 51,200 steps. The

saved models of PPO PyTorch, PPO JAX and TRPO count 1,024,000 steps, while the `step` column of the results reads 1,000,000. The E6 models tuned like the Zoo, with batches of 512 steps, count 1,003,520. SAC and CrossQ stop at 200,000 steps, as announced (deviation 14, item d).

The equal budget. The comparison of E2 is made at unequal budgets, which the report states each time. Deviation 12, item iii, adds a reading at equal budget: PPO is read at 200,000 steps on its evaluation curve, at the same point as SAC and CrossQ. This rule was written before any PPO score at 200,000 steps had been looked at.

5.6 The random seeds

What a random seed fixes. A random seed is the starting number from which the computer draws all the randomness of a training run: the initialisation of the networks, the initial positions of the simulated body, the noise of explored actions, the batches drawn from the replay buffer. `sim/run.py` passes it to the constructor of the program (`seed=SEED`), which derives from it the generators of NumPy, PyTorch or JAX and that of the training environment. Same program, same random seed, same machine: same run.

Training random seeds. The training random seeds are the integers 0 to 29, fixed before the first training run:

- 0 to 9 for every pair of program and task, the main results;
- 10 to 19 for the consolidation of deviation 12, the twenty random seeds;
- 0 to 29 for PPO PyTorch on HalfCheetah (E1) and for TRPO on HalfCheetah (B1).

Random seeds 10 to 19 of PPO PyTorch on HalfCheetah, trained for E1 with the same code, serve as they are in the consolidation.

Evaluation random seeds. Evaluation uses random seeds 10,000 to 10,009, one per evaluation episode. Training never uses them: it uses random seeds 0 to 29. The same ten random seeds serve at every evaluation point, in every run and in every physics, so two policies are always evaluated from the same ten starting positions.

Random seeds of the analyses. The draws of the analyses have their own random seeds, fixed in each script: 2018 for the draws of E1 and B1 (`np.random.default_rng(2018)`), 2026 for E5 and for every rliable bootstrap (section 5.9.10). Two executions of an analysis give a file identical byte for byte.

Each script names its random seeds. The analysis scripts first read every file present. With random seeds 10 to 19 from deviation 12, E6 would have gone from ten to twenty random seeds without saying so. Since deviation 14, item f, each script names its random seeds and stops if one is missing. The published results did not move.

5.7 Evaluation, held out

The evaluation protocol. After each training block, the `evaluate` function of `sim/run.py` plays ten episodes. For episode i, from 0 to 9:

1. a fresh environment is created (`gym.make(TASK)`), separate from the training one;
2. it is reset with random seed $10{,}000 + i$;
3. the policy chooses each action without randomness (`deterministic=True`): the mean action for PPO, SAC and CrossQ, instead of a draw from the policy's distribution;
4. the episode runs until the body falls or until 1,000 steps, then its return is recorded.

The function returns the mean and the standard deviation of the ten returns (`np.std`, without Bessel's correction). For the programs tuned like the Zoo, each observation goes through the training normalisation, whose statistics stay frozen during evaluation. A deterministic policy evaluated on ten fixed random seeds always gives the same score. The variation among the ten episodes comes from the starting positions alone.

The score of a run. The score of a run is the mean of the ten returns at the last evaluation, at its program's budget: the `eval_mean` column of the last line of its file. The protocol fixed it as the main measure. Experiment B3 tries two others.

A modified physics, at the end. At the end of training, the final policy plays ten more episodes, on the same evaluation random seeds, in a modified physics. The code modifies the environment's MuJoCo model before the reset:

- the mass of every body is multiplied by 1.2 (`body_mass[:] *= 1.2`);
- the sliding friction coefficient of every geometry, the floor included, is multiplied by 0.8 (`geom_friction[:, 0] *= 0.8`).

The mean of the ten returns goes into the `perturbed_mean` column of the last line. Experiment E7 uses it.

The physics modified by degrees. Deviation 12, item ii, re-evaluates each final policy in other physics, with no training at all (`sim/dose.py`). One quantity changes at a time:

- the masses, multiplied by 0.7, 0.8, 0.9, 1.0, 1.1, 1.2 and 1.3, from 30% lighter to 30% heavier;
- the sliding friction, multiplied by 0.6, 0.7, 0.8, 0.9, 1.1 and 1.2, from 40% weaker to 20% stronger.

Each physics is played on the same ten evaluation random seeds, with the same evaluation code. The factor 1.0 serves as a control: a correct re-evaluation gives back there the final score written by `sim/run.py`. The result of each policy is a file `results/dose/<program>/<task>/seed<k>.json`. The PPO policies of the main grid have no observation normalisation: `sim/dose.py` evaluates them as they are.

5.8 The run format

One CSV file per run. Each run writes a file `results/<program>/<task>/seed<k>.csv`, rewritten after each evaluation. Its columns are `step`, `eval_mean`, `eval_std`, `perturbed_mean`, `wall`:

- `step`, the number of steps announced at this evaluation, in increments of 50,000;
- `eval_mean` and `eval_std`, the mean and the standard deviation of the returns of the ten evaluation episodes;
- `perturbed_mean`, the mean of the returns in the modified physics, empty except on the last line;
- `wall`, the seconds elapsed since the start of the run.

A PPO or TRPO curve has twenty lines, a SAC or CrossQ curve has four. The module `src/pineau_rl/runs.py` reads this format. Its function `read_run` checks the required columns and that the steps strictly increase. It treats as unfinished a run whose last line is not at the program's budget. On request, it also cuts a curve at a given step (the equal-budget reading). The function `write_run` writes this format from any library.

The final policies. Each run saves its final policy in `models/<program>/<task>/seed<k>.zip`, with the normalisation statistics alongside for the programs tuned like the Zoo (`.vecnorm.pkl`). The file `sim/models/MANIFEST.sha256` gives the SHA-256 fingerprint of 250 policies, taken on 15 November 2026 on the machine that trained them. The PPO, TRPO and E6 policies are in the repository. Those of SAC and CrossQ, too heavy for git, are attached to the `policies-v1` release of the repository on GitHub. The script `sim/fetch_models.py` downloads them one at a time, separated by a random pause of 1.5 to 4 seconds, then checks each archive and each policy against the manifest. The option `--check` only checks the policies already present.

The logs. Each launch writes a log. Since deviation 14, each attempt keeps its own (`logs/<task>.attempt<n>.log`, never overwritten) and appends a line to the ledger `logs/attempts.jsonl`: task, attempt number, start, end, exit code, SHA-256 fingerprint of the result. For the earlier period, the ledger `logs/attempts_reconstructed.csv` rebuilds one line per attempt from the dates of the logs, the outputs of the launchers and the git history (section 5.11).

5.9 The statistics

This section defines each statistic used, in the order in which chapter 6 needs them. The functions are in `src/pineau_rl/stats.py` and `src/pineau_rl/bootstrap.py`.

5.9.1 The interquartile mean

The interquartile mean (IQM) discards the lowest quarter and the highest quarter of the scores, then averages the rest (Agarwal et al. 2021). It resists an outlying run better than a mean does and is more precise than a median. rliable computes it with SciPy's

trimmed mean at 25% on each side (`trim_mean(x, 0.25)`), which removes $\lfloor n/4 \rfloor$ scores at each end. With the sorted scores $x_{(1)} \le \cdots \le x_{(n)}$ and $g = \lfloor n/4 \rfloor$:

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

With ten random seeds, $g = 2$: the IQM is the mean of the six middle scores. With twenty, $g = 5$ and ten scores remain. With thirty, $g = 7$ and sixteen remain.

5.9.2 The stratified bootstrap interval

The uncertainty of an IQM is measured by the bootstrap (Efron 1979). We draw n scores with replacement among the n observed, recompute the IQM, then repeat B times. The 95% interval is formed by the 2.5th and 97.5th percentiles of the B values obtained (the percentile method, rliable's default):

$$\left[\, q_{0.025}\big(\mathrm{IQM}(x^{*1}), \ldots, \mathrm{IQM}(x^{*B})\big),\ q_{0.975}\big(\mathrm{IQM}(x^{*1}), \ldots, \mathrm{IQM}(x^{*B})\big) \,\right], \qquad B = 2{,}000. \tag{5.3}$$

rliable calls this bootstrap 'stratified': with several tasks, it resamples the runs within each task, without mixing tasks. Each of our calls handles one task, where stratification changes nothing: the bootstrap amounts to resampling the runs of one program.

5.9.3 The probability of improvement

The probability of improvement of a program X over a program Y is the probability that a run of X, drawn at random, beats a run of Y, drawn at random, ties counting as one half (Agarwal et al. 2021). With n_x scores x_i and n_y scores y_j:

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

where U is the Mann-Whitney statistic (Mann and Whitney 1947). rliable computes it this way (`scipy.stats.mannwhitneyu`). It equals 0.5 when the two programs are even. Its 95% interval comes from the same bootstrap as the IQM: the runs of X and those of Y are resampled separately, $B = 2{,}000$ times, then the 2.5th and 97.5th percentiles are taken.

5.9.4 The winner rule

The protocol fixes one rule for declaring that one program beats another. We call it *the settled-winner rule*. A gap is settled when one of two conditions holds:

- the 95% intervals of the two IQMs do not overlap;
- the 95% interval of the probability of improvement excludes 0.5.

Otherwise no winner is declared, whatever the gap between the IQMs. E3 applies both conditions (`e3.py`). E2, E2 with Bonferroni and B3 use the second: a pair is called 'settled' there when the interval of its probability of improvement excludes 0.5. E7 and the physics by degrees use the analogue for a ratio: a loss is settled when the interval of the IQM of the ratio stays below 1.

5.9.5 Student's and Welch's t-tests

The t-test compares two means. For two groups of sizes n_a and n_b, with means $\bar{a}$ and $\bar{b}$ and sample variances s_a^2 and s_b^2 (with Bessel's correction), Student's test (Student 1908) assumes equal variances and computes

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

with $n_a + n_b - 2$ degrees of freedom. Welch's test (Welch 1947) does not assume equal variances:

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

The two-sided p-value is $2\left(1 - F_\nu(|t|)\right)$, where F_ν is the cumulative distribution function of Student's distribution with ν degrees of freedom. Two groups of five give $\nu \geq 4$, since ν is at least the smaller of the two values $n - 1$. All our t-tests are Welch tests (`scipy.stats.ttest_ind`, option `equal_var=False`). Experiment E1 puts this test in question, as the protocol planned.

5.9.6 Three readings of the 2018 average t

The caption of figure 5 of Henderson et al. (2018) speaks of the 'average 2-sample t-test across entire training distribution'. It compares two groups of whole learning curves. Let $C_{s,k}^a$ be the evaluation score of run s of group a at evaluation point k, for $k = 1, \ldots, K$ ($K = 20$ for PPO and TRPO). Our analyses compute three readings of this caption (`welch_curve` and `welch_pooled`).

- *The average t.* A Welch test at each point, on the five scores of each group, gives t_k and ν_k. We report $\bar{t} = \frac{1}{K}\sum_k t_k$, converted to p with the mean of the ν_k.
- *The average p.* The same computation, summarised by $\bar{p} = \frac{1}{K}\sum_k p_k$.
- *The pooled points.* The $5 \times K$ points of each group are taken as independent observations, then compared by one Welch test, with about 198 degrees of freedom.

5.9.7 The statistic of Henderson's script

On 17 June 2017, Peter Henderson published the script `from_same_dist.py` (Henderson 2017), a companion to Islam et al. (2017). The script computes something other than the three readings above. The function `henderson_script` reproduces it on our files. Let $M_{s,k}$ be the `eval_mean` column and $S_{s,k}$ the `eval_std` column of run s at point k. Let A and B be the two groups of five runs.

1. The script averages the curves of each group, column by column: $\bar{M}_k^A = \frac{1}{5}\sum_{s \in A} M_{s,k}$ and $\bar{S}_k^A = \frac{1}{5}\sum_{s \in A} S_{s,k}$.
2. At each point k, it applies a Welch test built from these statistics alone (`scipy.stats.ttest_ind_from_stats`), with n episodes per group:

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

3. It reports the mean of the t_k and the mean of the p_k.

The denominator of equation (5.7) contains the standard deviation *between the episodes* of one run. The variability *between random seeds*, the one the 2018 figure was meant to measure, never enters it. The five random seeds of a group merge into one mean curve. For our runs, $n = 10$, the number of evaluation episodes. A sensitivity analysis takes $n = 5$, the number of trajectories per iteration in the 2018 runs (5,000 steps per iteration, episodes of 1,000 steps). Our $S_{s,k}$ is a standard deviation without Bessel's correction, passed to the test as it is, just as the script passes its own columns.

5.9.8 Corrections for multiple comparisons

When m tests are run, the probability that at least one is wrong exceeds the threshold of each. Two corrections are used.

- *Bonferroni* (Dunn 1961) raises each interval to the level $1 - 0.05/m$. The m intervals then hold jointly at a level of at least 95%. E2 counts $m = 24$ comparisons (6 pairs on 4 tasks): each interval moves to the level $1 - 0.05/24$, that is 99.79% (deviation 18).
- *Holm* (Holm 1979) adjusts sorted p-values $p_{(1)} \leq \cdots \leq p_{(m)}$ into $\tilde{p}_{(i)} = \min\big(1, \max_{j \leq i}(m - j + 1)\, p_{(j)}\big)$. E6 uses it over its two or three comparisons.

5.9.9 Other tests

The Shapiro-Wilk test (Shapiro and Wilk 1965) measures how far a distribution departs from the normal distribution. E1 applies it to the thirty scores of PPO PyTorch. The Mann-Whitney test (Mann and Whitney 1947) compares two groups by their ranks. The E6 diagnosis reports it next to the Welch test.

5.9.10 Reproducible intervals

rliable 1.2.0 accepts a `random_state` argument but draws its resamples from NumPy's global generator (`np.random.choice`). Six scripts did not give their intervals back identically from one execution to the next (deviation 13). Each script now seeds this generator at the start (`np.random.seed(2026)`), then calls rliable in a fixed order. Every call consumes the global generator, so the interval of a comparison depends on the calls that precede it in the same script. The correction changed no point estimate and no verdict. The bounds moved at the level of Monte Carlo noise, by at most 16 points on an IQM of 2,423 for E6.

A variant for the E6 diagnosis. The scripts `e6_diag.py` and `e6_official.py` compute the intervals of the IQMs with $B = 5{,}000$ resamples. Their probability of improvement has a bootstrap of its own (`probability_of_improvement_bootstrap`): 5,000 draws, the two groups resampled one after the other with a generator of random seed 2018, without rliable. The same score can therefore receive two slightly different intervals depending on the script that computes it: the IQM of default PPO PyTorch on Hopper-v5 has one interval in `e6.json` and another in `e6_diag.json`. The report cites the diagnosis interval, as `docs/conclusions.md` does.

5.10 The experiments, one by one

Table 5.2 summarises the designs. It holds design facts only: the question, the design, the number of runs and the moment the analysis was written.

Table 5.2. The experiments: the question asked, the design, the runs used and the moment the analysis was fixed. 'Preregistered' means written before any result of the experiment. Every number in this table is a design choice.

Experiment	Question	Design	Runs	Analysis fixed
E1	Do two groups of runs of the same program pass for two algorithms?	PPO PyTorch on HalfCheetah, 30 random seeds. 10,000 draws of two groups, Welch test on the final scores, at random then sorted.	30 training runs	preregistered (8 November)
E1, second statistic	Is the average t on the 2018 curve recovered?	The same 30 curves, Welch test at each of the 20 points, average t and average p.	the 30 of E1	after the E1 results (deviation 3)
B1, TRPO	Does the spread of TRPO explain the t of 9?	TRPO on HalfCheetah, 30 random seeds, three readings of the average t.	30 training runs	before any TRPO training (deviation 4)
B1, script	Does Henderson's computation explain the t of 9?	The 2017 script applied to the TRPO and PPO curves, same draws.	the 30 of E1 and the 30 of TRPO	after B1 TRPO, before this computation (deviation 6)
E2	Does the leading program change from task to task?	4 programs on 4 tasks, IQM and interval, probability of improvement per pair.	160 training runs	details fixed before any CrossQ score (deviations 7 and 14)
E2, Bonferroni	Does the E2 verdict survive 24 comparisons?	The 24 intervals of E2 at a simultaneous level of 95%.	those of E2	after E2, before this computation (deviation 18)
E3	Do two programs of the same algorithm give the same result?	PPO PyTorch against PPO JAX, 4 tasks, same random seeds.	80 of those of E2	preregistered
E4	What happens to the mean when the best 3 runs out of 10 are kept?	Mean and standard deviation of the 10, then of the best 3, reversals among 24 comparisons.	those of E2	before any CrossQ score (deviation 7)
E5	How many random seeds recover the correct ranking?	For each comparison, 10,000 draws with replacement of n runs, n from 3 to 20.	those of E2, the 30 of E1	before any CrossQ score (deviation 7)
E6	What does a baseline tuned with the published settings change?	PPO PyTorch on Hopper, default values against Zoo settings, then diagnosis and official script.	40 training runs beyond E2, plus 10 official	preregistered, diagnosis after the first result (deviations 2 and 5)

Experiment	Question	Design	Runs	Analysis fixed
E7	What does a modified physics cost?	Ratio of the score in modified physics to the score in training physics, per run.	those of E2	preregistered, descriptive ranking added afterwards (deviation 11)
Physics by degrees	Does the E7 loss come from one modification alone?	Masses from 0.7 to 1.3 times, friction from 0.6 to 1.2 times, one quantity at a time.	320 re-evaluations (with twenty random seeds)	rule written before any computation (deviations 12 and 14)
B3	Does the chosen measure change the ranking?	Final score, best score, mean of the curve, reversals between measures.	those of E2	before any CrossQ score (deviation 8)
Twenty random seeds	Do the results at ten random seeds hold?	Random seeds 10 to 19, same rules, E2, E3, E5, E7, B3, physics by degrees.	150 more training runs	before any new training (deviations 12 and 20)
Equal budget	Does the E2 ranking depend on PPO's budget?	PPO read at 200,000 steps, E2 and B3 recomputed.	those of E2	before any look at these scores (deviation 12)
E4 replicated	Does the E4 inflation reproduce?	E4 unchanged on random seeds 10 to 19.	160 runs of random seeds 10 to 19	before any new training (deviation 12)

5.10.1 E1: same code, different random seed

E1 trains PPO PyTorch thirty times on HalfCheetah-v5, one million steps each, random seeds 0 to 29, changing nothing else. The script `sim/analysis/e1.py` reads the thirty final scores, then computes, with a generator of random seed 2018:

- the statistics of the distribution: mean, standard deviation, median, minimum, maximum, coefficient of variation (standard deviation divided by the mean), Shapiro-Wilk test;
- for $k = 3$, 5 and 10, 10,000 draws of two disjoint groups of k runs, obtained by permuting the thirty, then taking the first k against the next k. Each draw receives a Welch test and the relative gap between the means $|\bar{a} - \bar{b}| / \min(\bar{a}, \bar{b})$. The script reports the share of draws where $p < 0.05$, along with the median and the 95th percentile of the relative gap;
- the 'sorted' split: 10,000 draws of ten runs without replacement, sorted by score, then a Welch test of the best five against the worst five. This split shows what sorting the runs before the comparison would produce;
- the selection of the best three out of ten: over 10,000 draws of ten runs, the mean of the best three divided by the mean of the ten.

Since the two groups always come from the same program, every declared difference is a false positive. With groups drawn at random and a well-calibrated test, the expected share is 5%.

The second statistic. Henderson et al. (2018) test curves, where we tested final scores (section 5.9.6). `sim/analysis/e1_curve.py` redoes their statistic on the thirty curves: 10,000 draws of ten runs, split into two groups of five at random or sorted by final score, then a Welch test at each of the twenty evaluation points. We report the share of draws declared different by the average t, by the average p and by the test at the last point, the 95th percentile of $|\bar{t}|$ and the share of draws where $|\bar{t}| \geq 9.09$. This statistic was added on 9 November, after the results of `e1.py` (deviation 3). It complements the preregistered analysis without replacing it.

5.10.2 B1: the puzzle of the t of 9

For TRPO on HalfCheetah, Henderson et al. (2018) publish $t = -9.09$ and $p = 0.0016$ between two groups of five runs of the same code. Pineau specifies that these groups are not sorted. B1 looks for the origin of this number in two stages, each written before its computation.

First stage: their algorithm. Deviation 4, written on 9 November before any TRPO training, launches thirty TRPO runs on HalfCheetah-v5, random seeds 0 to 29, one million steps, with the default values of sb3-contrib 2.9.2 and the same evaluation as E1. `sim/analysis/b1_trpo.py` applies the three readings of section 5.9.6 to the TRPO and PPO curves, with the same draws as `e1_curve.py`. The reading rule was fixed in advance. If the 95th percentile of $|\bar{t}|$ under random splits comes close to 9 with TRPO, the spread of the algorithm explains their number. If only the pooled reading comes close, their computation explains it. If no reading comes close, neither lead explains it: the report then says so.

Second stage: their script. Deviation 6, written on 9 November at 23:55 after the results of `b1_trpo.py` and before any computation with the script, notes two facts. The first is arithmetic. The published pair $(t, p) = (-9.09, 0.0016)$ implies about 3.4 degrees of freedom in a two-sided test. It rules out the pooled-points reading, with about 198 degrees of freedom, which would give a p of the order of 10^{-16}. It also rules out the average t converted to p, since a Welch test on five against five has at least 4 degrees of freedom. Only the average of the p values remains compatible. The second fact is the 2017 script (section 5.9.7). `sim/analysis/b1_script.py` applies this computation to the thirty TRPO curves and the thirty PPO curves, with the same draws (generator of random seed 2018, 10,000 draws of ten runs split five against five, at random then sorted), for $n = 10$ then $n = 5$. It also reports, per program, the ratio of the mean standard deviation between episodes at the last point to the standard deviation of the final scores between random seeds.

Differences stated in advance. Our evaluation episodes follow a deterministic policy, whereas the 2017 progress files record the training episodes, played by the stochastic policy. We have twenty evaluation points, while their curves have one line per iteration. We cannot prove that this script produced their figure 5. Its name (‘from the same distribution’) and its output (‘t=..., p=...’) make it likely.

5.10.3 E2: does the ranking change from task to task?

E2 trains the four programs on the four tasks, ten random seeds each. `sim/analysis/e2.py` reads the final score of each run, then, for each task, computes:

- the IQM of each program with its 95% bootstrap interval (equations (5.2) and (5.3));
- the ranking by IQM;
- for each of the six pairs, the probability of improvement of the higher-ranked program over the other, with its interval (equation (5.4)) and the flag 'settled' if the interval excludes 0.5;
- a performance profile: for 25 thresholds τ from 0 to 1.2, the share of runs whose score, divided by the best IQM of the task, exceeds τ.

The *leader* of a task is the program with the highest IQM. The ranking *changes* if the leader is not the same on every task. The change is *settled* when the leader of one task is beaten, in a settled way, by another program on another task. The script counts these settled changes: each triple (task where X leads, task where X loses, program that beats X) counts as one. These details were written on 10 November, after the PPO and SAC results and before looking at any CrossQ result (deviation 7, read with the correction of deviation 14, item b).

With the Bonferroni correction. `sim/analysis/e2_bonferroni.py` makes the same calls, in the same order, after the same random seed: the bootstrap resamples are identical to those of `e2.py`. Only the percentile level changes, to 99.79%. It counts the settled changes with the same definition (deviation 18).

5.10.4 E3: two programs of the same algorithm

`sim/analysis/e3.py` pits PPO PyTorch against PPO JAX on each task, with the same random seeds and the scores at one million steps. It reports the two IQMs with their intervals, the ratio of the IQMs, the probability that a PPO JAX run beats a PPO PyTorch run with its interval, the overlap of the intervals, then the winner under the rule of section 5.9.4.

5.10.5 E4: the best three out of ten

For each program and each task, `sim/analysis/e4.py` computes the mean and the standard deviation of the ten runs, then of the best three, finally the inflation $\bar{x}_{\text{best }3}/\bar{x}_{10} - 1$. A comparison is a pair of programs on one task: there are 24. It *reverses* if the sign of the difference in means changes when each program keeps only its best three runs. The script reports the median inflation, its range, the median ratio of the standard deviations and the number of reversals.

5.10.6 E5: how many random seeds?

`sim/analysis/e5.py` starts from the 24 comparisons. For each, the correct order is the sign of the difference between the means of all the available runs. The observed distribution of the runs stands in for the true distribution of the algorithm's scores.

For $n = 3$, 5, 10 and 20, we draw n runs with replacement 10,000 times for each of the two programs, then count the share of draws that recover the correct order:

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

For PPO PyTorch on HalfCheetah, the analysis at ten random seeds draws from the thirty runs of E1. The script reports the median of $\hat{P}_n$ over the 24 comparisons, then over the *close* comparisons, those whose relative gap between means $|\bar{x} - \bar{y}| / \max(|\bar{x}|, |\bar{y}|)$ stays below 20%. Deviation 20 fixed a defect before the computation at twenty random seeds: the script took as the correct order the mean of the first ten runs, even with twenty. It now takes the mean of all the chosen random seeds. The analysis at ten random seeds came back identical to the byte.

5.10.7 E6: the tuned baseline

E6 was meant to illustrate flaw F7, the poorly tuned baseline: the gap a 'new method' can claim for itself when it alone is tuned. The initial design pits, on Hopper-v5, ten runs of PPO PyTorch with its default values (those of E2) against ten runs with the Zoo settings (`e6.py`).

The diagnosis. The first result triggered a diagnosis, added after we saw it (deviation 2). `e6_diag.py` compares five groups of ten runs: default values on v5, Zoo with the discount-factor error on v5, corrected Zoo on v5, default values on v4, corrected Zoo on v4. Three comparisons are corrected by Holm: the error (corrected Zoo against Zoo with the error, on v5), the fair test on the Zoo's version (corrected Zoo against default values, on v4) and the same test on v5. Each comparison reports the Welch p, the Mann-Whitney p and the probability of improvement with its interval.

The official pipeline. Deviation 5, written after the diagnosis and before any official training, asks whether the plateau seen in the first result, a score that stops rising after the first training blocks, comes from the published recipe or from the way we applied it. The script `sim/zoo_official.sh` runs the collection's official script ten times, `python -m rl_zoo3.train --algo ppo --env Hopper-v4 --seed i`, for i from 0 to 9, in the separate environment `sim/env-zoo`, on one thread, with no evaluation during training (`--eval-freq -1`). `e6_official.py` then evaluates each official policy with our protocol: ten deterministic episodes, evaluation random seeds 10,000 to 10,009, observation normalisation loaded then frozen, no reward normalisation. Two comparisons, fixed in advance and corrected by Holm, pit the official policies against our corrected Zoo on v4, then against our default values on v4. The reading rule was written down. If the official runs clearly exceed 1,000, our application was at fault. If they plateau too, the published recipe does worse than the default values in this version of the libraries. The report then says so without speaking of a baseline badly tuned by us.

5.10.8 E7: the modified physics

For each run s, `sim/analysis/e7.py` computes the ratio of the two final scores

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

then, for each program and each task, the IQM of the ρ_s with its bootstrap interval. The verdict is a *loss* if the interval stays below 1, a *gain* if it stays above, 'no detectable change' otherwise. The IQM of the ratios differs from the ratio of the IQMs: two rankings of scores do not show the share of score kept.

The descriptive ranking. `e7_rank.py` ranks the four programs on each task by the IQM of the score, in the training physics then in the modified physics. It says whether the leader changes. This complement, decided after the E7 results, has no test (deviation 11).

The physics by degrees. `sim/analysis/dose.py` reads the 160 re-evaluation files (320 with twenty random seeds). For each program, task, quantity and factor, it computes the IQM of the ratio to the score at 1.0 times, with its bootstrap interval, then the same verdict as E7. For each task and each physics, it ranks the four programs by the IQM of the score, with no test. Finally it checks that the score at 1.0 times recovers the final score of `run.py` and reports the largest relative gap. The rule dates from deviation 12, item ii, written before any computation. The script itself was written when 30 of the 160 re-evaluations already existed, one of which had been displayed: the commit that called it 'written before the results' was wrong (deviation 14, item a). The script has no free parameter.

5.10.9 B3: does the measure change the ranking?

`sim/analysis/b3.py` computes three measures per run on the recorded curves, with no training at all:

- the final score, at the last point;
- the best score reached along the curve;
- the mean of the curve, the area under the curve divided by its length.

For each task and each measure, it computes the IQM of each program and the ranking. A pair of programs *reverses* between two measures if the order of their IQMs changes. The reversal is *settled* if, under each of the two measures, the interval of the probability of improvement excludes 0.5, in opposite directions. PPO curves have twenty points over one million steps, SAC and CrossQ curves four points over 200,000: the mean of the curve does not cover the same budget (deviation 8).

5.10.10 The consolidation: twenty random seeds and equal budget

Deviation 12, written on 15 November before any new training, adds three parts with the same code and the same settings:

- random seeds 10 to 19 for the four programs on the four tasks, that is 150 new training runs, since PPO PyTorch on HalfCheetah already had its own;
- the physics by degrees of section 5.10.8;
- the equal-budget reading of section 5.5.

The module `sim/analysis/runs.py` reads three environment variables, all inactive by default: `SEEDS` (for example `0:20`), `BUDGET=equal` and `SUFFIX`, appended to the name

of the output file. Deviation 20, written on 21 November after all training had finished and before any computation on random seeds 10 to 19, fixes the list of computations, all launched by `make analysis`:

- on the twenty random seeds (suffix `_s20`): E2, E3, E5, E7, B3, the physics by degrees, the descriptive ranking of E7 and the Bonferroni correction of E2;
- E4 on random seeds 10 to 19 alone (suffix `_r10`), as an independent replication;
- E2 and B3 at equal budget, on ten random seeds (`_equal`) then on twenty (`_equal_s20`).

Before deviation 20, the only scores of random seeds 10 to 19 that had been read were the controls of the physics by degrees at 1.0 times, which compare each file with itself. The last two additions of the first list were not in deviation 12: they apply rules already written to the twenty random seeds. They are published whatever they show. The results at ten random seeds remain the main results.

5.11 The queue and the compute

The machine. Every training run used the processor of the Mac of section 5.2, shared with other projects. A training run or a re-evaluation occupies one core (one thread). Table 5.3 gives, per program and task, the number of runs, the steps, the median duration and the core-hours.

The launchers, in the order they were used. Several scripts launched the runs. Each skips a run whose final policy already exists.

- `sim/drive.sh` reads a list of jobs (`jobs.txt` by default) and launches P of them at a time through `xargs -P`. A run that fails is relaunched once, with a `.retry.log` log. `drive2.sh` is a copy of it for `jobs2.txt`, the corrected E6 runs.
- `sim/relay.sh` switches queues without killing a run: it waits for the running runs to finish, then restarts the main queue at one JAX training run (about two cores) and the B1 TRPO queue at two training runs, to stay at four cores.
- `sim/zoo_official.sh` waits for the TRPO queue to finish, then launches the ten official E6 training runs on two lanes of five.
- `sim/queue.sh` reads `jobs_consolidation.txt`, the list of deviation 12: 160 `train` lines and 320 `dose` lines. It reserves each job with an atomic `mkdir`, so two copies run without getting in each other's way. It keeps the ledger `logs/attempts.jsonl`. A stop by signal, with exit code 128 or above, is not relaunched (deviation 14, item g).
- `sim/worker.sh` is `queue.sh` with a ceiling: it reads `workers.max` before each job and exits if too many copies are running (deviation 15).
- `sim/scale.sh` sets this ceiling every hour. It sums the compute taken by the other programs on the machine (`%cpu` from `ps`) and deduces the free cores, minus one as a margin. It grants the project at most `MAX_CORES` cores, the GRPO run on the processor taking two. It launches new `worker.sh` copies when the ceiling rises. It launches none without the watchdog. Its decisions are in `logs/scale.log`.

Table 5.3. Compute of the experiments, read from the result files: for each program and task, number of finished runs, training steps per run (column `step` of the last row), median wall-clock time of one run and sum of the times, in hours. Wall-clock time is the column `wall` of `sim/results/<program>/<task>/seed<s>.csv` at the last evaluation (seconds since launch, evaluations included), and for GRPO the last `wall` of the training steps of `sim/llm/results/<model>/mul2/<reward>/seed<s>.jsonl`. These are not core-hours: each training run ran on one thread, PPO JAX on two at first (deviation 4), GRPO on the Apple GPU or on two CPU threads, on a shared machine whose load varied; the re-evaluations of E7 at several strengths and the ten official Zoo agents (E6) do not record their duration and are missing here. The PPO and TRPO models actually count 1,024,000 steps (blocks of 51,200, deviation 14 (d)). The reconstructed ledger `sim/logs/attempts_reconstructed.csv` counts 292 launches for the training runs of the grid, of B1 and of the E6 diagnosis, 250 of them successful (deviation 14 (e)).

Program	Task	runs	steps per run	median time (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 corrected	Hopper-v4	10	1,000,000	0.64	6.3
	Hopper-v5	10	1,000,000	0.73	7.7
Total MuJoCo training		400			525.4
GRPO, Qwen2.5 0.5B	mul2, true reward	5	100	4.32	19.0
	mul2, random reward	5	100	5.61	29.4
GRPO, Llama 3.2 1B	mul2, true reward	5	100	1.21	6.1
	mul2, random reward	4	100	1.58	7.1
Total GRPO (finished runs)		19			61.7

- `sim/watchdog.sh`, the watchdog, measures every 20 minutes the free disk space and the growth of swap memory since its launch. It stops every launcher and every run if the disk falls below `MIN_FREE_GB` gigabytes (10 by default) or if swap has grown by more than 8 GB. It records each stopped job in the ledger and releases its reservation.

The number of cores over the days. Until 15 November, the number of simultaneous runs was set by hand: four cores for the project from 9 November, then two copies of `queue.sh` for the consolidation. Deviation 15 (15 November) raised the ceiling to eight cores, set every hour by `scale.sh`. Deviation 19 (19 November) brought it down to two cores for the whole project, the GRPO run on the processor included. The number of parallel jobs changes no result: each run is determined by its random seed and runs on a single thread.

Stops and relaunches. The protocol required a crash to be relaunched once and recorded. The reconstructed ledger `logs/attempts_reconstructed.csv` counts 292 launches for the 250 training runs that precede the consolidation: 42 attempts were stopped and none crashed. Of the 42 stops, only 2 had been declared (deviation 14, item e). The stops have three causes: a general stop on 8 November, probably the one of the memory incident of the GRPO pilot, a stop requested on 9 November for an overloaded machine, then four stops to stay at four cores. Each stop hit every running run at once. No run was relaunched or discarded after its final score was known, nor for any reason linked to its score. For PPO PyTorch and PPO JAX, the relaunches recover the evaluation points of the stopped attempts identically. For SAC and CrossQ, the stopped curves were overwritten, which makes this check impossible. The later stops by the watchdog, on 12, 15 and 16 November, are described in deviations 10, 14 and 16 (chapter 9).

The chain to the documents. The runs write their CSV and JSON files. The scripts of `sim/analysis/` derive the JSON result files from them. `make verify` recomputes every analysis and compares the files byte for byte, then `docs/check_numbers.py` and `article/check_article.py` check every number in the documents against these files. Chapter 8 describes this chain and draws it (figure 8.1).

5.12 What was written before the results, what was written after

A protocol written in advance protects against an analysis chosen after seeing the data. It protects only if it states honestly what was decided afterwards. The decisions that affect an analysis are sorted below, deviation by deviation.

Written before any result concerned.

- the protocol itself (8 November): E1, E2, E3, E4, E5, E6, E7, their statistics, the settled-winner rule;
- the budget of SAC and CrossQ brought down to 200,000 steps (deviation 1), before the first training run;
- B1 with TRPO and its three readings (deviation 4), before any TRPO training;

- the official E6 pipeline and its reading rule (deviation 5), before any official training;
- the computation of Henderson's script (deviation 6), before this computation;
- the details of E2, E4 and E5 (deviation 7) and the design of B3 (deviation 8), before looking at any CrossQ score, when 5 to 7 CrossQ runs out of 40 had finished, instead of the 3 that deviation 7 stated (deviation 14, item b);
- the consolidation (deviation 12), before any new training;
- the Bonferroni correction (deviation 18), after E2 but before this computation;
- the list of computations at twenty random seeds (deviation 20), before any computation on random seeds 10 to 19.

Written after seeing a result.

- the E6 diagnosis (deviation 2), triggered by the first E6 result;
- the second statistic of E1 (deviation 3), after the results of `e1.py`;
- the descriptive ranking of E7 (deviation 11), after the E7 results, with no test;
- the random seed of the global generator of the bootstraps (deviation 13), a technical correction with no effect on the verdicts;
- the seven corrections of deviation 14: the date of the physics-by-degrees script, the counts of deviations 7 and 8, the version of Stable-Baselines3, the exact number of steps, the undeclared relaunches, the random seeds named in each script, the ledger of attempts.

None of these decisions removed a run, changed a setting or changed the settled-winner rule. The deviations with no effect on the analysis (organisation, interruptions, number of cores, battery) are in chapter 9.

CHAPTER 6

The experiments: results

This chapter gives the result of each experiment described in chapter 5. Every section follows the same order: the question, the plan in one sentence, the result with its intervals, then what the result means and what it leaves open.

Reading conventions. The main results are those with ten random seeds (0 to 9), as the protocol announced. The results with twenty random seeds (0 to 19) and at equal budget come next. Every divergence is stated. A score is the mean return of the ten final evaluation episodes of a run (section 5.7). The score of a program is an interquartile mean (IQM) of these scores over the random seeds. An interval written in square brackets, such as [2,268, 3,369], is a 95% bootstrap interval. A gap is *settled* when it satisfies the settled-winner rule (section 5.9.4): the IQM intervals are disjoint, or the interval of the probability of improvement excludes 0.5. The tables are produced by the script `report/tools/tables.py` from the JSON files of `sim/analysis/`. The numbers in the text come from the same files, rounded as in `docs/conclusions.md`.

6.1 E1: same code, another random seed

Question and plan. Can two groups of runs of the same program, which differ only in their random seeds, pass for two different algorithms? E1 trains PPO PyTorch thirty times on HalfCheetah-v5, one million steps each, random seeds 0 to 29, then compares groups drawn from these thirty runs with Welch's test (section 5.10.1).

A wide spread that is not bell-shaped. The mean of the thirty scores is 1,688, the standard deviation 568, or 34% of the mean. The median is 1,580. Twenty-seven runs end between 1,138 and 1,999. Three take off beyond 3,000, up to 3,430 (figure 6.1). These three runs pull the mean above the median. The Shapiro-Wilk test clearly rejects the normal distribution ($p = 4 \times 10^{-7}$). A t-test assumes roughly normal group means. With five runs drawn from such a distribution, the assumption is fragile.

Chance rarely fools the test, sorting often does. Table 6.1 and figure 6.2 give the share of the 10,000 draws in which Welch's test, at the 5% level, declares two groups from the same program different. Each of these declarations is a false positive.

- Three runs against three, drawn at random: 1.6% of the draws.
- Five against five: 1.2%.
- Ten against ten: 1.7%.
- The five best against the five worst of a draw of ten runs: 31.0%.

Drawn at random, the groups fool the test less often than the nominal 5% level. The test is conservative on this skewed distribution. The most likely explanation: an outlying

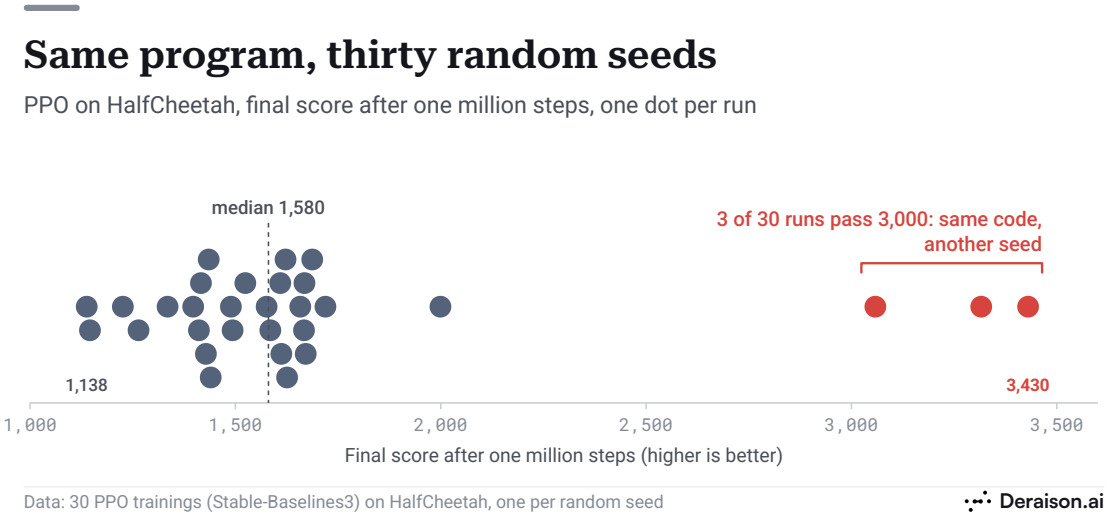


Figure 6.1. Thirty trainings of the same program on the same task: PPO written in PyTorch (Stable-Baselines3) on HalfCheetah-v5, one million steps each. Each dot is a run, placed at its final evaluation score, the mean return of ten episodes played without randomness on random seeds never seen in training. Only the training random seed changes from one dot to the next. Mean 1,688, median 1,580 (dotted line). Twenty-seven runs end between 1,138 and 1,999, three beyond 3,000.

run inflates the variance of the group that receives it along with its mean. Sorted, the groups fool the test nearly one time in three. The median p-value of the sorted split is 0.09.

Chance fools the eye. The groups drawn at random rarely fool the test, yet their means differ a great deal. The relative gap between the means, divided by the smaller of the two, has a median of 17% for three against three, 17% for five against five and 11% for ten against ten. One time in twenty, it exceeds 62%, 47% and 33%. Two groups of five runs drawn at random often trace two well-separated curves. In the sorted split, the median gap reaches 42%.

Keeping the best three runs out of ten. Over 10,000 draws of ten runs, the mean of the best three averages 2,249, against 1,692 for all ten: an inflation of 33%. Experiment E4 extends this measurement to the four programs (section 6.5).

The 2018 statistic, out of PPO's reach. Henderson et al. (2018) test whole learning curves. The second statistic of E1, added after the previous results (deviation 3), recomputes their average t on our thirty curves: a Welch test at each of the twenty evaluation points, then the mean of the t values (section 5.9.6). One time in twenty, the absolute value of the average t exceeds 1.4 when the groups of five are drawn at random, 2.5 when they are sorted by final score. The 20,000 draws all stay below 9. The test at the last point alone recovers the result on final scores: about 31% false positives for sorted groups.

What E1 establishes and its limits. On these thirty runs, the chance of the random seeds fools the eye far more often than the test. A Welch test on groups drawn at random stays cautious, even with three runs per group. Sorting the runs turns this chance into false positives. Selecting the best runs inflates the mean by a third. E1 covers one

Table 6.1. Experiment E1: thirty training runs of PPO PyTorch (Stable-Baselines3) on HalfCheetah-v5, same code and same hyperparameters, random seeds 0 to 29; final score: mean of 10 evaluation episodes on seeds never seen in training. Top part: statistics of the 30 scores; the Shapiro-Wilk test rejects normality (three runs take off beyond 3,000). Middle part: 10,000 draws per row (generator `default_rng(2018)`), Welch test on the final scores of two groups of the same program: every difference declared is a false positive. The apparent gap divides the difference between the two means by the smaller of the two. Groups drawn at random fool the test less often than the 5% threshold; **in bold**, sorted groups (5 best against 5 worst of 10 runs drawn), which fool it nearly one time in three. Bottom part: means over 10,000 draws of 10 runs. Source: `sim/analysis/e1.json` (`sim/analysis/e1.py`).

PPO PyTorch, HalfCheetah-v5, $n = 30$			
Final scores of the 30 runs (evaluation at 1 million steps)			
Mean; standard deviation	1,688; 568	Coef. of variation	34%
Median	1,580	Minimum; maximum	1,138; 3,430
Shapiro-Wilk, p	4×10^{-7}	IQM	see Table 6.3
Groups compared	**False positives ($p < 0.05$)**	**Median apparent gap**	**Gap exceeded 1 time in 20**
3 against 3, at random	1.6%	17%	62%
5 against 5, at random	1.2%	17%	47%
10 against 10, at random	1.7%	11%	33%
5 best against 5 worst, out of 10	**31.0%**	42%	not computed (median p 0.09)
Keeping the best 3 of 10	mean of the 3: 2,249	mean of the 10: 1,692	inflation: 33%

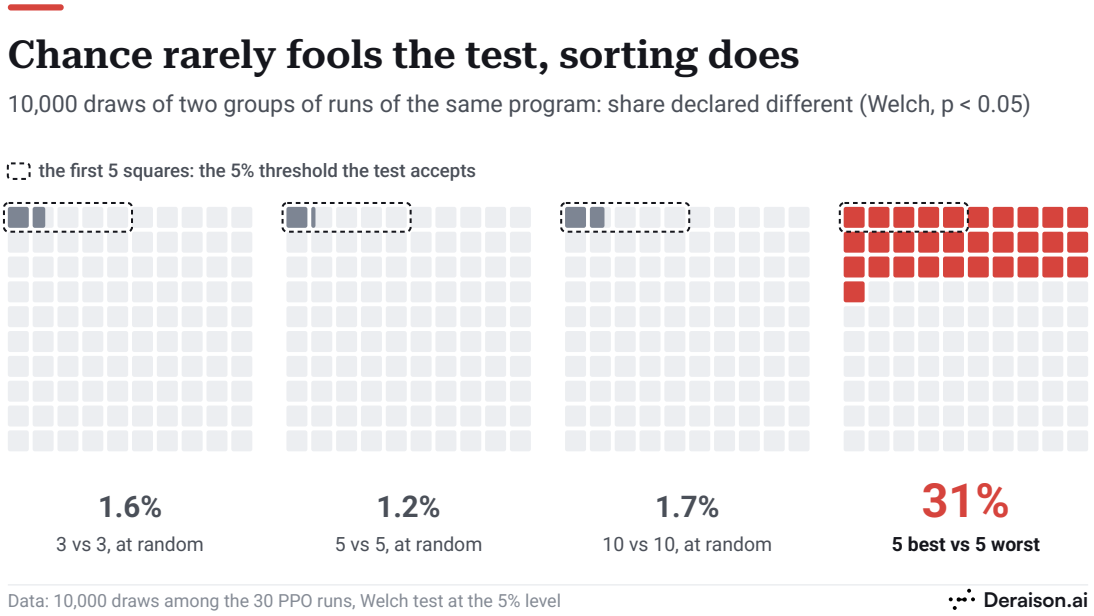


Figure 6.2. Share of the 10,000 draws in which Welch's test, at the 5% level, declares two groups of runs of the same program different (PPO PyTorch on HalfCheetah, thirty runs that differ only in their random seed). Each grid has 100 cells, one cell standing for one hundredth of the draws. The cells framed with a dotted line mark the 5% level. Drawn at random, three against three, five against five or ten against ten, the groups fool the test less often than this level. The five best set against the five worst of a draw of ten fool it nearly one time in three.

program and one task. The shape of the distribution, with a few runs that take off, depends on both. On a more symmetric distribution, the test on random groups would come closer to its nominal level.

6.2 B1: the puzzle of the t of 9

Question. Figure 5 of Henderson et al. (2018) sets two groups of five TRPO runs against each other on HalfCheetah, with the same code and the same settings. It reports $t = -9.09$ and $p = 0.0016$. Pineau stated in 2018 that the groups are not sorted. Yet E1 shows that with PPO, groups drawn at random never come near an average t of 9. Where does this figure come from? Three leads remained open: the spread specific to TRPO, another reading of their caption, or another computation (section 5.10.2).

First lead: TRPO spreads more, without nearing 9. The thirty TRPO runs, trained with the default values of sb3-contrib, spread far more than the PPO runs. Their final mean is 1,951, their standard deviation 930, or 48% of the mean, against 34% for PPO. Even so, with groups of five drawn at random, the average t over the curve exceeds 1.4 only one time in twenty, as for PPO. With sorted groups, it exceeds 1.9 one time in twenty. The rule fixed in advance (deviation 4) rules out the spread of the algorithm as the explanation.

Second lead: stacked points, a faulty computation that still falls short. The third reading of section 5.9.6, the pooled points, stacks the hundred points of the five curves of each group, as if they were independent observations. One time in twenty, its $|t|$ exceeds 5.0 with groups drawn at random, 6.3 with sorted groups. No draw reaches 9. This reading has a worse defect than its failure. It declares a difference in 53% of the random draws with TRPO and in 23% with PPO, although both groups run the same code. The twenty points of a curve are not independent: a run that starts well often stays ahead.

The published p gives the computation away. A two-sided Welch test that yields $t = 9.09$ and $p = 0.0016$ has about 3.4 degrees of freedom. This pair rules out the stacked points, which have about 198 degrees of freedom and would give a p of the order of 10^{-16}. It also rules out the average t converted into a p: two groups of five give at least 4 degrees of freedom (section 5.9.5). Only a mean of p-values remains compatible with the published pair.

The 2017 script. In June 2017, Peter Henderson published a script with a telling name, `from_same_dist.py` (Henderson 2017). Section 5.9.7 gives its exact computation. In short:

1. it averages the curves of the five random seeds of each group, point by point;
2. at each point, it compares the two mean curves with a Welch test whose variance is the variance *of the episodes* of one iteration, divided by the number of episodes;
3. it reports the mean of the t values and the mean of the p values.

The variability between random seeds, the one the figure was meant to measure, never enters the test. The script counts episodes instead of independent training runs.

The script finds the t of 9 on our TRPO runs. Applied to our thirty TRPO curves with $n = 10$ episodes, following a plan written before the computation (deviation 6), the script gives the results of table 6.2:

- With groups drawn at random, the absolute value of the average t reaches 9.09 in 38% of the draws. Its median is 7.1. The test declares a difference in 43% of the draws.
- With sorted groups, 70% of the draws reach 9.09.
- With $n = 5$, the sensitivity analysis, 23% of the random draws reach 9.09. The test then declares a difference in 25% of the draws.

Applied to our thirty PPO curves, the same computation never reaches 9.09. One time in twenty, its average $|t|$ exceeds 3.6 with random groups. Figure 6.3 shows this contrast, figure 6.4 the full distributions.

Why TRPO alone gets there. The denominator of the script is the standard deviation between episodes. The correct denominator is the standard deviation between random seeds. Let ρ be the ratio of the first to the second, measured at the last evaluation point. The smaller ρ, the larger the t of the script compared with the t of a test on random seeds, roughly as $1/\rho$. For our TRPO, the standard deviation between the episodes of one agent is only 7% of the standard deviation between agents. For our PPO, it is 30%. A TRPO agent scores almost the same at every episode, while two TRPO agents differ a great deal. In this case, counting episodes instead of random seeds manufactures huge t values.

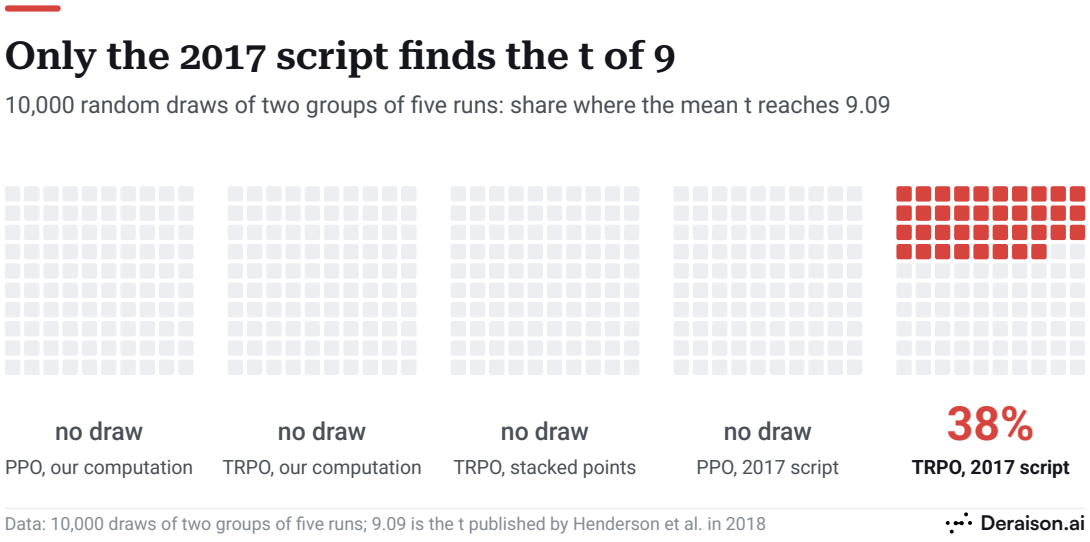


Figure 6.3. Two groups of five runs drawn at random from thirty, 10,000 times, for each algorithm and each way of computing the average t along the learning curve. Each grid has 100 cells, one cell standing for one hundredth of the draws. It colours the share of draws in which the average t, sign ignored, reaches 9.09, the value Henderson and his coauthors published in 2018 for TRPO on HalfCheetah. Only the script they published in 2017, which tests the spread between episodes instead of the spread between random seeds, gets there, applied to TRPO: 38% of the draws. PPO with this script, PPO and TRPO with our test on random seeds, TRPO with stacked points: no draw at all.

Table 6.2. Experiment B1: the statistic of the 2018 figure ($t = -9.09$, Henderson et al., TRPO on HalfCheetah, two unsorted groups of five runs) recomputed on our thirty PPO runs (those of E1) and our thirty TRPO runs (sb3-contrib, default values), HalfCheetah-v5, one million steps, 20 evaluation points. Each row: 10,000 draws of 10 runs out of 30, split into two groups of 5, at random or sorted by final score (generator `default_rng(2018)`). Three readings of 'the average t-test across the entire training curve' (deviation 4 of `sim/SIM_PROTOCOL.md`, written before any TRPO run): Welch t at each point then averaged, turned into a p with the mean degrees of freedom; mean of the p values; the 5 runs $\times$ 20 points of each group stacked as 100 independent observations. Then the script `from_same_dist.py` Henderson published in 2017 (deviation 6): a Welch test at each iteration on the standard deviation between the episodes of one point, mean of the t and p values; the spread between seeds never enters it. Columns: share of draws declared different ($p < 0.05$, false positives since both groups run the same code), 95th percentile of $|t|$, share of draws where $|t|$ reaches 9.09. Shares in percent rounded to the unit, to one significant digit below 1%. **In bold**, the only rows where $|t|$ reaches 9.09: the 2017 script applied to TRPO, whose standard deviation between episodes is only 7% of the standard deviation between seeds (PPO: 30%). Coefficient of variation of the final scores: TRPO 48%, PPO 34%. Sources: `sim/analysis/b1_trpo.json`, `sim/analysis/b1_script.json` (the PPO mean-t row repeats the computation of `sim/analysis/e1_curve.json`).

		Groups drawn at random			Sorted groups		
Algorithm	Computing t	false pos. (%)	$\lvert t\rvert$, 95th pct	$\lvert t\rvert \geq 9.09$ (%)	false pos. (%)	$\lvert t\rvert$, 95th pct	$\lvert t\rvert \geq 9.09$ (%)
PPO	mean t along the curve (seeds)	0.2	1.4	0	6	2.5	0
	mean p along the curve (seeds)	0	1.4	0	0.2	2.5	0
	stacked points (100 per group)	23	3.0	0	86	4.4	0
	Henderson's script, $n = 10$ episodes	2	3.6	0	7	5.3	0
	Henderson's script, $n = 5$	0.1	2.5	0	0.9	3.8	0
TRPO	mean t along the curve (seeds)	0.04	1.4	0	1	1.9	0
	mean p along the curve (seeds)	0	1.4	0	0.06	1.9	0
	stacked points (100 per group)	53	5.0	0	84	6.3	0
	Henderson's script, $n = 10$ episodes	**43**	**22.3**	**38**	**51**	**28.2**	**70**
	Henderson's script, $n = 5$	**25**	**15.8**	**23**	**31**	**19.9**	**49**

What remains uncertain. We cannot prove that this script produced figure 5. Its name, its output ('t=..., p=...') and its compatibility with the published (t, p) pair make it likely. Our conditions differ from theirs, as we stated before the computation. Our evaluation episodes are played by a deterministic policy. Their progress files record the training episodes of a stochastic policy, which probably vary more. We have twenty evaluation points instead of their iterations. Our TRPO comes from sb3-contrib, while their code was different. The order of magnitude comes back despite these differences.

What B1 establishes. The t of 9 of 2018 needs no sorting of the runs to appear. A test that counts episodes instead of random seeds produces it on our own TRPO runs. Flaw F11, same code, another random seed, goes deeper than the figure showed: the chance of the random seeds fooled the test that was meant to measure it. Figure 6.5 brings E1 and B1 together against the published figure.

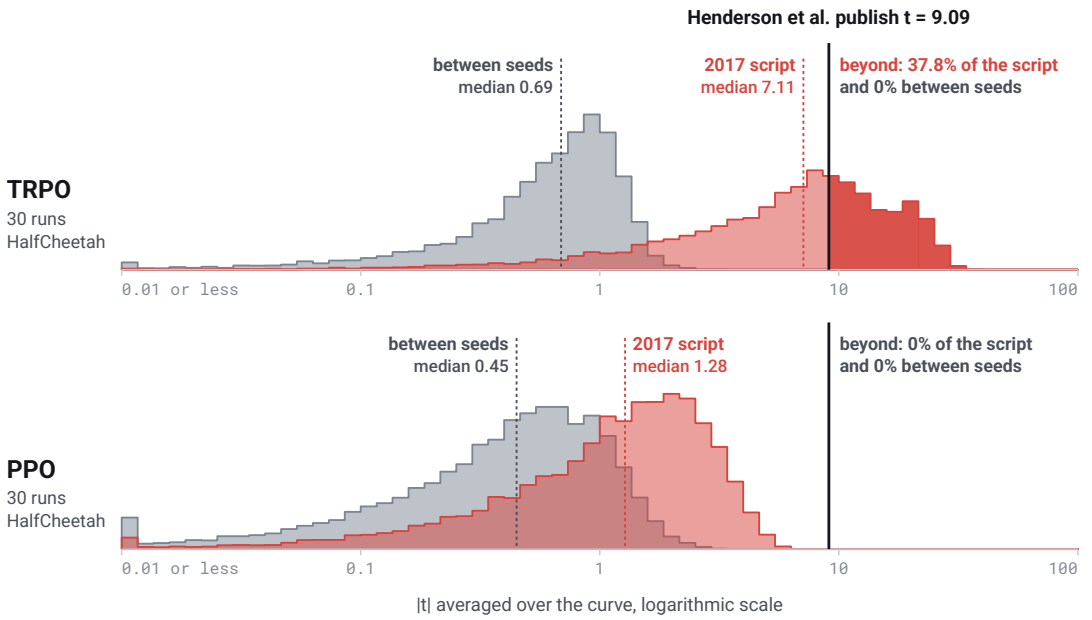


Figure 6.4. Distribution of the absolute value of the average t over 10,000 draws of two groups of five runs taken at random from thirty, on HalfCheetah, for TRPO (top) and for PPO PyTorch (bottom), on a logarithmic scale. In grey, the Welch test computed between random seeds at each point of the curve, then averaged. In red, the computation of Henderson's script (2017), which averages the curves of each group and tests the spread between the ten evaluation episodes. The vertical line marks 9.09, the value published in 2018. Only the script applied to TRPO exceeds it, in 38% of the draws.

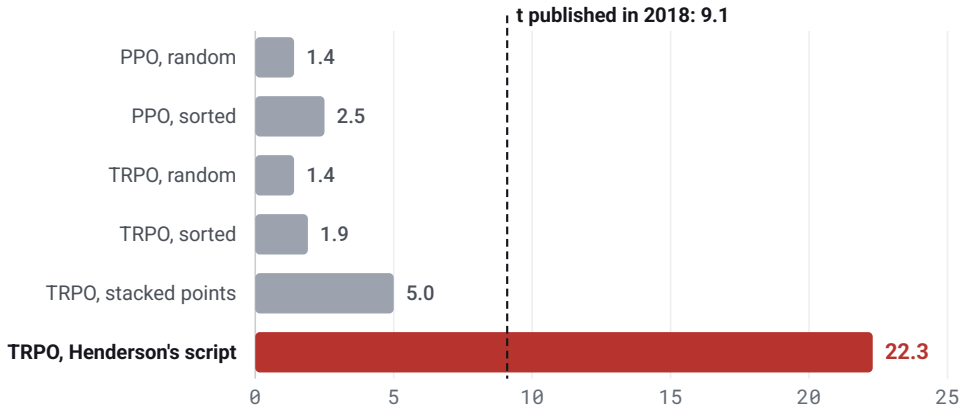
6.3 E2: the leading program changes from task to task

Question and plan. Henderson et al. (2018) showed that the algorithm leading on one task stopped leading on another. E2 redoes the comparison with four programs of 2026 (PPO PyTorch, PPO JAX, SAC, CrossQ) on the four tasks, ten random seeds each, at each program's budget: one million steps for PPO, 200,000 for SAC and CrossQ (section 5.10.3).

Ten random seeds: four tasks, three leading programs. Table 6.3 gives the IQM of each program on each task with its interval. Figure 6.6 divides each IQM by the best one of its task.

Only Henderson's script exceeds the 2018 t: the spread between seeds cannot reach it

HalfCheetah, two groups of five runs out of thirty: 95th percentile, over 10,000 draws, of the absolute mean t along the learning curve



Data: sim/analysis/e1_curve.json, b1_trpo.json, b1_script.json
(docs/figures/data/e1_henderson_en.csv)

Deraison.ai

Figure 6.5. Our thirty PPO runs and our thirty TRPO runs on HalfCheetah, compared with the figure published in 2018. For each way of splitting ten runs into two groups of five, at random or sorted by final score, the bar gives the absolute value of the average t along the learning curve exceeded one time in twenty, over 10,000 draws. The stacked-points bar treats the hundred curve points of each group as independent observations. The bar for Henderson's script applies his 2017 computation, which tests the spread between episodes, to our TRPO runs. The bar for Henderson and his coauthors is the value they publish on their own runs.

- HalfCheetah: SAC leads with 5,191 [4,431, 5,601], ahead of CrossQ, 4,801 [2,427, 7,815]. The two PPOs follow far behind, 1,407 for PPO PyTorch and 1,254 for PPO JAX.
- Hopper: PPO JAX leads with 3,064, ahead of PPO PyTorch (2,877), CrossQ (2,302) and SAC (1,308).
- Walker2d: CrossQ leads with 5,414 [4,994, 5,627], far ahead of PPO PyTorch (2,158), PPO JAX (1,272) and SAC (1,236).
- Swimmer: PPO JAX leads with 112, ahead of CrossQ (110) and PPO PyTorch (107). SAC stays at 47.

A settled change, a lead that is rarely clear. The leading program differs from task to task. The change is settled under the protocol's rule: SAC, leading on HalfCheetah, is beaten in a settled way on Hopper by both PPOs, on Walker2d by CrossQ and PPO PyTorch, on Swimmer by the other three. `e2.py` counts 14 settled changes. The leading program, however, beats all the others in a settled way only on Walker2d. On the other three tasks, ten random seeds do not separate it from the runner-up: SAC from CrossQ on HalfCheetah, PPO JAX from PPO PyTorch on Hopper, PPO JAX from CrossQ on Swimmer. The interval of CrossQ on HalfCheetah, from 2,427 to 7,815, shows how much its runs vary on this task.

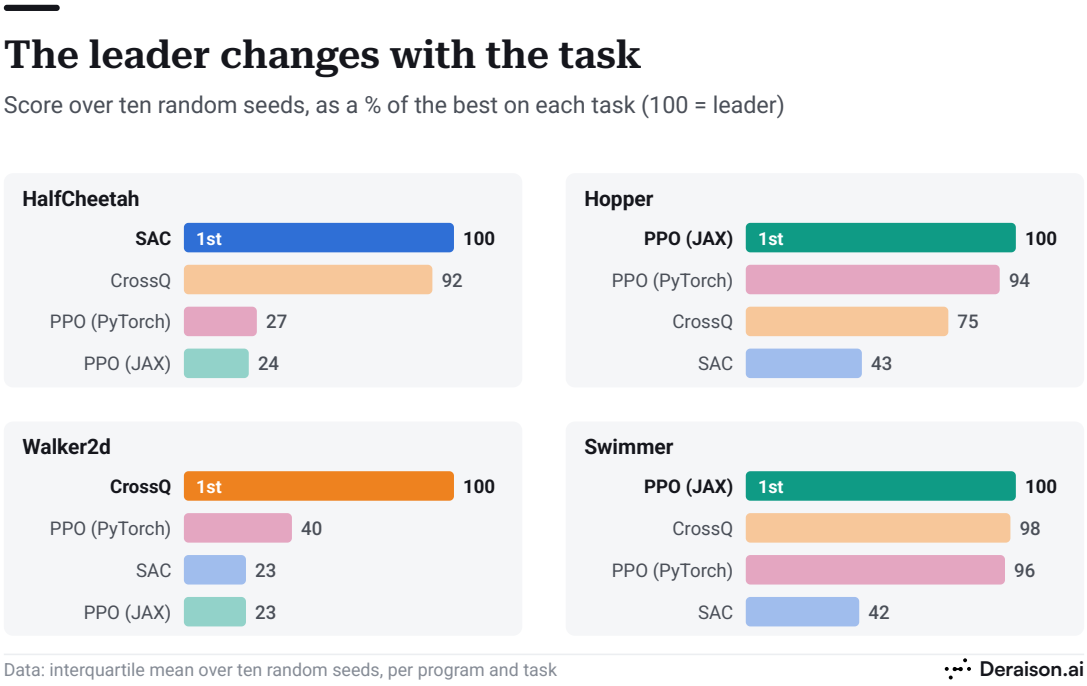


Figure 6.6. Score of each program divided by the best score of its task, over ten random seeds: the bar at 100 marks the leading program. The score of a program is the interquartile mean (the mean of the middle six scores out of ten) of its final evaluation scores. PPO trains for one million steps, SAC and CrossQ for 200,000. The leading program changes: SAC on HalfCheetah, PPO written in JAX on Hopper and Swimmer, CrossQ on Walker2d.

With Bonferroni's correction. E2 computes 24 probability-of-improvement intervals with no correction. Raised to a simultaneous 95% level, that is 99.79% each (deviation 18), they leave 12 settled changes instead of 14 (table 6.4). Two pairs lose their status: PPO JAX against CrossQ on Hopper, PPO PyTorch against SAC on Walker2d. The verdict of E2 survives the correction.

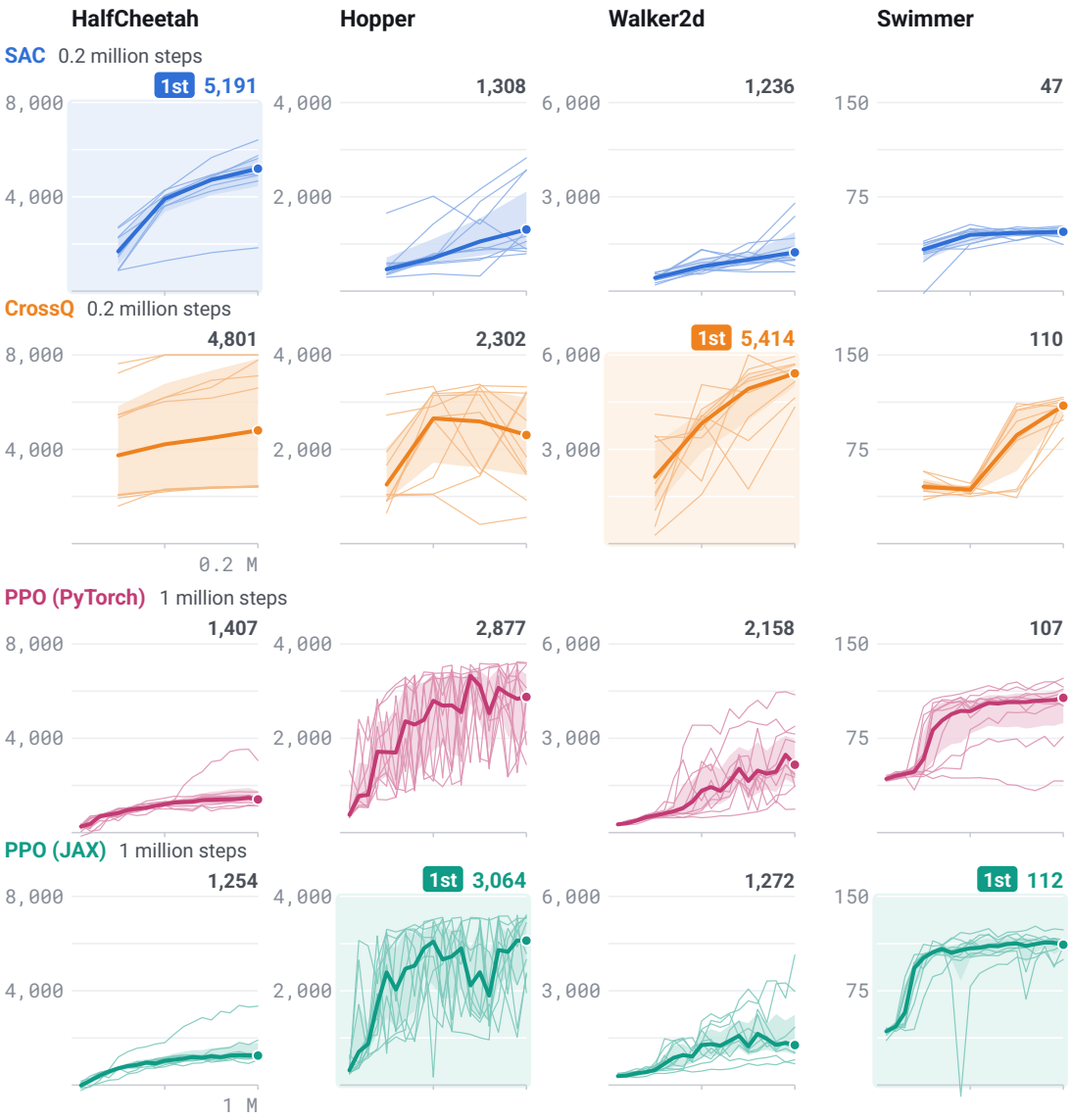
The learning curves. Figure 6.7 shows the evaluation curves of the sixteen program-task pairs. It recalls what a final ranking hides: the curves of SAC and CrossQ stop at 200,000 steps, those of PPO continue to one million. The equal-budget reading, below, compares the four programs at the same point of these curves.

Twenty random seeds. With random seeds 0 to 19, the leading program still changes with the task: CrossQ leads on HalfCheetah and Walker2d, PPO JAX on Hopper and Swimmer. There remain 10 settled changes, 6 with Bonferroni's correction. Three divergences from the results at ten random seeds need stating.

- On HalfCheetah, SAC yields the lead to CrossQ: its IQM goes from 5,191 to 4,214, that of CrossQ from 4,801 to 4,694. The gap between the two is settled neither at ten nor at twenty random seeds.
- On Hopper, CrossQ rises from third to second place (2,302, then 2,792). The lead of PPO JAX over CrossQ is no longer settled.
- The leading program now beats all the others in a settled way on two tasks, Walker2d and Swimmer.

The leading program changes from task to task

Evaluation score along training, per program and task. Thin line: one random seed. Thick line: interquartile mean of the ten seeds, with its 95% band.



Number above each panel: interquartile mean at the last step, the one the ranking uses; tinted ground for the leader of the task. One vertical scale per task, starting at zero. SAC and CrossQ train for 0.2 million steps, PPO for one million.

Data: sim/results, seeds 0 to 9; band: percentile bootstrap of the interquartile mean (numpy, 2,000 draws, seed 2026)

Deraison.ai

Figure 6.7. Evaluation curves of the four programs (PPO PyTorch, PPO JAX, SAC, CrossQ) on the four tasks (one task per column). The horizontal axis gives the training steps, one point every 50,000 steps: up to one million for PPO, 200,000 for SAC and CrossQ. The vertical axis gives the mean return of ten evaluation episodes played without randomness. The solid line is the interquartile mean of the ten random seeds, the band its 95% bootstrap interval, the faint lines the individual runs.

Table 6.3. Experiment E2: interquartile mean (IQM) of the final score of each program on each MuJoCo task (Gymnasium 1.1.1), with its 95% stratified bootstrap confidence interval (rliable, 2,000 resamples, `np.random.seed(2026)`). Final score: mean of 10 evaluation episodes on seeds never seen in training, at 1 million steps for PPO and 200,000 for SAC and CrossQ (deviation 1). 10 seeds (0 to 9): the main, preregistered result; 20 seeds (0 to 19) and equal budget (PPO read at 200,000 steps on its evaluation curve): the consolidation of deviation 12, computed on 21 November 2026 (deviation 20). The SAC and CrossQ columns are the same at equal and at own budget. **In bold**, the leader of each task (highest IQM, no test). Last row: settled changes of leader, when the leader of one task is beaten by another program on another task with a probability-of-improvement interval that excludes 0.5; in parentheses, with the 24 intervals made simultaneous by Bonferroni (deviation 18, Table 6.4). Sources: `sim/analysis/e2.json`, `e2_s20.json`, `e2_equal.json`, `e2_equal_s20.json`, `e2_bonferroni.json`, `e2_bonferroni_s20.json`.

Task, program	10 seeds	20 seeds	Equal budget, 10	Equal budget, 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]**
Settled changes of leader	14 (12)	10 (6)	3	0

At equal budget. PPO receives five times more steps than SAC and CrossQ. Read at 200,000 steps on its curve (deviation 12, item iii), PPO no longer leads on any task.

- With ten random seeds, SAC leads on HalfCheetah, CrossQ on the other three tasks. There remain 3 settled changes: SAC, leading on HalfCheetah, is beaten in a settled way by CrossQ on Walker2d and on Swimmer, then by PPO JAX on Swimmer. On Hopper, no pair is settled.
- With twenty random seeds, CrossQ leads on all four tasks and no change is settled. On HalfCheetah, its lead over SAC (4,694 against 4,214) is not settled either.

What E2 establishes and its limits. The preregistered verdict holds: with ten random seeds and at the protocol's budgets, the leading program changes from task to task, in a settled way, even after Bonferroni's correction. Two qualifications weigh on it. The leading program rarely beats all the others clearly. With twenty random seeds, the change of leader from one task to the next owes much to PPO's budget, five times

Table 6.4. Experiment E2, pairwise comparisons: probability of improvement $P(A > B)$, the probability that a run of A beats a run of B on the task, with its stratified bootstrap interval (rliable, 2,000 resamples). 95% CI: the interval of each comparison alone, the one of the preregistered rule (deviation 7); Bonferroni CI: the 24 intervals (6 pairs on 4 tasks) made simultaneous at the 95% level, each at level $1 - 0.05/24$, that is 99.79% (deviation 18 of `sim/SIM_PROTOCOL.md`, written after the E2 results and before this computation). Pairs are oriented in the order of the ten-seed ranking; †: at twenty seeds the ranking flips and the file gives $P(B > A)$, reported as it is for the flipped pair. **In bold**, intervals that exclude 0.5: settled comparison. Sources: `sim/analysis/e2.json`, `e2_s20.json`, `e2_bonferroni.json`, `e2_bonferroni_s20.json`.

	10 seeds			20 seeds		
Task, pair $A > B$	$P(A > B)$	95% CI	Bonferroni CI	P	95% CI	Bonferroni CI
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]**

larger: at equal budget, CrossQ leads everywhere. A paper that compares two methods at different budgets also compares these budgets. Finally, the ranking applies to the libraries' default values. Tuning per task might change it, in a direction we do not know: experiment E6 shows, on one case, a published tuning doing worse than the default values.

6.4 E3: two programs of the same algorithm

Question and plan. Do two programs that implement the same algorithm give the same result? E3 sets PPO PyTorch against PPO JAX on the four tasks, with the same random seeds, the same default values and one million steps (section 5.10.4). The two programs differ in their numerical library and in how they initialise the weights (section 5.4).

Ten random seeds: no winner, wide gaps. Table 6.5 gives the two IQMs, their ratio and the probability that a PPO JAX run beats a PPO PyTorch run. On no task does one version win under the settled-winner rule. Yet the apparent gaps are wide. On Walker2d, the IQM is 2,158 with PyTorch and 1,272 with JAX, 41% lower. The probability that JAX does better is 0.27, from 0.06 to 0.52: the interval still touches 0.5. The two IQM intervals overlap. On the other three tasks, the probabilities of improvement stay close to 0.5: 0.36 on HalfCheetah, 0.54 on Hopper, 0.66 on Swimmer. The IQM intervals of `e3.json` differ by a few points from those of `e2.json` for the same runs, because each script consumes the random generator in its own order (section 5.9.10).

Twenty random seeds: the two versions part ways. Of all the results, this one moves most when random seeds are added. With twenty random seeds, two tasks have a settled winner:

Table 6.5. Experiment E3: two implementations of the same algorithm, Stable-Baselines3's PPO (PyTorch) and SBX's PPO (JAX), same seeds, final score at 1 million steps. IQM of the final score with its 95% stratified bootstrap interval (rliable, 2,000 resamples); relative gap between the two IQMs; probability that a JAX run beats a PyTorch run, with its bootstrap interval. Preregistered rule: a winner is declared only when the two IQM intervals do not overlap, or when the interval of the probability of improvement excludes 0.5. **In bold**, the tasks where a winner is declared: none at ten seeds, two at twenty, where the gaps resemble those at ten seeds with narrower intervals. Sources: `sim/analysis/e3.json`, `sim/analysis/e3_s20.json`.

Task	PPO PyTorch IQM [95% CI]	PPO JAX IQM [95% CI]	JAX/PyTorch − 1	$P(\text{JAX} > \text{PyTorch})$ [95% CI]	Winner
10 seeds (0 to 9), main result					
HalfCheetah	1,407 [1,251, 1,776]	1,254 [1,171, 1,830]	−11%	0.36 [0.11, 0.63]	none
Hopper	2,877 [2,232, 3,339]	3,064 [2,489, 3,536]	+6%	0.54 [0.28, 0.80]	none
Walker2d	2,158 [1,543, 3,005]	1,272 [973, 2,180]	−41%	0.27 [0.06, 0.52]	none
Swimmer	107 [88, 113]	112 [109, 115]	+4%	0.66 [0.39, 0.89]	none
20 seeds (0 to 19), deviation 12					
HalfCheetah	1,515 [1,404, 1,665]	1,341 [1,228, 1,800]	−11%	0.38 [0.20, 0.56]	none
Hopper	2,668 [2,107, 3,227]	2,864 [2,411, 3,257]	+7%	0.52 [0.33, 0.70]	none
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**

- On Walker2d, PyTorch wins, 2,204 against 1,435. The probability that JAX does better is 0.30, from 0.15 to 0.48.
- On Swimmer, JAX wins, 115 against 107. The probability is 0.79, from 0.63 to 0.91, with disjoint IQM intervals.

The gaps resemble those at ten random seeds. The ten added random seeds narrow the intervals.

What E3 establishes and its limits. With ten random seeds, we cannot say that the two versions of PPO differ. With twenty, they differ on two tasks, in opposite directions. A method compared with 'PPO' on Walker2d would win or lose depending on the version chosen. Flaw F14, the implementation that changes the result, still applies in 2026 between two libraries by the same author with identical default values. E3 does not identify which code difference causes it: the weight initialisation, the random generators or the rounding of the numerical library.

6.5 E4: the best three out of ten

Question and plan. What happens to a result when only the best three of ten runs are published, a practice Henderson et al. (2018) pointed out? E4 applies this selection to each of the sixteen program-task pairs of E2, then counts the comparisons, out of 24, that change direction (section 5.10.5).

A mean always inflated. Table 6.6 and figure 6.8 give the inflation for each pair. All sixteen inflations are positive, by construction. The mean inflates by 32% at the median, from 6% to 84% depending on the program and the task. The largest hit PPO JAX on Walker2d (84%), SAC on Hopper (78%) and CrossQ on HalfCheetah (74%). The standard deviation of the best three has a median of 35% of that of all ten: the selection inflates the mean while it narrows the apparent spread, which makes the result look more certain.

Three comparisons reversed. Out of 24 comparisons, 3 change direction when each program keeps only its best three runs: PPO PyTorch against PPO JAX on HalfCheetah and on Hopper, CrossQ against PPO PyTorch on Swimmer. Two of the three set the two versions of PPO against each other. When the gap is small, keeping one's best runs changes the winner.

Replication on random seeds 10 to 19. Repeated identically on ten other random seeds, the selection inflates the mean by 43% at the median, from 7% to 85%. It again reverses 3 comparisons out of 24: CrossQ against PPO PyTorch on Hopper, PPO PyTorch against PPO JAX on Hopper, CrossQ against PPO PyTorch on Swimmer. Two of the three reversals are the same as on random seeds 0 to 9.

What E4 establishes. Selecting the best runs always inflates the mean, by about a third at the median. It reverses close comparisons. The result replicates on independent random seeds. E4 measures the effect of a practice. How many papers follow the practice is a question for the survey (criterion X3, chapter 4).

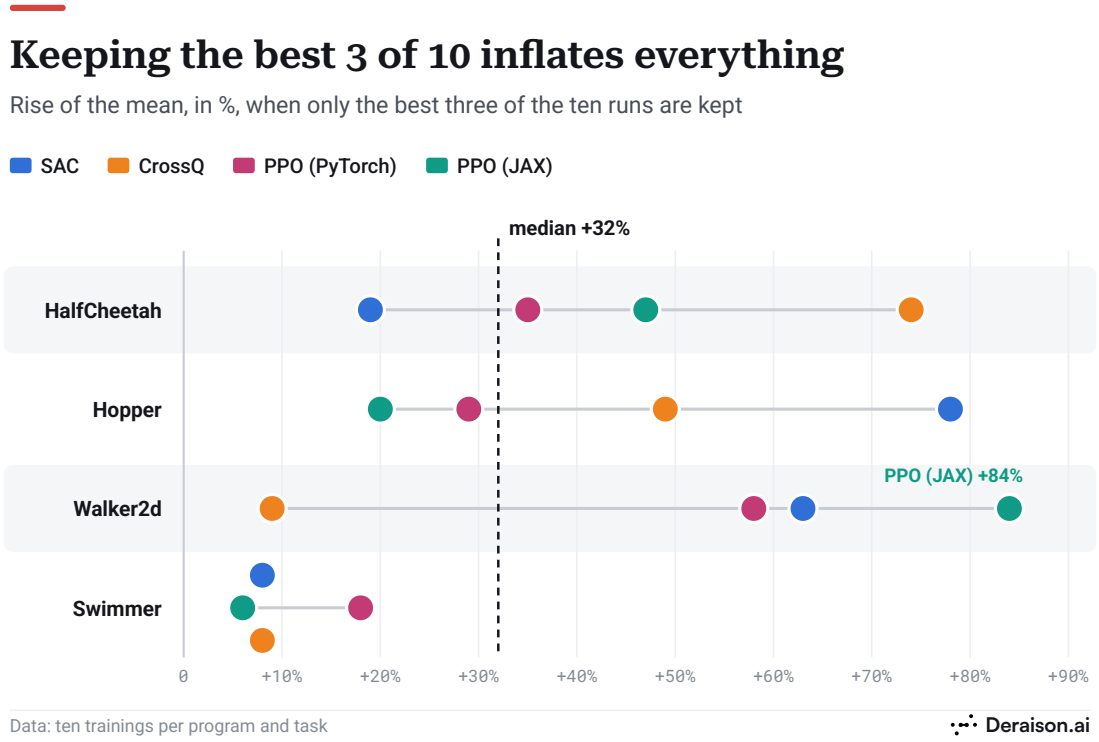


Figure 6.8. Rise of the mean when only the best three of ten runs are published, for each program (one colour) and each task (one row), as a percentage of the mean of all ten. The dotted line marks the median, +32%. All sixteen rises are positive: sorting always serves the one who sorts. Data: ten random seeds per pair, final evaluation scores.

6.6 E5: how many random seeds?

Question and plan. How many random seeds does it take to recover the correct order of two programs? E5 draws n runs per program with replacement from those observed, 10,000 times, then counts the share of draws that recover the order of the means of all the runs (equation (5.8), section 5.10.6).

Ten random seeds. Over the 24 comparisons, the median share of draws that recover the correct order is 94% with three random seeds, 97% with five and 99% with ten (table 6.7). This reassuring figure owes much to the wide gaps between our programs. Seven comparisons are close, with means that differ by less than 20%. For them, the median falls to 65% with three random seeds, 71% with five, 77% with ten and 85% with twenty. The hardest comparison, CrossQ against SAC on HalfCheetah, recovers the correct order in only 60% of the draws with five random seeds.

Twenty random seeds. The correct order becomes that of the means of twenty runs. The draws are made among the twenty. Over the 24 comparisons, the median barely moves: 93% with three random seeds, 97% with five. The close comparisons now number eight. They are harder than ten random seeds suggested: the median is 59%, 62%, 67% and 74% with three, five, ten and twenty random seeds. CrossQ and PPO PyTorch on Swimmer have means that differ by less than 1%: even twenty random seeds order them in only 53% of the draws, about as often as a coin toss.

Table 6.6. Experiment E4: keeping the best 3 runs out of 10. For each program and task, mean and standard deviation of the final scores of the 10 runs, then of the best 3, and the inflation of the mean (mean of the best 3 divided by that of the 10, minus 1). A comparison (a pair of programs on one task, 24 in all) reverses when the sign of the gap between means changes once each program keeps only its best 3 runs (deviation 7 of `sim/SIM_PROTOCOL.md`). On the right, the same analysis redone on seeds 10 to 19 (deviation 12). Reversed comparisons, seeds 0 to 9: HalfCheetah: PPO PyTorch / PPO JAX; Hopper: PPO PyTorch / PPO JAX; Swimmer: CrossQ / PPO PyTorch; seeds 10 to 19: Hopper: CrossQ / PPO PyTorch; Hopper: PPO PyTorch / PPO JAX; Swimmer: CrossQ / PPO PyTorch. The standard deviation of the best 3 falls at the median to 35% of that of the 10. Sources: `sim/analysis/e4.json`, `sim/analysis/e4_r10.json`.

		Seeds 0 to 9 (main result)					Seeds 10 to 19 (replication)		
Task	Program	mean 10	sd 10	mean 3	sd 3	infl.	mean 10	mean 3	infl.
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%
Median inflation (range)				+32% (+6% to +84%)			+43% (+7% to +85%)		
Comparisons reversed					3 / 24				3 / 24

What E5 establishes and its limits. Five random seeds, the median value the survey finds in the papers of 2026 (chapter 4), suffice for wide gaps. They do not suffice for gaps under 20%: with twenty random seeds as the reference, such a gap is read backwards more than one time in three. E5 lets the observed runs stand in for the true distribution. With ten or twenty runs, the stand-in is uncertain, so for the closest comparisons the correct order is uncertain too.

6.7 E6: a baseline that looked a little too good

Question and plan. E6 was meant to illustrate flaw F7: the gap a new method credits to itself when it alone is tuned. On Hopper-v5, E6 sets ten PPO PyTorch runs with the default values of Stable-Baselines3 against ten runs with the settings published by RL Baselines3 Zoo, the reference collection for this library (section 5.10.7). Table 6.8 gathers every group.

First result: the published settings would do three times worse. The default version reaches an IQM of 2,877 [2,268, 3,369]. The tuned version reaches 1,003 [856, 1,146]. In

Table 6.7. Experiment E5: how many seeds recover the right ranking? For each E2 comparison, 10,000 draws with replacement of n runs per program from its runs (the observed distribution stands in for the program), then the share of draws whose means give the right order, in percent. Right order: that of the means of all the chosen runs (10, or 20 with deviation 20); at ten seeds, PPO PyTorch on HalfCheetah draws from the 30 runs of E1, as preregistered. Gap: relative difference of the two means, divided by the larger in absolute value. **In bold**, close comparisons (gap under 20%); the last row gives their number and the median of their shares. Sources: `sim/analysis/e5.json`, `sim/analysis/e5_s20.json`.

		10 seeds (main result)					20 seeds (deviations 12 and 20)				
Task	Pair	gap	$n = 3$	5	10	20	gap	$n = 3$	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
Median, 24 comparisons			94	97	99	100		93	97	100	100
Median, close comparisons		7	65	71	77	85	8	59	62	67	74

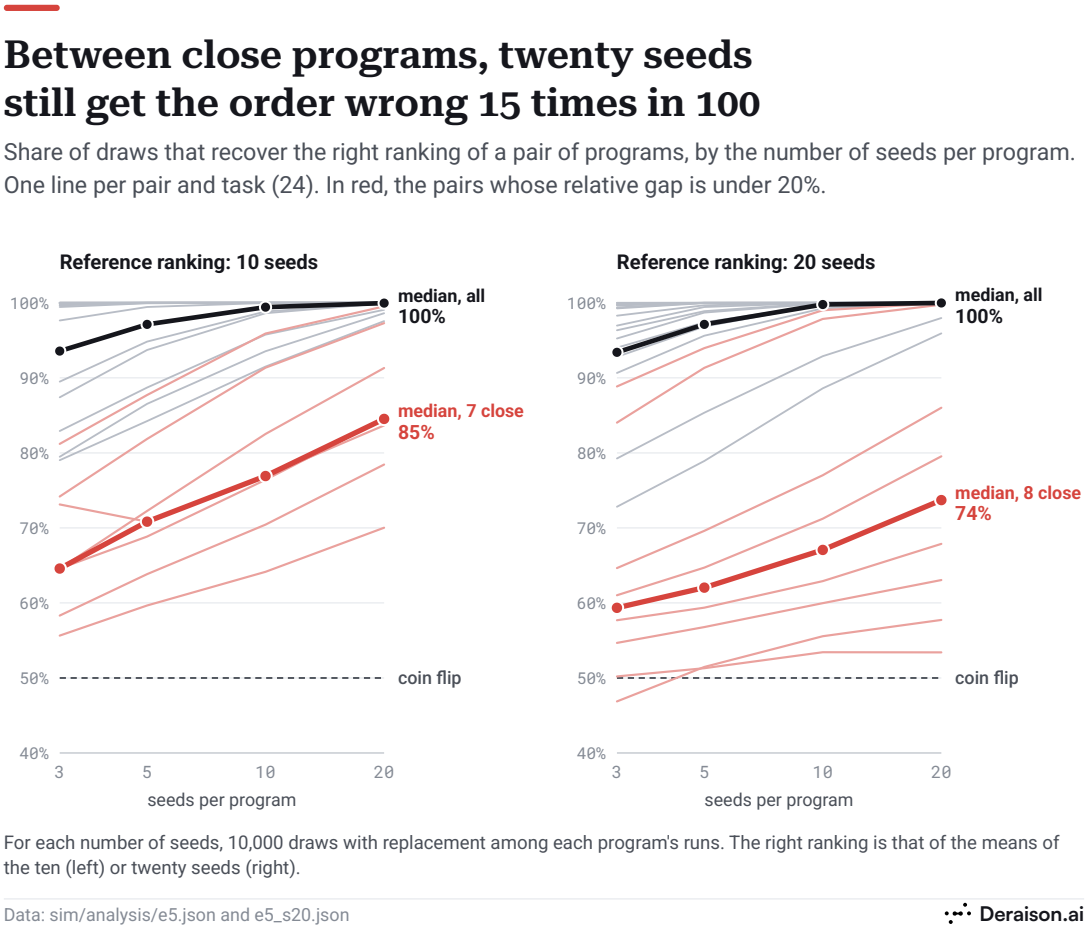


Figure 6.9. Share of the 10,000 draws that recover the correct order of two programs, as a function of the number of random seeds drawn for each (3, 5, 10, 20). The correct order is that of the means of all the observed runs. One curve per comparison between two programs on one task, 24 in all. In red, the close comparisons, whose means differ by less than 20%. Two panels: the correct order and the draws come from ten random seeds, then from twenty. With twenty random seeds as the reference, five recover the correct order of the close comparisons in only 62% of the draws at the median.

means, 2,776 against 994, with a plateau near 1,000 reached as early as 50,000 steps. The published settings would do nearly three times worse than the default values.

An error on our side that explains nothing. The result was too good to believe without a check. We found an error in our code. The Zoo's reward normalisation uses the algorithm's discount factor, 0.999. Our first pass left it at its default value, 0.99 (deviation 2). The diagnosis, decided after this first result, reruns the corrected recipe on Hopper-v5 and on Hopper-v4, plus the default values on v4, ten random seeds each.

- The corrected recipe obtains 1,012 [902, 1,135] on v5 and 984 [773, 1,002] on v4, the version the collection was tuned for.
- The default values obtain 2,423 [1,657, 3,058] on v4.
- The probability that a corrected run beats a run with the error is 0.47, from 0.21 to 0.73: the error changes nothing detectable.

- On v4, a corrected Zoo run beats a default run with a probability of only 0.07, from 0 to 0.24. The gap survives Holm's correction ($p = 0.001$). On v5, the same probability is 0.01.

Our error was real, yet it does not explain the gap. The version does not explain it either: on v4 as on v5, the recipe plateaus near 1,000.

The official script plateaus too. Two explanations remained: the published recipe really does worse than the default values with today's libraries, or our way of applying it still differs from the collection's. Deviation 5, written before any official training, decides between them with the collection's own script. The ten policies trained by `rl_zoo3.train` on Hopper-v4, evaluated with our protocol, obtain 995 $[821, 1{,}006]$. This matches our corrected recipe, 984 $[777, 1{,}002]$ in this analysis: the probability that an official policy beats ours is 0.58, from 0.32 to 0.82. The default values obtain 2,423 $[1{,}687, 3{,}058]$. An official policy beats a default policy with a probability of only 0.09, from 0 to 0.30. The gap survives Holm's correction ($p = 0.001$). The intervals of 984 and 2,423 differ by a few points between the two analyses, because each draws its own resamples.

What E6 establishes. Under the reading rule written in advance, our application of the recipe was not at fault. With Stable-Baselines3 2.9.2, Gymnasium 1.1.1 and MuJoCo 4.0.0, the recipe tuned and published by the collection does worse than the default values on Hopper, on version 4 as on version 5. E6 therefore does not show a baseline its authors tuned badly. E6 shows a published tuning that does worse today than the default values of the same library. The most likely explanation is that the tuning has aged with the software around it. E6 also recalls that a result that looks too good deserves a check that goes as far as the official script.

Limits. We do not know which library change made the recipe age. We tried only one task and one algorithm of the collection. The plateau near 1,000 roughly matches what the bonus given to a hopper for staying upright yields over 1,000 steps: it suggests an agent that stands without moving forward. We did not check this on the videos of each policy.

6.8 E7: the cost of a modified physics

Question and plan. Pineau asked that training be kept apart from evaluation. Our agents are already evaluated on random seeds never seen in training. E7 goes further: each final policy also plays in a modified physics, with masses made 20% heavier and sliding friction reduced by 20%. For each run, we compute the share of score kept, the ratio of the score in the modified physics to the score in the training physics, then its IQM over the ten random seeds with its interval (section 5.10.8).

Twelve pairs out of sixteen lose. Table 6.9 and figure 6.10 give the share kept for each program-task pair. Twelve pairs out of sixteen lose a settled share of their score: their interval stays below 1. Three change by no detectable amount. One gains: SAC on Swimmer, at 1.07 $[1.02, 1.12]$. The loss depends on the program and the task.

- On Hopper, PPO keeps only 22% of its score in its PyTorch version and 18% in its JAX version, while SAC keeps 64% and CrossQ 65%.

Table 6.8. Experiment E6, the baseline: PPO PyTorch on Hopper, one million steps, 10 seeds per group, Stable-Baselines3 default values against the settings published by RL Baselines3 Zoo. The first pass (preregistered, `sim/analysis/e6.json`, bootstrap with 2,000 resamples) normalised observations with a gamma of 0.99 instead of 0.999; the diagnosis decided after that result (deviation 2) reruns the corrected recipe on Hopper v5 and v4, the version the Zoo was tuned for (`e6_diag.json`, 5,000 resamples, Holm over three comparisons); then ten agents trained with the Zoo's official script, evaluated with our protocol (deviation 5, written before any official training; `e6_official.json`, 2,000 resamples, Holm over two comparisons). The bounds of one group therefore vary from one file to the next at the Monte Carlo level. IQM with its 95% bootstrap interval (rliable); $P(A > B)$: probability that a run of A beats a run of B, bootstrap interval; Welch test, Holm correction, Mann-Whitney test where the script computes it. **In bold**, comparisons that pass the Holm correction: the published recipe does worse than the default values, with our code as with the official script. Mean and standard deviation of the official agents computed here from their ten scores (`e6_official.json`).

Group (PPO PyTorch, 10 seeds)	mean	sd	IQM [95% CI]	source
Default values, Hopper-v5	2,776	752	2,877 [2,268, 3,369]	`e6_diag`
Zoo, first pass (gamma 0.99), v5	994	285	1,003 [856, 1,146]	`e6_diag`
Zoo corrected (gamma 0.999), v5	1,021	300	1,012 [902, 1,135]	`e6_diag`
Default values, Hopper-v4	2,360	911	2,423 [1,657, 3,058]	`e6_diag`
Zoo corrected, v4	877	309	984 [773, 1,002]	`e6_diag`
Official Zoo script, v4	898	310	995 [821, 1,006]	`e6_official`
Zoo corrected, v4			984 [777, 1,002]	`e6_official`
Default values, Hopper-v4			2,423 [1,687, 3,058]	`e6_official`
Default values, Hopper-v5, preregistered			2,877 [2,278, 3,368]	`e6`
Zoo, first pass (gamma 0.99), v5, preregistered			1,003 [855, 1,149]	`e6`

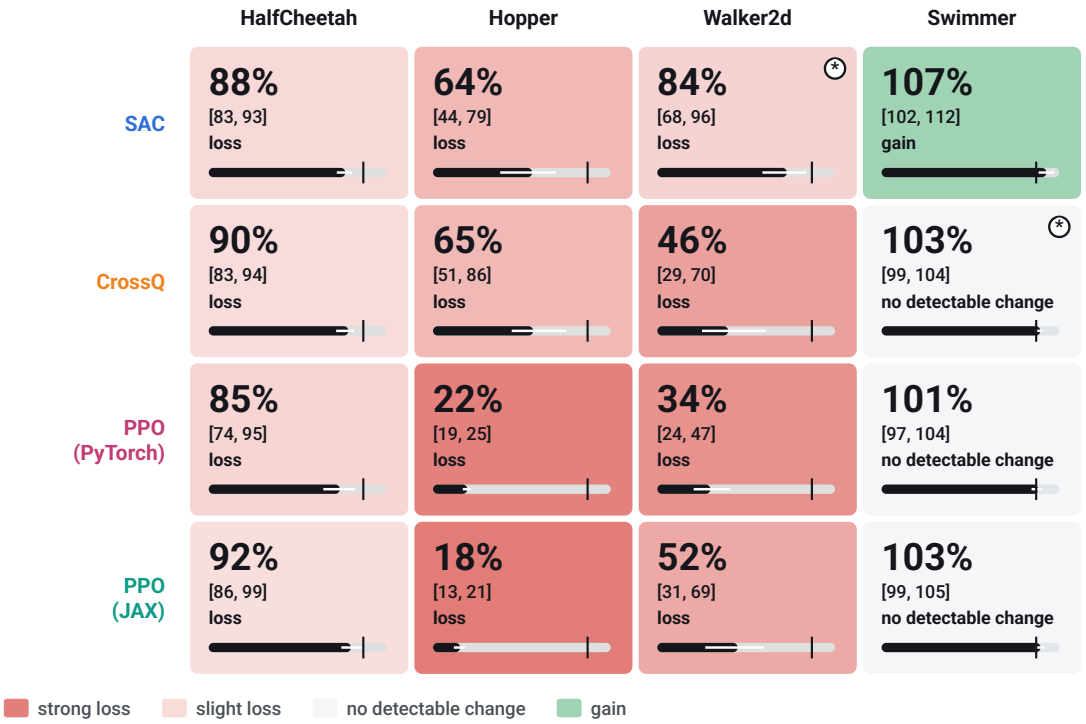
Comparison A against B	$P(A > B)$ [CI]	Welch p	Holm p	Mann-Whitney p
Zoo corrected (gamma 0.999), v5 / Zoo, first pass (gamma 0.99), v5	0.47 [0.21, 0.73]	0.84	0.84	0.85
Zoo corrected, v4 / Default values v4	**0.07 [0.00, 0.24]**	**0.0005**	**0.001**	**0.001**
Zoo corrected (gamma 0.999), v5 / Default values v5	**0.01 [0.00, 0.06]**	**2×10^{-5}**	**6×10^{-5}**	**0.0002**
Official Zoo script, v4 / Zoo corrected, v4	0.58 [0.32, 0.82]	0.88	0.88	
Official Zoo script, v4 / Default values v4	**0.09 [0.00, 0.30]**	**0.0005**	**0.001**	

- On Walker2d, PPO PyTorch keeps 34%, CrossQ 46%, PPO JAX 52%, SAC 84%.
- On HalfCheetah, the losses are smaller: from 85% kept for PPO PyTorch to 92% for PPO JAX.
- On Swimmer, no program loses.

The ranking upended on Hopper. The descriptive complement of deviation 11, with no test, ranks the programs in each physics by the IQM of their score. On Hopper, the order reverses. PPO JAX, first in the training physics with 3,064, ends last in the modified physics with 525. CrossQ, third with 2,302, takes the lead with 1,302, ahead of SAC (751) and PPO PyTorch (609). On Walker2d, SAC moves from last place to second. On HalfCheetah and Swimmer, the order does not change. Figure 6.11 shows the reversal on Hopper.

Under modified physics, twelve program-task pairs out of sixteen lose score

Share of the score kept when masses go to 120% and friction to 80%, interquartile mean over ten seeds with its 95% interval. A loss is declared only when the interval stays below 100%.



Gauge: share kept from 0 to 115%, its interval as a white line, a tick at 100%. Asterisk: the verdict changes with twenty seeds. SAC on Walker2d: no detectable change (91%); CrossQ on Swimmer: gain (103%).

Data: sim/analysis/e7.json and e7_s20.json, stratified bootstrap intervals (rliable, 2,000 draws) Deraison.ai

Figure 6.10. Share of score kept in a modified physics (masses multiplied by 1.2, sliding friction by 0.8), for each program and each task. Each cell gives the interquartile mean, over ten random seeds, of the ratio of the score in the modified physics to the score in the training physics, both measured on the same evaluation random seeds. Tasks are in columns, programs in rows. A cell marked as a loss has a 95% interval that stays below 1, a cell marked as a gain an interval above 1.

Twenty random seeds. The verdicts stay the same for 14 pairs out of 16. Two change. SAC on Walker2d no longer loses by a detectable amount: 0.91, from 0.79 to 1.01. CrossQ on Swimmer gains a little: 1.03, from 1.01 to 1.04. That makes 11 losses and 2 gains. The descriptive ranking still reverses on Hopper, the only task where the leading program changes.

What E7 establishes and its limits. A program that leads in the training physics ends last in a neighbouring physics: PPO JAX on Hopper. A ranking established in the training world says little about robustness. E7 tests only one modification, which changes two quantities at once. The next section separates the two quantities and varies the size of the change. The rankings in each physics compare IQMs without a test. The share kept is computed run by run, so it does not read as the ratio of the two IQMs of the ranking.

Table 6.9. Experiment E7, modified physics: each final policy evaluated on the same held-out evaluation seeds, in the training physics and then with masses multiplied by 1.2 and friction by 0.8. Ratio: score in the modified physics divided by the score in the training physics, per run; IQM of the ratios over the seeds with its 95% stratified bootstrap interval (rliable, 2,000 resamples). Preregistered verdict: loss when the interval stays below 1, gain when it stays above, n.s. (no detectable change) otherwise; **in bold**, losses and gains. Rank: the program's place on the task by IQM of the score, in the training physics and then in the modified physics (descriptive complement of deviation 11, decided after the results, no test). The leader changes on Hopper at ten seeds, on Hopper at twenty seeds. Sources: `sim/analysis/e7.json`, `e7_s20.json`, `e7_rank.json`, `e7_rank_s20.json`.

		10 seeds (main result)			20 seeds (deviation 12)		
Task	Program	ratio [95% CI]	verdict	rank	ratio [95% CI]	verdict	rank
HalfCheetah	PPO PyTorch	**0.85 [0.74, 0.95]**	**loss**	3→3	**0.89 [0.80, 0.97]**	**loss**	3→3
	PPO JAX	**0.92 [0.86, 0.99]**	**loss**	4→4	**0.89 [0.85, 0.94]**	**loss**	4→4
	SAC	**0.88 [0.83, 0.93]**	**loss**	1→1	**0.91 [0.88, 0.93]**	**loss**	2→2
	CrossQ	**0.90 [0.83, 0.94]**	**loss**	2→2	**0.91 [0.86, 0.93]**	**loss**	1→1
Hopper	PPO PyTorch	**0.22 [0.19, 0.25]**	**loss**	2→3	**0.23 [0.20, 0.27]**	**loss**	3→3
	PPO JAX	**0.18 [0.13, 0.21]**	**loss**	1→4	**0.19 [0.16, 0.23]**	**loss**	1→4
	SAC	**0.64 [0.44, 0.79]**	**loss**	4→2	**0.64 [0.49, 0.77]**	**loss**	4→2
	CrossQ	**0.65 [0.51, 0.86]**	**loss**	3→1	**0.55 [0.41, 0.73]**	**loss**	2→1
Walker2d	PPO PyTorch	**0.34 [0.24, 0.47]**	**loss**	2→3	**0.36 [0.25, 0.46]**	**loss**	2→3
	PPO JAX	**0.52 [0.31, 0.69]**	**loss**	3→4	**0.48 [0.34, 0.64]**	**loss**	3→4
	SAC	**0.84 [0.68, 0.96]**	**loss**	4→2	0.91 [0.79, 1.01]	n.s.	4→2
	CrossQ	**0.46 [0.29, 0.70]**	**loss**	1→1	**0.47 [0.32, 0.71]**	**loss**	1→1
Swimmer	PPO PyTorch	1.01 [0.97, 1.04]	n.s.	3→3	1.02 [0.99, 1.03]	n.s.	3→3
	PPO JAX	1.03 [0.99, 1.05]	n.s.	1→1	1.02 [0.98, 1.04]	n.s.	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]	n.s.	2→2	**1.03 [1.01, 1.04]**	**gain**	2→2
Losses, gains (out of 16)		12, 1			11, 2		

6.9 The physics modified by degrees

Question and plan. Does the reversal of E7 hinge on one modification chosen in advance? Deviation 12, item ii, re-evaluates each final policy while changing one quantity at a time: masses from 30% lighter to 30% heavier, friction from 40% weaker to 20% stronger. A pair loses when the interval of the IQM of its ratio to the original score stays below 1, as in E7 (section 5.10.8).

The control. At 1.0 times, the re-evaluations recover the final score written by `run.py`: the largest relative gap is zero, for the 160 policies at ten random seeds as for the 320 at twenty.

The loss grows with the size of the change. Table 6.10, figure 6.12 and figure 6.13 count the pairs that lose, out of 16, for each physics. With ten random seeds:

- masses multiplied by 0.7, 0.8, 0.9: 8, 6 and 3 pairs lose;
- masses multiplied by 1.1, 1.2, 1.3: 9, 12 and 11 pairs lose;
- friction multiplied by 0.6, 0.7, 0.8, 0.9: 7, 6, 4 and 3 pairs lose;

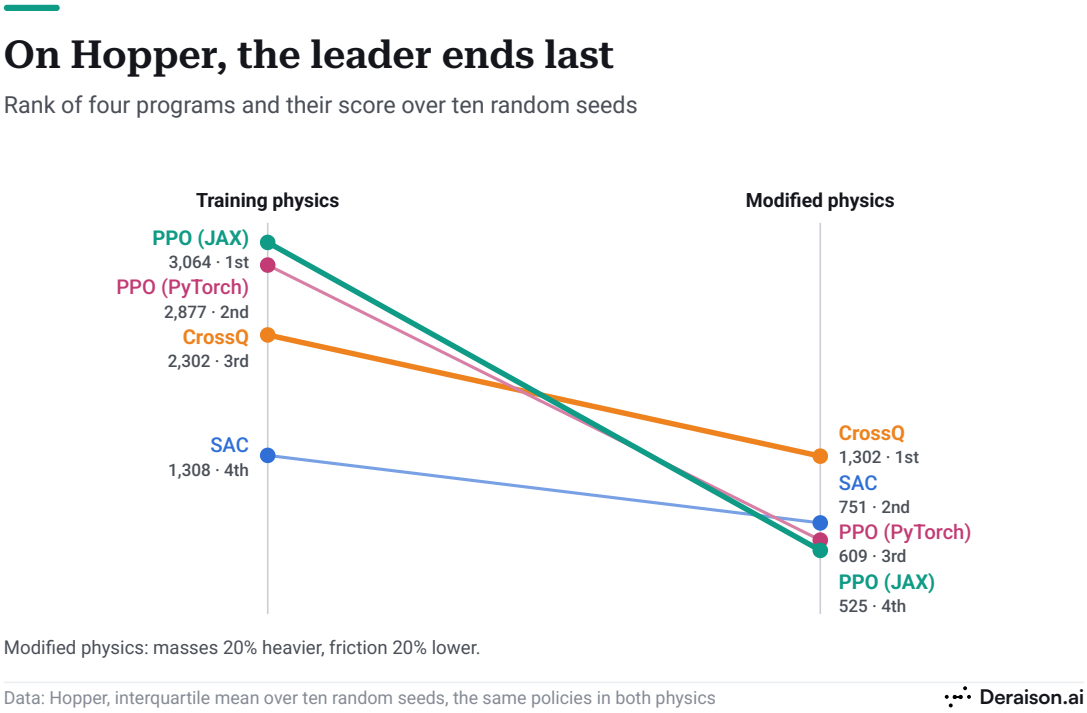


Figure 6.11. Hopper, four programs, ten random seeds each. On the left, the score in the training physics. On the right, the score with masses made 20% heavier and friction reduced by 20%. Each line links the score and rank of one program in the two physics. PPO written in JAX goes from first to last place, CrossQ from third to first. The scores are interquartile means of the final evaluation scores. The ranks compare them without a test.

- friction multiplied by 1.1 and 1.2: 1 and 9 pairs lose.

The losses trace a V around the original physics, steeper on the side of heavier masses. Making the masses 20% heavier alone makes 12 pairs lose, as many as the combined modification of E7. Reducing friction by 20% makes only 4 lose. In E7, the loss seems to come mostly from the masses, although neither experiment tests this decomposition. A few pairs gain, mostly when friction drops: 3 gains at 0.6 times and at 0.7 times.

The leading program on Hopper, physics by physics. On Hopper, PPO JAX, first in the original physics, is no longer first in 10 of the 12 modified physics. It keeps the lead only with friction at 0.9 and 1.1 times. The lead passes to PPO PyTorch or to CrossQ depending on the quantity and the factor. On the other three tasks, the leading program changes little: SAC in all twelve physics of HalfCheetah, CrossQ in all twelve of Walker2d, PPO JAX in ten of the twelve of Swimmer.

Twenty random seeds. The V keeps its shape. With masses at 0.9, 1.1, 1.2 and 1.3 times, 4, 11, 12 and 12 pairs lose. With friction at 1.2 times, 10. On Hopper, PPO JAX still loses the lead in 10 physics out of 12. On HalfCheetah, CrossQ takes the lead in 9 of the 12 physics, as it does in the original physics at twenty random seeds.

What the physics by degrees establishes. The reversal on Hopper does not hinge on one chosen modification. It appears in almost every neighbouring physics, at ten as at twenty random seeds. The rankings per physics still come without a test. The

Table 6.10. Experiment E7 at several strengths (deviation 12 (ii) of `sim/SIM_PROTOCOL.md`, rule written before any computation; deviation 14 (a) on the date of the script): each final policy re-evaluated on the same ten evaluation seeds, changing a single quantity at a time. For each strength, number of program and task pairs, out of 16, that lose (IQM of the ratio to the score in the original physics, 95% stratified rliable bootstrap interval with 2,000 resamples, entirely below 1) or that gain (interval above 1). Leader on Hopper: the program with the highest IQM of the score, no test; in the original physics, PPO JAX leads there. It loses first place at 10 strengths out of 12 at ten seeds, 10 out of 12 at twenty seeds. Control: at 1.0, the re-evaluations recover the final score of `run.py` (largest relative gap 0.0 at ten seeds, 0.0 at twenty). Sources: `sim/analysis/dose.json`, `sim/analysis/dose_s20.json`.

		10 seeds			20 seeds		
Quantity	Factor	losses	gains	leader on Hopper	losses	gains	leader on 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
friction	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

modifications are small and uniform: each mass or each friction coefficient changes by the same factor. They cannot replace a real test of generalisation, on other bodies or other tasks.

6.10 B3: does the measure change the ranking?

Question and plan. Can a paper that does not define its measure choose its winner? B3 ranks the four programs by three measures computed on the same curves: the final score, the best score reached along the curve and the mean of the curve (section 5.10.9).

Ten random seeds: eight reversals, none settled. Eight pairs of programs change order from one measure to another (table 6.11). On two tasks, the leading program changes with the measure.

- On HalfCheetah, the mean of the curve puts CrossQ ahead of SAC, which leads by the final score and by the best score.
- On Hopper, the best score puts PPO PyTorch ahead of PPO JAX, which leads by the other two measures.

The other reversals involve CrossQ and PPO PyTorch on Hopper and on Swimmer. None is settled: they set close programs against each other, which ten random seeds do not separate under any measure. The mean of the curve covers one million steps for PPO and 200,000 for SAC and CrossQ. It favours a program that learns fast early on.

Table 6.11. Experiment B3, does the measure change the ranking? (deviation 8 of `sim/SIM_PROTOCOL.md`, no training). On the recorded evaluation curves (one point every 50,000 steps), three measures per run: the final score, the best score reached along the curve, the mean of the curve. Per task, IQM of each measure and ranking of the programs; leader: highest IQM. A pair reverses when its IQM order changes from one measure to another; the reversal is settled when, under each of the two measures, the interval of the probability of improvement (stratified rliable bootstrap, 2,000 resamples) excludes 0.5 in opposite directions. **In bold**, leaders that change with the measure and settled reversals. The mean of the curve covers a million steps for PPO and 200,000 for SAC and CrossQ, except at equal budget (deviation 12 (iii)). Sources: `sim/analysis/b3.json`, `b3_s20.json`, `b3_equal.json`, `b3_equal_s20.json`.

Task	leader, final	leader, best	leader, mean	reversed pairs (measures)
10 seeds, own budgets (main result): 8 reversals, 0 settled				
HalfCheetah	**SAC**	**SAC**	**CrossQ**	CrossQ, SAC (final/mean, best/mean)
Hopper	**PPO JAX**	**PPO PyTorch**	**PPO JAX**	PPO PyTorch, PPO JAX (final/best, best/mean); CrossQ, PPO PyTorch (final/mean, best/mean)
Walker2d	CrossQ	CrossQ	CrossQ	
Swimmer	PPO JAX	PPO JAX	PPO JAX	CrossQ, PPO PyTorch (final/mean, best/mean)
20 seeds, own budgets: 8 reversals, 0 settled				
HalfCheetah	CrossQ	CrossQ	CrossQ	
Hopper	**PPO JAX**	**PPO PyTorch**	**CrossQ**	CrossQ, PPO PyTorch (final/best, best/mean); PPO PyTorch, PPO JAX (final/best, final/mean); CrossQ, PPO JAX (final/mean, best/mean)
Walker2d	CrossQ	CrossQ	CrossQ	
Swimmer	PPO JAX	PPO JAX	PPO JAX	CrossQ, PPO PyTorch (final/best, final/mean)
10 seeds, equal budget (200,000 steps): 4 reversals, 0 settled				
HalfCheetah	**SAC**	**SAC**	**CrossQ**	CrossQ, SAC (final/mean, best/mean)
Hopper	CrossQ	CrossQ	CrossQ	
Walker2d	CrossQ	CrossQ	CrossQ	
Swimmer	CrossQ	CrossQ	CrossQ	PPO PyTorch, SAC (final/best, best/mean)
20 seeds, equal budget: 6 reversals, 1 settled				
HalfCheetah	CrossQ	CrossQ	CrossQ	
Hopper	CrossQ	CrossQ	CrossQ	PPO PyTorch, PPO JAX (final/mean, best/mean)
Walker2d	CrossQ	CrossQ	CrossQ	PPO PyTorch, PPO JAX (final/mean, best/mean)
Swimmer	CrossQ	CrossQ	CrossQ	PPO PyTorch, SAC (final/best, **best/mean**)

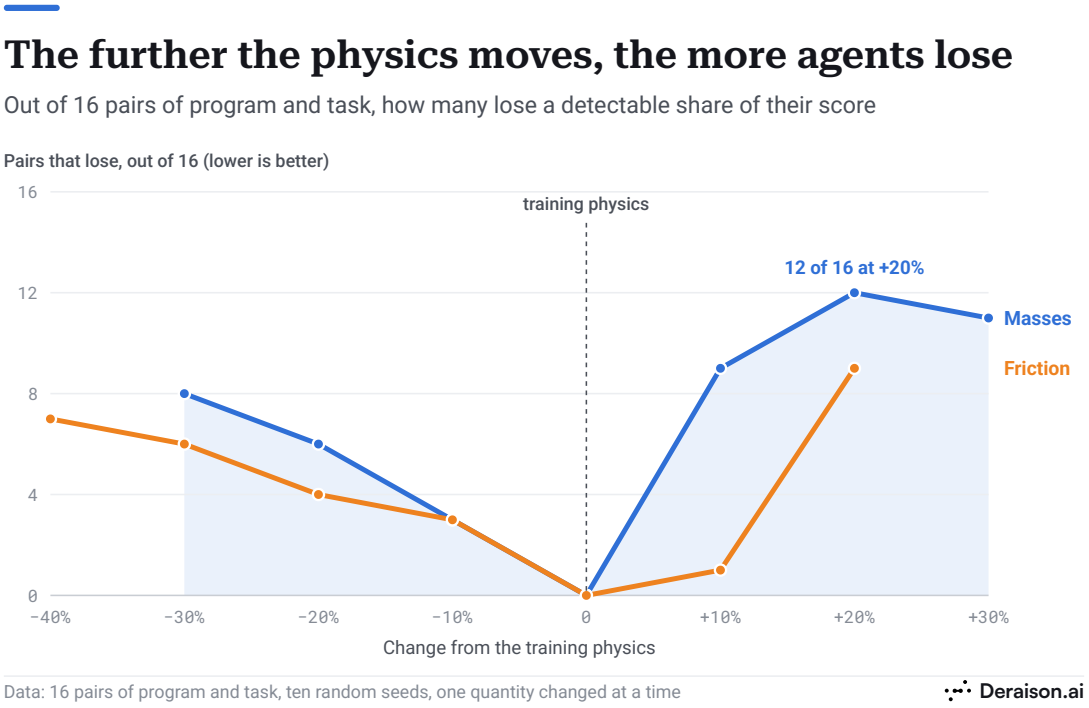


Figure 6.12. Four programs on four tasks, ten random seeds each, or 16 program-task pairs. Each point counts the pairs that lose a detectable share of their score when one quantity of the physics changes: the masses (from 30% lighter to 30% heavier) or the sliding friction (from 40% weaker to 20% stronger). A pair loses when the 95% interval of its score, divided by its score in the training physics, stays below 1. At 0%, the training physics serves as the reference: no loss can occur there.

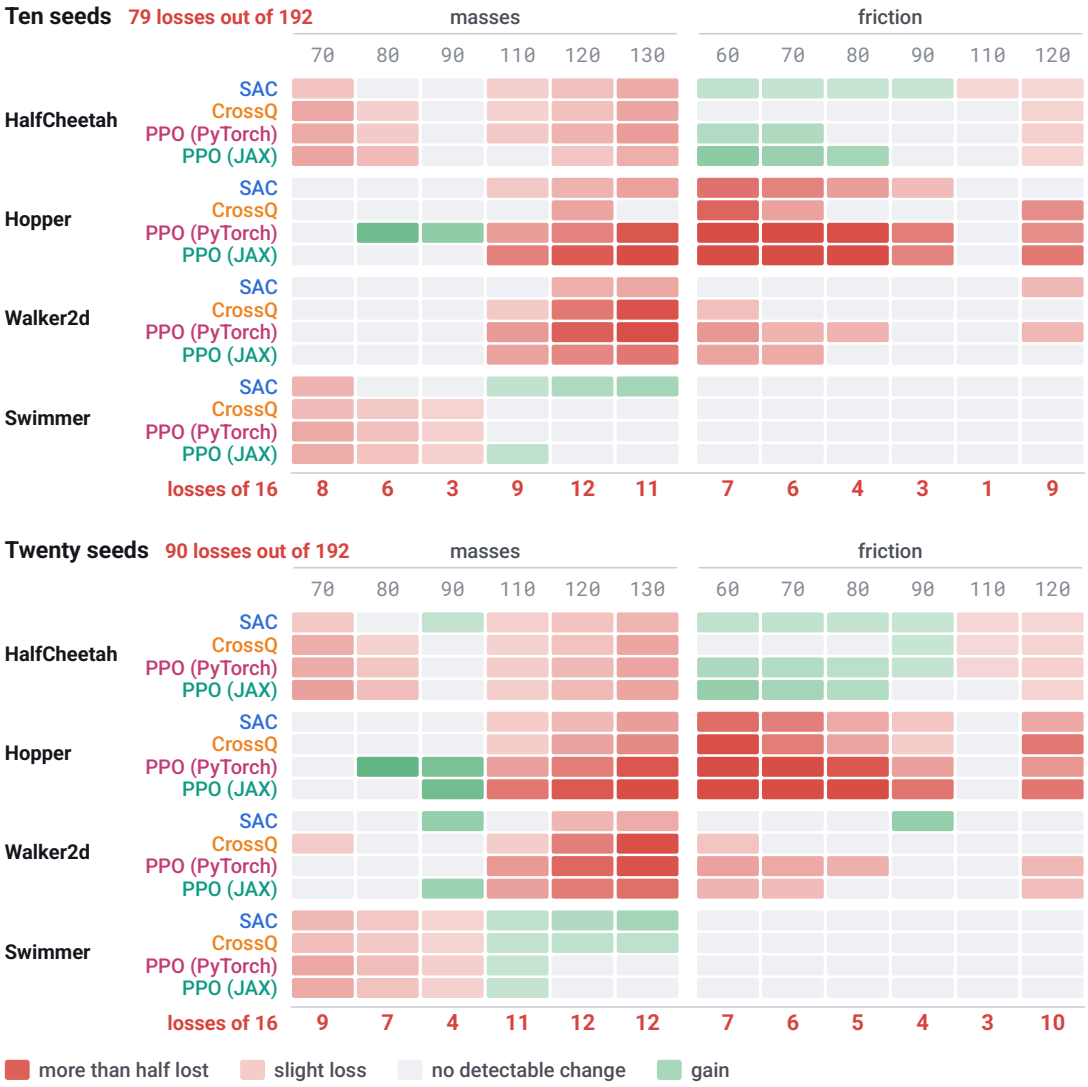
Twenty random seeds and equal budget.

- At unequal budgets with twenty random seeds, 8 pairs still change order, with no settled reversal. The leading program changes with the measure only on Hopper.
- At equal budget with ten random seeds, 4 reversals, none settled. The leading program changes with the measure only on HalfCheetah, between SAC and CrossQ.
- At equal budget with twenty random seeds, 6 reversals, of which 1 is settled: on Swimmer, PPO PyTorch and SAC change order between the best score reached and the mean of the curve. The leading program changes with the measure on no task.

What B3 establishes. Between two close programs, the chosen measure sometimes decides the apparent winner. Between clearly separated programs, the measure changes nothing. Only one reversal is settled across the four analyses. It involves two programs far from the leader. A paper that does not state its measure leaves the winner of a close comparison an open question.

Away from the training physics, losses pile up: 12 of 16 at 120% of the masses

Share of the score kept, one quantity changed at a time: masses from 70 to 130%, friction from 60 to 120%. One cell per program, task and strength, coloured when the 95% interval avoids 100%.



Above the columns, the strength in percent of the training value. Twenty seeds narrow the intervals: more losses become detectable.

Data: sim/analysis/dose.json and dose_s20.json, stratified bootstrap intervals (rliable, 2,000 draws) Deraison.ai

Figure 6.13. Share of score kept in twelve modified physics, with ten random seeds, then twenty, shown next to each other. One row per program and task (16), one column per physics: masses from 70% to 130% of their training value, then sliding friction from 60% to 120%, one quantity changed at a time. A cell is coloured only if the 95% interval of the interquartile mean of the ratio to the original score avoids 1: red for a loss, darker when more than half the score is lost, green for a gain. Under each column, the number of pairs that lose, out of 16.

6.11 Ten or twenty random seeds, equal budget: what holds and what changes

The results with ten random seeds remain the main results, as the protocol announced. The additions of deviation 12 confirm or qualify them.

What holds.

- The leading program changes with the task at the protocol's budgets (E2), in a settled way, with or without Bonferroni's correction.
- Selecting the best three runs out of ten always inflates the mean and reverses 3 comparisons out of 24, on random seeds 10 to 19 as on random seeds 0 to 9 (E4).
- The modified physics upends the ranking on Hopper. The verdicts of E7 stay the same for 14 pairs out of 16. The physics by degrees keeps its V shape.
- The reversals of B3 at unequal budgets still number 8, with no settled reversal.

What changes.

- On HalfCheetah, CrossQ takes the lead from SAC, with no settled gap between them (E2).
- The two versions of PPO part ways on Walker2d and Swimmer, in opposite directions (E3).
- The close comparisons are harder to order: with five random seeds, the correct order is recovered in only 62% of the draws at the median, instead of 71% (E5).
- At equal budget with twenty random seeds, CrossQ leads on all four tasks: the ranking no longer changes with the task (E2).

The last point is the qualification that weighs most. The verdict of E2, as announced, holds. With twenty random seeds, however, the change of leading program from one task to the next is largely explained by PPO's budget, five times larger.

6.12 What these experiments leave open

The limits specific to each experiment are stated in its section. A few limits apply to all of them.

- All runs were made on one machine, a Mac with an Apple M2 Max chip, on the CPU. A rerun elsewhere would give other curves for the same random seeds.
- The programs keep the default values of their libraries. The rankings apply to these settings. They say nothing of the algorithms at their best.
- Four MuJoCo tasks, all with planar or nearly planar bodies, do not represent the reinforcement learning of 2026. The language-model part (chapter 7) covers another share of it, at small scale.
- Ten evaluation episodes, played without randomness from ten fixed starting positions, measure a policy on a small part of the situations it could meet.

- Ten random seeds, then twenty, leave wide intervals for close comparisons. E5 puts a number on this.
- Several analyses were added after a result had been seen: the second statistic of E1, the diagnosis of E6, the descriptive ranking of E7. Section 5.12 lists them. The text presents them as such.

CHAPTER 7

The language-model strand

The previous chapters deal with programs that learn to act in a simulated world. This chapter describes a smaller strand, devoted to the generative AI models that write text. It sets out the vocabulary we use, the algorithm as we wrote it, the protocol, the runs and their results, then the measurement trap these runs exposed. The full protocol is in `sim/llm/LLM_PROTOCOL.md`. Its seventeen deviations are summarised in chapter 9.

7.1 Vocabulary

The vocabulary of RL has spread to generative AI models. Some RL algorithms are borrowed to refine the answers of a language model, with no agent acting in a world. A language model is an AI model that produces a text token by token, a token being a piece of a word. For a mathematics problem, a program compares the final answer with the correct answer. The reward is 1 when the two agree. The literature calls this RL with verifiable rewards (RLVR, *reinforcement learning with verifiable rewards*). We keep the acronym when we cite a paper that uses it, without adopting its suggestion that the problem is the same as that of an agent in a world.

Three differences separate this setting from the classical RL of chapters 5 and 6:

- an ‘episode’ shrinks to one answer: no state evolves under the actions, no reward is delayed over several decisions;
- the starting policy is a model already trained on huge corpora, which often answers correctly before any fine-tuning;
- the reward depends on the final text alone.

Fine-tuning from human feedback (RLHF, *reinforcement learning from human feedback*, Ouyang et al. 2022) is never called RL here: we refer to it by its acronym, as one fine-tuning technique among others.

7.2 Why this strand

In 2025, Shao et al. (2025) showed that a reward drawn at random, without checking whether the answer is correct, improves Qwen models in mathematics, while Llama and OLMo models do not improve. On the Qwen2.5-Math-7B model and the MATH-500 benchmark, the true reward brings a gain of 29.1 percentage points. A reward that praises a wrong answer fixed in advance brings 24.1, a reward drawn at random 21.4. The authors trace this to a behaviour that training amplifies without creating it: the model writes its reasoning as code, without running it, in 65% of answers before training and in more than 90% after training with a misleading reward. They recommend validating a method on several model families.

This result extends two flaws of 2018: the best algorithm changes with the task (F13) and benchmarks that are too simple get learned (F16). The survey of chapter 4 measures the same risk in papers: sixteen of the 27 papers of the first 2026 draw that apply what they call RL to large language models try one model family only (criterion X6). Our strand looks at the phenomenon closely, at small scale, with two model families. The protocol announces its scope in advance: an illustration on a laptop. It does not replicate Shao et al. (2025).

7.3 The GRPO algorithm, as we wrote it

GRPO (*Group Relative Policy Optimization*) comes from Shao et al. (2024). It keeps the clipped objective of PPO (Schulman et al. 2017) and replaces the critic network, which estimates the value of a state, with a normalisation within a group of answers to the same problem. We wrote it by hand, in PyTorch, in one file of 159 lines, `sim/llm/grpo.py`, to control every step. The file imports nothing but PyTorch and the Transformers library, which loads the model and its tokeniser: no fine-tuning library is involved.

7.3.1 The task and the reward

The task we chose, `mul2`, asks for the product of two integers from 12 to 99. The file enumerates the $88 \times 88 = 7{,}744$ ordered pairs, shuffles them with a generator whose random seed is fixed (`random.Random(12345)`) and independent of the random seed of the run, then sets the first 300 aside for testing. The other 7,444 serve for training. The prompt follows a fixed template, 'What is a * b? Think step by step, then finish with the final answer as an integer.', passed through the model's chat template.

An answer is correct when the last integer it contains, thousands separators removed, equals the product. The verdict requires no format, so that format is never rewarded. Let $c(o, q) \in \{0, 1\}$ be this verdict for answer o to problem q. The two conditions of the protocol define the reward r of each answer:

$$\text{true reward (`true`)}: \quad r = c(o, q), \tag{7.1}$$

$$\text{reward drawn at random (`random`)}: \quad r \sim \text{Bernoulli}(1/2), \text{ drawn without looking at } o. \tag{7.2}$$

The coin toss and the draw of each step's problems come from the same generator, `random.Random(seed)`, which depends on the random seed of the run and nothing else.

7.3.2 One training step

Each step draws $B = 4$ training problems without replacement. For each problem q_b, the current policy $\pi_{\theta_{\text{old}}}$ produces $G = 8$ answers $o_{b,1}, \ldots, o_{b,8}$, sampled at temperature 1 from the untruncated distribution (neither *top-p* nor *top-k*) and limited to 160 tokens. Each answer receives its reward $r_{b,i}$. The advantage of an answer compares its reward with those of its group:

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

The standard deviation s_b follows the PyTorch convention, with the factor $G-1$. A group in which all eight answers receive the same reward gives each of them a zero advantage and contributes no gradient.

For token t of answer $o_{b,i}$, the probability ratio compares the policy being updated with the policy that produced the answer:

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

The loss we minimise is the negative of the clipped objective, averaged over tokens within each answer, then over answers within the step:

$$\mathcal{L}(\theta) = -\frac{1}{BG}\sum_{b=1}^{B}\sum_{i=1}^{G}\frac{1}{|o_{b,i}|}\sum_{t=1}^{|o_{b,i}|}\min\Big(\rho_{b,i,t}(\theta)\,\hat{A}_{b,i},\ \text{clip}\big(\rho_{b,i,t}(\theta), 1-\varepsilon, 1+\varepsilon\big)\,\hat{A}_{b,i}\Big), \qquad \varepsilon = 0.2. \tag{7.5}$$

The length $|o_{b,i}|$ counts the tokens of the answer that are not padding (the mask `target != pad_token_id` in the function `token_logps`). The reference log-probabilities $\log \pi_{\theta_{\text{old}}}$ are computed once, without gradient, before the first update.

Each batch of 32 answers serves for two update passes (`EPOCHS = 2`). In each pass, the gradient accumulates over sub-batches of eight answers (two for Llama, setting `MICRO`), which gives the same gradient with less memory. The gradient norm is then clipped at 1. The Adam optimiser (Kingma and Ba 2015), with its default moments, applies a learning rate of 3×10^{-6}. The weights stay in 32-bit floating point.

How our GRPO differs from that of Shao et al. (2024). Our loss has no Kullback-Leibler divergence term with respect to the starting model, as the protocol of 8 November 2026 announced. The group holds 8 answers per problem and the step 4 problems, far fewer than in the reference papers. The 160-token limit is very short for written reasoning. The learning rate was not chosen by the planned pilot, which was stopped at step 10 (deviations 4 and 6). These choices are adequate for an illustration. They rule out reading our figures as a measurement of GRPO at the scale of the papers.

7.3.3 Safeguards against a corrupted computation

The first runs on the Mac's graphics processor (GPU, through PyTorch's MPS interface) crashed on non-finite probabilities. Deviation 10 of the protocol documents the cause: macOS interrupts GPU work that hinders interactive use of the machine. Such a computation sometimes comes back without an error, with corrupted values. The same step, redone on the central processor (CPU), gives a gradient norm of 14.4 before clipping, with no non-finite value. Since that deviation, `grpo.py` applies four rules:

1. an update whose gradient norm before clipping is non-finite or exceeds 1,000 is recomputed, up to five times in a row, after which the run stops (`GRAD_SANE`, `RETRIES`);
2. a generation that raises a GPU error is redone, up to five times;
3. the run stops if the reference log-probabilities or the weights become non-finite;
4. the log counts the recomputations at each step (field `retries`).

The threshold of 1,000 comes from the norm of 14.4 measured on CPU. A gradient that is corrupted but finite, below this threshold, then clipped at 1, would pass without a

trace. The protocol records this as a limit, in particular for the first three runs, which ran before these rules existed.

7.3.4 The measurement

Evaluation uses the test set and combines two modes. Greedy accuracy takes one answer with no randomness, the most probable token at each step, for each of the first 150 test problems. Sampled accuracy draws 8 answers at temperature 1 for each of the first 50 test problems. With c_{jk} the verdict on answer k to problem j and τ_{jk} the indicator of a cut-off answer:

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

An answer is cut off when its 160 tokens contain neither an end token nor a padding token (function `truncated`). Here pass@8 is the observed frequency over the eight draws, without the unbiased estimator used when more than k answers are drawn. Evaluation takes place at step 0, step 50 and step 100 (`EVAL_EVERY = 50`). Every ten steps, the log also records the share of correct and of cut-off answers in the training batch, the loss, the gradient norm, the recomputations and the elapsed time. Each run writes a JSONL file, `sim/llm/results/<model>/mul2/<reward>/seed<seed>.jsonl`.

The main measure is sampled pass@1, the quantity GRPO trains (deviation 3). Greedy accuracy and pass@8 stay in the logs. Pass@8 answers the question of Yue et al. (2025): does fine-tuning widen what the model can do, or does it concentrate probability on answers the model already found within several draws?

7.4 The protocol and its runs

Models. Two families, two small open models: Qwen2.5 with 0.5 billion parameters, *Instruct* version (Qwen Team 2025), then Llama 3.2 with 1 billion, *Instruct* version, in the copy published by `unsloth` (Grattafiori et al. 2024; Meta AI 2024). Since deviation 15 (c), `grpo.py` pins the exact revision of the downloaded files (`7ae5576…` for Qwen, `5a8abab…` for Llama), the only revisions present in the machine's cache, downloaded on 22 and 27 October 2026, before the first run.

Pilot. A pilot without training, on problems outside the test set, was to fix the task: the one on which both models lie between 15% and 70% accuracy. In greedy decoding, no task places both models in this range. The sum of three numbers (`add3`) is too easy for Llama. The multiplication `mul2` gives 7% (Qwen) and 73% (Llama) in greedy decoding, 11% and 23% in sampled pass@1 (deviation 3). The same pilot showed that 64 tokens cut off the answers of Qwen, which writes 100 to 160, which gave an apparent accuracy of 0.7%. The limit went up to 160 tokens, after reading real outputs, before any training (deviation 1).

Plan. Two models, two rewards, 100 steps per run, a learning rate of 3×10^{-6} for all conditions. The protocol planned 200 steps and five random seeds. Deviation 6 reduced the plan to 100 steps and three random seeds (0 to 2), before any retained run, to fit the available compute. Deviation 14, written on 15 November 2026 before any new run,

restored random seeds 3 and 4, with the same code and the same settings. The starting point of each run is the base model, measured with the random seed of the run.

Statistics. For each random seed s, the gain is $g_s = \text{pass@1}_{100} - \text{pass@1}_0$. Each configuration (model, reward) is summarised by the mean gain and a 95% percentile bootstrap interval: 10,000 resamples of the gains with replacement, generator `random.Random(0)`, bounds at ranks 250 and 9,750 of the sorted means (`sim/llm/summary.py`). The details of the bootstrap were fixed after the first results, with no choice among several variants (deviation 13). The interval is very rough: three random seeds allow only ten distinct resamples, five random seeds 126.

A run that fails twice. The Llama run with the random reward and random seed 4 stopped two nights in a row on the first safeguard, at its first update. Deviation 16 counts this second failure as a result: the configuration is summarised over its four finished runs, with the failure stated next to them. A sensitivity analysis also replaces the missing gain with the smallest, then with the largest gain of the nineteen finished runs.

Hardware. All runs ran on the project's Mac (chapters D and 8). The first three Qwen runs ran on the GPU, early in the morning, on an idle machine. After the crashes of deviation 8 and the diagnosis of deviation 10, the remaining Qwen runs moved to the CPU, with two threads (`cpu.sh`, deviation 11). The Llama runs, about two and a half times slower than Qwen on CPU, stayed on the GPU, at night only (`night.sh`), with the safeguards. Table 7.1 gives the device of each run.

7.5 Results

Table 7.1 gives, for each of the twenty runs, pass@1 at the start and after 100 steps, the device and the share of cut-off answers at the end. Figure 7.1 shows each run as an arrow, figure 7.2 the three evaluation points of each run.

7.5.1 The reward drawn at random

With the coin toss, success falls to 1% or less in all nine finished runs. The mean gain is -10 points on Qwen (from -11 to -9) and -24 points on Llama (from -24 to -23). Replacing the missing Llama gain with the smallest gain of the nineteen finished runs changes little (-24, from -24 to -23). Replacing it with the largest, $+51$, lifts the mean to -9, with an interval from -24 to $+21$. In our hands, at this scale, the reward drawn at random destroys the starting competence of both families. It does not reproduce the gain that Shao et al. (2025) observe on Qwen2.5-Math-7B. Our conditions differ from theirs in almost every respect (size and specialisation of the model, task, answer length, number of steps, absence of a KL term), so this disagreement does not refute their result.

7.5.2 The true reward, in two stages

With three random seeds, Qwen starts at about 10% and ends at 36%, 61% and 13%. The mean gain is $+26$ points, with an interval from $+2$ to $+51$. On Llama, success starts between 23% and 25% and ends at 20%, 48% and 25%, a mean gain of $+7$ points, from

Table 7.1. GRPO strand (`sim/llm/LLM_PROTOCOL.md`): two small open models, Qwen2.5 with 0.5 billion parameters and Llama 3.2 with 1 billion, trained for 100 steps on the `mul2` task (product of two numbers from 12 to 99), with the true reward and then with a reward decided by a coin toss, five seeds (0 to 2 then 3 and 4, deviation 14). First-attempt success: sampled pass@1 on the test problems, in percent; gain: step 100 minus step 0, in points; cut-off answers: share of answers that hit the 160-token limit before giving a result, all scored as wrong. Device: Apple GPU (MPS) or CPU; three Qwen runs moved to CPU after non-finite gradients on MPS (deviations 10 and 11), so device and seed are confounded for Qwen. Bottom part: the preregistered statistic, mean of the per-seed gains with a 95% percentile bootstrap interval (10,000 draws, `random.Random(0)`), very rough with five seeds; **in bold**, intervals that exclude zero. The Llama configuration with the random reward is summarised over its four finished runs (deviation 16), with the sensitivity to the missing gain. Sources: `sim/llm/results/summary_5.json`, `gains_5.json`, `extra_5.json`; cut-off share at step 0: first evaluation of each file `sim/llm/results/<model>/mul2/<reward>/seed<s>.jsonl`.

				First-attempt success (%)			Cut-off answers (%)	
Model	Reward	Seed	Device	step 0	step 100	gain (pts)	step 0	step 100
Qwen2.5 0.5B	true	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
	random	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	true	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
	random	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)	failed twice (deviation 16)				

Mean gain in points [95% bootstrap CI]	
Qwen2.5 0.5B, true, 5 seeds	+14 [−3, +34]
Qwen2.5 0.5B, random, 5 seeds	**−10 [−11, −9]**
Llama 3.2 1B, true, 5 seeds	+6 [−0.4, +15]
Llama 3.2 1B, random, 4 seeds	**−24 [−24, −23]**
Qwen2.5 0.5B, true, CPU only, seeds 2, 3, 4	−2 [−11, +4]
Qwen2.5 0.5B, random, CPU only, seeds 1, 2, 3, 4	**−11 [−11, −10]**
Llama 3.2 1B, random, seed 4 = smallest gain, −25	−24 [−24, −23]
Llama 3.2 1B, random, seed 4 = largest gain, +51	−9 [−24, +21]

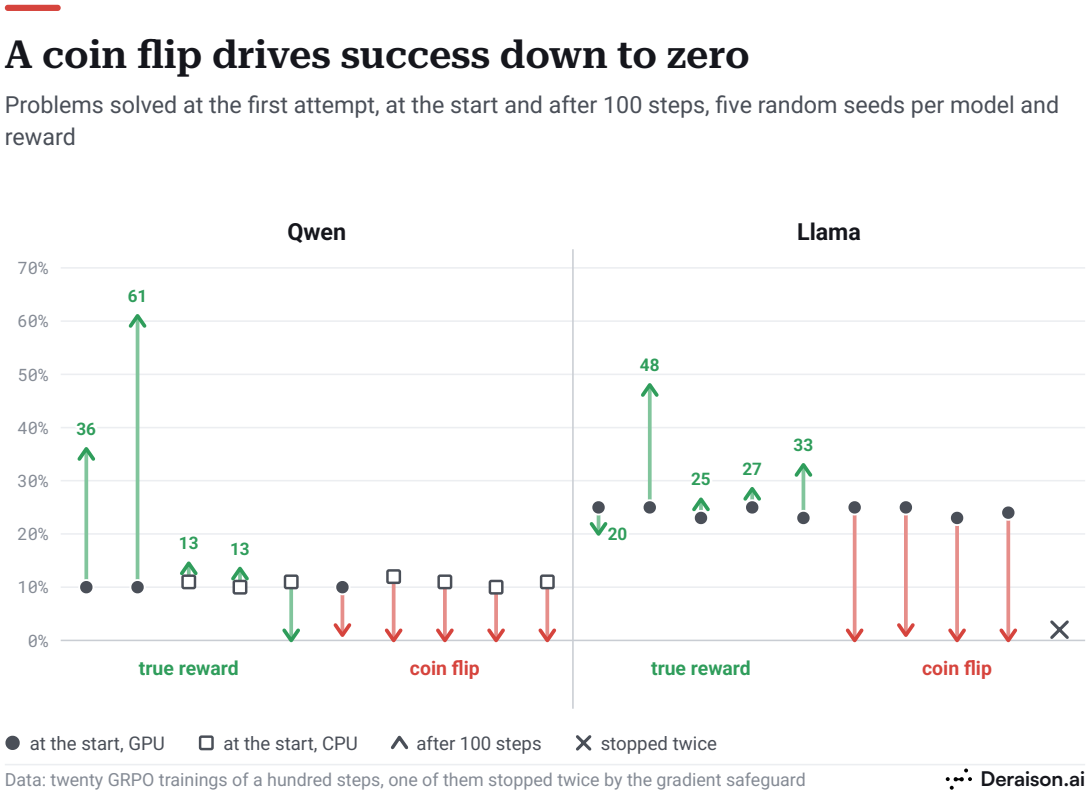


Figure 7.1. Pass@1 at the start and after a hundred training steps, for each model, each reward and each of the five random seeds. A filled circle marks a run on GPU, an empty square a run on CPU, a cross the run that stopped twice. With the true reward, the same program gives very different results from one random seed to another. With the coin toss, success falls to zero or nearly so.

-4 to $+22$. The same program gives three very different results: this is flaw F11, same code with another random seed, in miniature.

With two more random seeds, the new Qwen runs end at 13% and 0%. Over five random seeds, the mean gain falls to $+14$ points, from -3 to $+34$: the interval now contains zero. The gain we read on Qwen with three random seeds no longer holds. On Llama, the two new runs end at 27% and 33%. The mean gain is $+6$ points, from -0.4 to $+15$: zero stays inside the interval. With five random seeds, we cannot say that GRPO improves either model on this task, under these conditions.

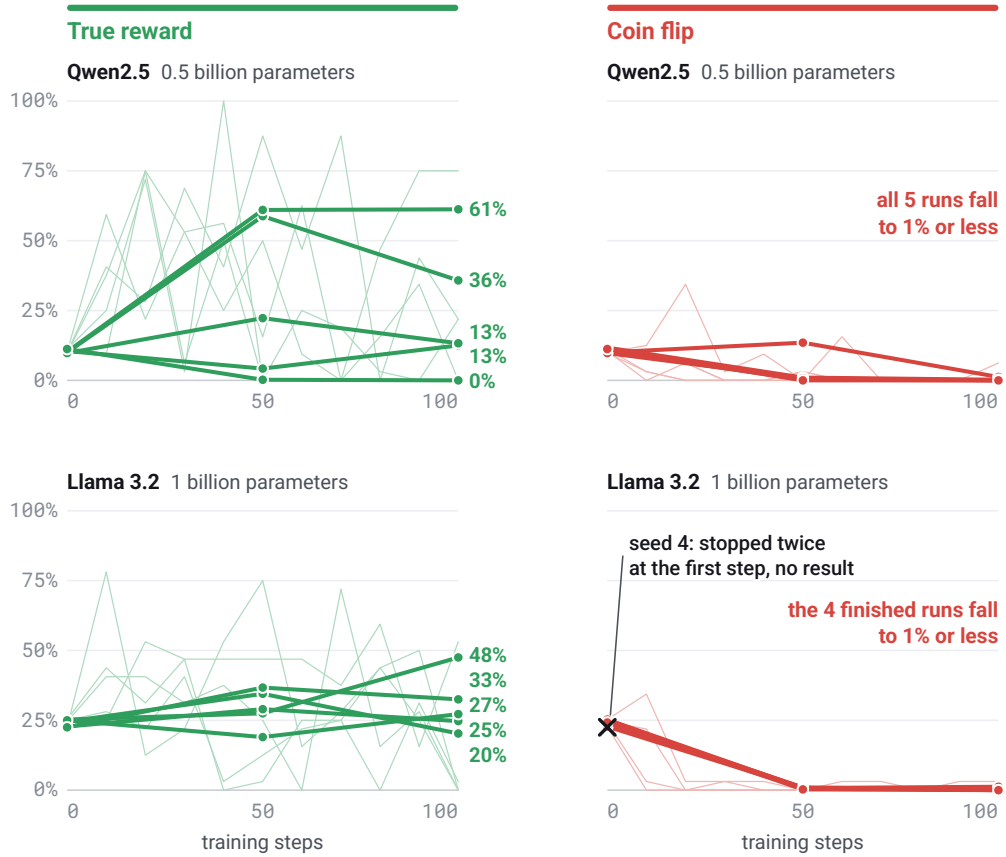
7.5.3 Hardware and random seed, mixed up

The Qwen gain with the true reward comes from the two runs made on GPU, random seeds 0 and 1: $+26$ and $+51$ points. The three true-reward runs that ran on CPU (random seeds 2 to 4) have a mean gain of -2 points, from -11 to $+4$. The four CPU runs with the random reward (random seeds 1 to 4) have a mean gain of -11 points, from -11 to -10. On CPU alone, the true reward brings Qwen no detectable gain. Data: `sim/llm/results/gains_5.json` and `extra_5.json`.

These figures do not separate two causes. The hardware changed with the random seed: each Qwen random seed with the true reward ran on one device only. Two explanations remain open. In the first, the chance of the random seeds explains everything: the two favourable random seeds happened to fall on the GPU. In the second, the device

True reward: each seed its own result. Coin flip: success dies out.

Share of problems solved at the first attempt during the hundred GRPO steps, one line per random seed (five per configuration). Thick line: evaluation on the held-out problems, at steps 0, 50 and 100. Thin line: success on the training batch, every ten steps.



Number on the right: success at step 100, rounded. Cross: run with no result. The training batch holds 32 answers: its thin line jumps from one step to the next.

Data: sim/llm/results/<model>/mul2/<reward>/seed<k>.jsonl, summary_5.json Deraison.ai

Figure 7.2. Test pass@1 at steps 0, 50 and 100 of each run, per model, reward and random seed. Evaluation takes place only every fifty steps, so each curve joins three points.

changes the course of training, through the rounding of the computations or through GPU computations that were corrupted but finite and that the first runs would not have detected. Five runs do not decide between these explanations.

The logs show that the device already alters the starting measurement. Pass@1 at step 0 depends on the base model and the random seed alone, since the reward comes into play only after this first evaluation: for a given model and random seed, the two conditions start from the same point if the computation is deterministic. They do start from the same point when both runs ran on the same device without incident, for example Qwen with random seeds 2, 3 and 4 on CPU. For Qwen with random seed 1, the true-reward run, on GPU, starts from 0.102,5. The random-reward run, on CPU, starts from 0.117,5. For Llama with random seed 3, both runs ran on GPU and start from 0.25 and 0.242,5: the MPS computation is not deterministic from one attempt to the next, as deviation 7 had already observed (table 7.1).

A run outside the protocol. Deviation 16 (e) planned to run the Llama run with the random reward and random seed 4 once on CPU, after all other computations, to learn whether its double failure was due to the GPU. The run completed on 21 November 2026, with no recomputation. Its pass@1 goes from 0.232,5 at step 0 to 0.002,5 at step 100 (`sim/llm/results/llama1_cpu/mul2/random/seed4.jsonl`). The result stays out of every statistic, as the deviation had set. It is consistent with a failure due to the GPU without proving it, since one run does not separate the device from chance. On CPU, this tenth random-reward run behaves like the other nine: success collapses.

7.6 The cutoff trap

7.6.1 The diagnosis

The incident of deviation 4 exposed this trap. During the learning-rate pilot, the share of correct answers in the training batch was 0.91 at step 10, while the test pass@1 of the base model was 0.11. The diagnosis of deviation 5 (`sim/llm/diag/replay_batch.py`) replays, without training, the exact batch of step 10 of the pilot: the four multiplications 48×49, 14×36, 21×50 and 45×17, eight answers each, with the starting Qwen model. The model gives 5 correct answers out of 32, in line with the test. Yet 23 of the 32 answers reach the 160-token limit before giving a result. None of these cut-off answers is correct. Among the 9 complete answers, 5 are correct (figures 7.3 and 7.4).

The 0.91 of step 10 was no counting error. The simplest explanation is that the true reward first teaches the model to finish before the cutoff, so that the training batch had become easier to finish. The reward drawn at random has no such lever, since it never looks at the answer. The protocol kept the 160 tokens and added the measurement of the cut-off share, in training and in test, before any retained run.

7.6.2 What the twenty runs show

At the start, the share of cut-off answers is high for Qwen and low for Llama: from 71% to 77% for Qwen, 3% at most for Llama, according to the evaluation logs at step 0. With the true reward, over the ten runs, it ends between 0% and 29%. With the coin toss, it ends above one half in six runs out of nine, between 2% and 16% in the other three. Two Llama runs with the random reward end with 99% and 100% of answers cut off. A

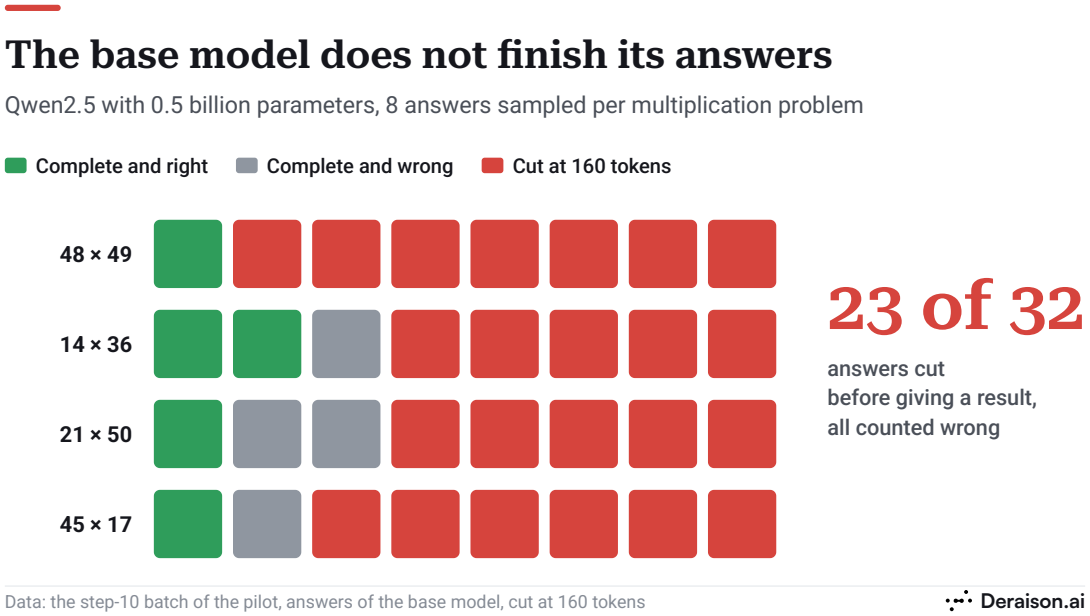


Figure 7.3. The answers of the Qwen2.5 model with 0.5 billion parameters before any training, eight per problem, on the four multiplications of the tenth step of the pilot. A complete correct answer is green, a complete wrong answer grey, a cut-off answer red. Most answers reach the 160-token limit before giving a result and are scored as wrong.



Figure 7.4. The same thirty-two answers, as stacked bars: for each problem, the share of the eight answers that are complete and correct, complete and wrong, or cut off at 160 tokens.

third cuts off only 2% and no longer gives any correct answer: the reward drawn at random does more than make the answers overrun the limit.

A cut-off answer counts as wrong. A model that learns to answer more briefly therefore gains pass@1 points without computing any better. Part of the gain measured with the true reward may come from this detail of the benchmark: learning to stop in time. Our logs do not measure the accuracy of the complete answers alone at each evaluation, so we cannot put a figure on this part. For Qwen, whose starting answers are cut off more than 70% of the time in the test, it may be large.

> **Lesson for a language-model benchmark.** A length limit turns a measure of accuracy into a mixed measure of accuracy and concision. A paper that reports a pass@1 gain is read together with the length limit, the share of cut-off answers before and after training and the accuracy of the complete answers. Without these three numbers, the reader does not know what the gain measures.

7.7 What this strand establishes and what it does not

This strand establishes three facts, for two small models, one multiplication task and 100 steps. With the true reward, the spread between random seeds is as large as the mean gain, as in the classical RL of 2018. A reward drawn at random makes the success of both families collapse. Part of the apparent gain may come from the length limit, which the protocol had not anticipated.

The strand establishes neither the effect of GRPO at the scale of the papers nor the absence of the effect of Shao et al. (2025) on larger models or on models specialised in mathematics. It does not separate the effect of the hardware from that of the random seed for Qwen. The limits that follow are taken up in chapter 10.

CHAPTER 8

Software and reproducibility

In 2018, Pineau asked for work that is reproducible, reusable and robust. This chapter describes how the project repository, `github.com/warith-harchaoui/pineau-rl`, tries to meet these three demands. It presents the layout of the repository, the Python package that gathers the reusable computations, the file formats, the mechanical check of every published number, the tests, continuous integration, the frozen environments, the hardware and the compute used. A last section states what the repository lacks and how to obtain it.

8.1 Repository layout

The repository follows a one-way flow, from runs to documents (figure 8.1). A run writes a raw result file: an evaluation curve in CSV format for a simulation, a JSONL log for a GRPO run, a JSON reading file for a paper of the survey. An analysis script reads these files and writes a JSON result file. Two checking scripts then recompute every number of the texts from these JSON files. Every number in a document comes from them.

Table 8.1. The directories of the repository and their role.

Path	Content
`study/`	The survey: protocol (`PROTOCOL.md`), reader's sheet (`GRILLE.md`), draws, readings (`codes/`, `codes_v2/`), harmonisation (`overrides*.json`), provenance of the readers (`readers/`), analysis scripts and JSON result files.
`sim/`	The simulations: protocol (`SIM_PROTOCOL.md`), one run (`run.py`), a re-evaluation under modified physics (`dose.py`), run lists (`jobs*.txt`), launchers and watchdog (`drive.sh`, `queue.sh`, `worker.sh`, `scale.sh`, `watchdog.sh`), curves (`results/`), logs (`logs/`), final policies (`models/`).
`sim/analysis/`	One script and one JSON file per experiment (E1 to E7, B1, B3, dose, Bonferroni correction), with their variants at twenty random seeds and at equal budget.
`sim/llm/`	The GRPO strand: protocol, `grpo.py`, launchers, diagnostics (`diag/`), logs, results and `summary.py`.
`src/pineau_rl/`	The Python package of reusable computations (section 8.2).
`tests/`	The unit tests (section 8.7).
`envs/`	The frozen environments, the platform and the document toolchain (chapter D).
`docs/`	The summary document in two languages, its bibliography, its figures and `check_numbers.py`.
`article/`	The blog article in two languages, its figures and `check_article.py`.
`report/`	This technical report, its generated tables and its figures.
`tools/`	`compare_results.py`, which compares JSON files value by value.

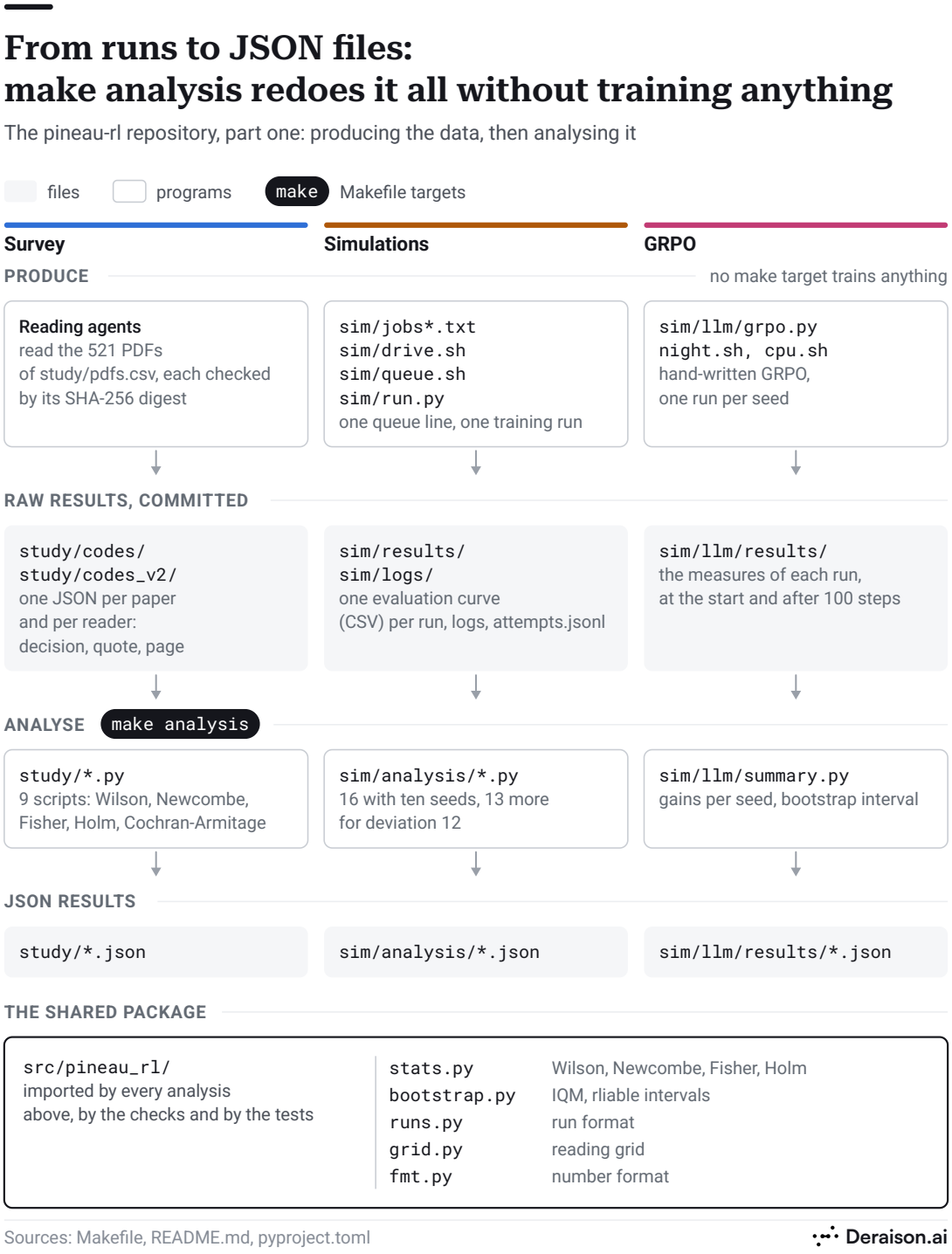


Figure 8.1. The repository, first part: from runs to JSON files. Training runs write the raw results (CSV, JSONL, logs). The analysis scripts, built on the `pineau_rl` package, derive one JSON file per experiment from them. `make analysis` redoes this whole chain without any training.

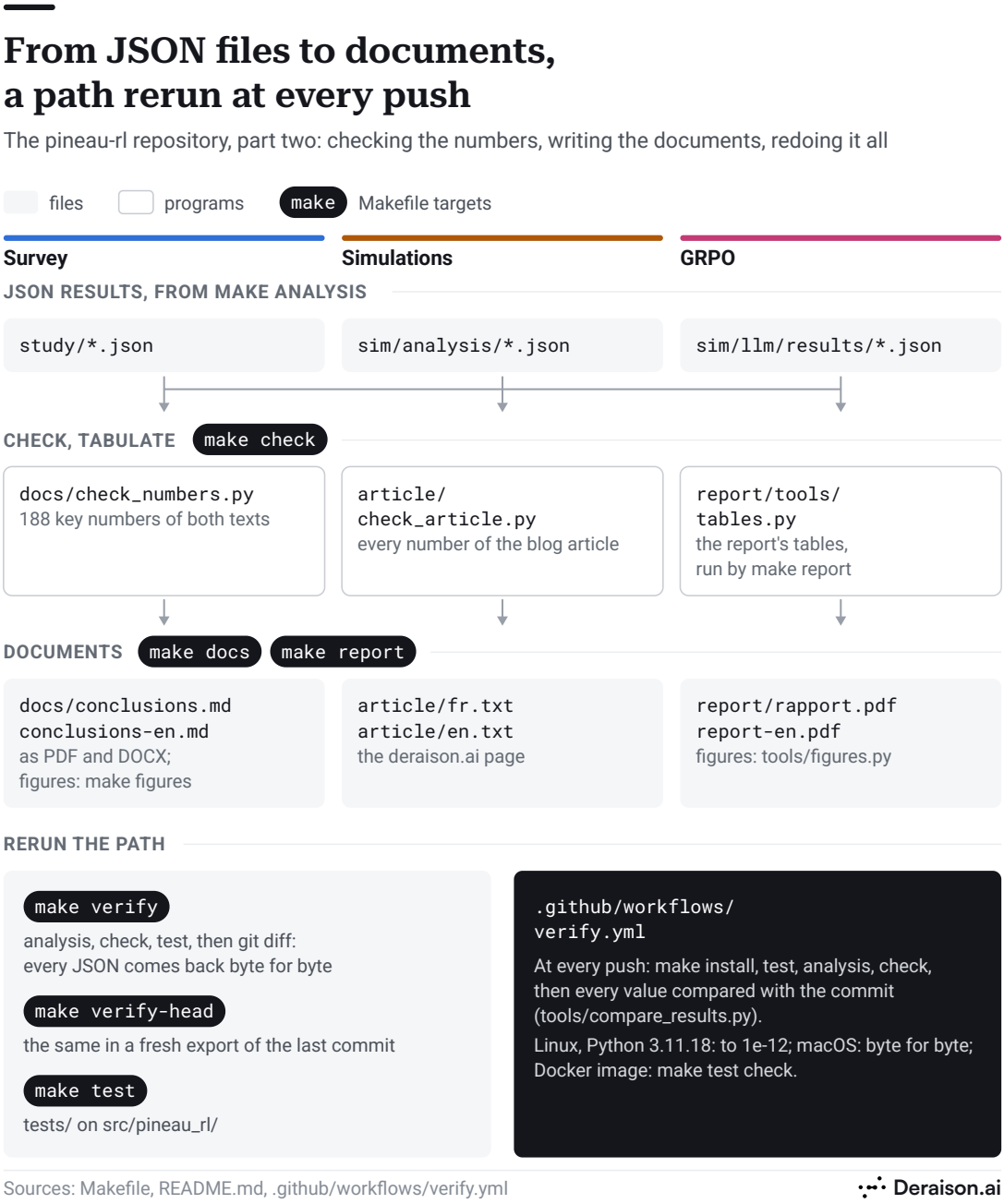


Figure 8.2. The repository, second part: from JSON files to documents. `check_numbers.py` and `check_article.py` hold every number of the texts against its JSON file, `tools/tables.py` generates the tables of this report. `make verify` chains analyses, checks and tests, then confirms that every JSON file comes back identical. Continuous integration reruns this path at every push to GitHub.

8.2 The `pineau_rl` package

The computations that serve several scripts form an installable Python package, `src/pineau_rl/`, version 1.0.0, under the BSD 3-Clause licence. Its only dependencies are NumPy, SciPy, statsmodels and rliable. Every analysis script imports it. The `Makefile` puts `src/` on Python's search path, so the scripts run without installing the package. Submodules load on demand: `import pineau_rl.stats` does not load rliable. The package has four public modules (`stats`, `bootstrap`, `runs`, `grid`) and a formatting module (`fmt`).

8.2.1 `pineau_rl.stats`

This module gathers the statistics of the survey and of the experiments. Every function is deterministic. The module header warns that the computations follow the exact order of the original scripts, so that the result files regenerate byte for byte: simplifying an expression changes the last digit of a result.

- `wilson(k, n, z=1.96)` and `wilson_ci(k, n, alpha=0.05)`: the interval of Wilson (1927) for a proportion k/n. With $\hat{p} = k/n$, its centre is $(\hat{p} + z^2/2n)/(1 + z^2/n)$ and its half-width $z\sqrt{\hat{p}(1-\hat{p})/n + z^2/4n^2}\,/(1 + z^2/n)$. The second function delegates to statsmodels, with $z = \Phi^{-1}(1-\alpha/2)$.
- `newcombe_diff(ka, na, kb, nb)`: the 95% interval for the difference of two proportions by the hybrid method of Newcombe (1998), built from the two Wilson intervals (l_a, u_a) and (l_b, u_b): $d - \sqrt{(\hat{p}_a - l_a)^2 + (u_b - \hat{p}_b)^2}$ and $d + \sqrt{(u_a - \hat{p}_a)^2 + (\hat{p}_b - l_b)^2}$, with $d = \hat{p}_a - \hat{p}_b$.
- `fisher(ka, na, kb, nb)`: the two-sided p-value of Fisher's exact test on the 2×2 table.
- `compare(a, b)`: the two previous functions combined, `{diff, lo, hi, p}`, at full precision.
- `holm(pvalues)`: the p-values adjusted by the step-down procedure of Holm (1979), $\tilde{p}_{(i)} = \min\big(1, \max_{j \le i}(m - j + 1)\, p_{(j)}\big)$.
- `cochran_armitage(k, n, x)`: the test for a linear trend in proportions (Armitage 1955). With $\bar{p} = \sum k_i / \sum n_i$ and $\bar{x} = \sum x_i n_i / \sum n_i$, the statistic is $z = \sum x_i (k_i - n_i \bar{p}) / \sqrt{\bar{p}(1-\bar{p}) \sum n_i (x_i - \bar{x})^2}$.
- `cohen_kappa(a, b)`: the kappa of Cohen (1960), $(p_o - p_e)/(1 - p_e)$, `None` when $p_e = 1$.
- `probability_of_improvement(x, y)`: $P(X > Y) + \frac{1}{2} P(X = Y)$ over all pairs of runs, which is the Mann-Whitney U statistic divided by $n_x n_y$ (Agarwal et al. 2021).
- `welch_curve(a, b)`, `welch_pooled(a, b)` and `henderson_script(M, S, a, b, n)`: the three readings of the statistic of Henderson et al. (2018) studied by experiment B1. The first applies the test of Welch (1947) at each evaluation point, then averages the t values or the p-values. The second stacks all the points of the curves as independent observations. The third redoes the computation of the script `from_same_dist.py` (Henderson 2017): a Welch test built from the means and standard deviations across episodes, into which the spread between random seeds never enters.

```
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. The computation of Henderson's script, in `pineau_rl.stats`.

8.2.2 `pineau_rl.bootstrap`

The bootstrap module wraps the stratified bootstrap intervals of rliable (Agarwal et al. 2021). `iqm(x)` returns the interquartile mean, the mean of the middle half of the scores. `iqm_interval(scores, reps=2000)` returns, for each group of runs, the interquartile mean and its 95% percentile bootstrap interval. `improvement_interval(x, y, reps=2000, confidence=0.95)` returns the probability of improvement and its interval, at the requested level: the Bonferroni analysis of deviation 18 of the simulations uses it with $1 - 0.05/24$. `probability_of_improvement_bootstrap(x, y, rng)` resamples x and y with an explicit NumPy generator, without rliable.

A defect of rliable 1.2.0 dictates how this module is used. The library accepts a `random_state` argument but draws its resamples from NumPy's global generator (deviation 13 of the simulations). A script therefore seeds this generator once, at the start (`np.random.seed(2026)`), then calls the functions in a fixed order: each call consumes the global generator, so an interval depends on the calls that precede it. The module imports rliable as soon as the module itself loads, never inside a function, so that no import happens after the random seed is set.

8.2.3 `pineau_rl.runs`

The runs module defines the format of a simulation run (section 8.3) and its readers. `run_path(root, algo, task, seed)` gives the path of a run. `read_run(path, final_step=None, upto=None)` reads a file and checks it: required columns present, steps strictly increasing. It returns `None` for a missing file or an unfinished run. The argument `upto` cuts the curve at a smaller budget, which serves the equal-budget analysis. `write_run(path, rows)` writes the format from any library. `finals(…)` returns the score of each random seed at the requested step, `load_grid(…)` the full grid `{task: {algorithm: scores}}` with the list of missing pairs.

8.2.4 `pineau_rl.grid`

The grid module reads the survey grid. It defines the lists of criteria (`V1`: Q1 to Q8 and X1 to X7; `V2`: Y1 to Y12), the integer criteria (`INT`: X1, X6, Y3), the allowed values and the kinds of uncertainty of Y11 (`KINDS`). `load_reads(root, readers)` loads the reading files, `load_overrides(path)` the harmonisation. The final value of a criterion follows one of two semantics, kept apart because the scripts differ: `value_v1` (one reading and its harmonisation) and `value` (the reading of the v2 grid when it has the criterion, otherwise that of the v1 grid, with Y3 turned into yes or no from 2 upward). `tally(values)` returns the numerator and the denominator of a share, `na` values excluded. `validate(rec, …)` checks a reading file (section 8.3).

8.2.5 `pineau_rl.fmt`

Numbers come out of this module the way the texts write them. `half_up(x, nd)` rounds half away from zero, as a human reader does ($1.15 \to 1.2$), whereas Python's `round` rounds half to even. The float goes through its shortest representation, so that 1.15 stays 1.15. `pct`, `num`, `ci` and `pts` produce a share as a percentage, an integer with a thousands separator, an interval in whole points and the signed bounds of a difference, in French or English style.

8.3 File formats

A simulation run. The file `sim/results/<algo>/<task>/seed<k>.csv` holds one run, that is one program, task and random seed, with one line per evaluation:

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

`eval_mean` and `eval_std` give the mean and the standard deviation of the returns of the ten evaluation episodes, `perturbed_mean` the mean return under modified physics (empty when not measured), `wall` the elapsed seconds. A run is finished when its last step reaches the program's budget. The two lines above are the first of PPO PyTorch on HalfCheetah, random seed 0.

A GRPO run. One JSONL file per model, reward and random seed (chapter 7). Evaluation lines contain `step`, `greedy`, `pass1`, `pass8` and `truncated`. Training lines, every ten steps, contain `train_right`, `truncated`, `loss`, `grad_norm`, `retries` and `wall`.

A paper reading. One JSON file per paper and per reader, stored under `study/codes_v2/` in the reader's directory. For each criterion, it holds the value, an exact quotation from the paper and its page, or the statement 'absent after reading the experiments, appendices and links' with a null page (chapter 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": "..."}
```

The validator. `pineau_rl.grid.validate` returns the list of problems in a reading file, empty when the file is valid. It checks the schema: every criterion present, values `yes`, `no` or `na`, integers or `null` for X1, X6 and Y3, an X5 family among `llm`, `classic` and `theory`, a `kind` field when Y11 is yes, a quotation for every criterion. It checks the scope rules of deviations 3 and 14 of the protocol: Q1 set to no forces `na` everywhere; Y2 and Y11 are `na` if and only if X2 is no; Y4 is `na` when Y3 is 1; Y5 is `na` if and only if Q4 is `na`. Given a function that searches the text of the PDF, it finally checks that every quotation appears there. The function `segments` first removes the reader's notes in square brackets, splits a quotation stitched together with '[...]' into pieces and

ignores pieces shorter than twelve letters. The script `study/check_codes.py` applies this check with the PDFs. The reader's sheet asks each reader to run it on their files until it reports zero problems.

8.4 Every number checked against its file

Two scripts check the published texts. They read no number typed by hand: they recompute it.

`docs/check_numbers.py`. For each numerical claim of the summary document, the script recomputes the value from a result file, formats it as each language writes it (`pineau_rl.fmt`), then searches the text for the resulting string, with spaces and line breaks normalised. A few survey counts that the analysis scripts do not write are recounted from the readings, with the same rules. Assertions protect the qualitative facts, for example the name of the task on which the leader changes under modified physics. The script checks 188 claims in the two languages and stops at the first missing number, so `docs/build.sh` does not compile a document in which a figure has drifted.

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

Listing 8.2. One claim of `docs/check_numbers.py`: the expected string is built from the data, never typed.

`article/check_article.py`. The script of the blog article adds a coverage check. Each claim builds, from a result file, the sentence that the two languages write, then searches the text for it. The few numbers without a file (a slide, a cited paper, a sample size) are searched for in the summary document. The script then removes from the text all checked sentences and the proper names that contain digits: any number left over is a failure. The bibliography lines are confronted with the file `docs/refs.bib`. The article has 161 checked claims in each language.

8.5 The targets of the `Makefile`

The `Makefile` trains nothing. Training goes through `sim/run.py`, `sim/drive.sh` and `sim/llm/grpo.py`.

`make install`
: installs the package in editable mode in the analysis environment, with no dependency downloaded: they come from `envs/analysis.lock`.

`make analysis`
: reruns every analysis script, in dependency order: the nine of the survey, the sixteen of the simulations at ten random seeds, the statistical analysis of the official Zoo pipeline, the variants of deviation 12 (twenty random seeds, replication of E4 on random seeds 10 to 19, equal budget), then `summary.py` at three and at five random seeds. The variants are selected through environment variables (`SEEDS`, `SUFFIX`, `BUDGET`).

`make check`
runs the two checking scripts of section 8.4.

`make test`
runs the unit tests with pytest.

`make lint`
runs ruff and mypy on the package and the tests.

`make verify`
chains `analysis`, `check` and `test`, then asks git whether every JSON result file has come back byte for byte.

`make verify-head`
does the same in a fresh export of the last commit, without the policies, the Zoo runs or the training logs, which no analysis reads. This target leaves untouched a working copy in which runs are still writing.

`make figures`
regenerates the figures of the summary document (`docs/figures/build.py`).

`make docs`
runs `check`, then builds the summary document (`docs/build.sh`).

`make report`
builds this report (`report/build.sh`): tables generated from the JSON files, figures, then LuaLaTeX and biber, in French and in English.

```
python3.11 -m venv sim/env-an
sim/env-an/bin/pip install -r envs/analysis.txt -r envs/test.txt
make verify          # analyses, numbers of the texts, tests, byte-for-byte
    diff
make verify-head     # the same in a fresh export of the last commit
```

Listing 8.3. Recomputing every number from the committed results, without training.

8.6 Continuous integration

The file `.github/workflows/verify.yml` replays the verification on every push and every pull request, on two platforms. The GitHub actions it uses are pinned to a commit hash, since a tag can be moved. Each job installs the analysis environment from `envs/analysis.lock`, with every package checked against its SHA-256 hash (`pip install --require-hashes`), then installs the package, runs the tests, reruns all the analyses, checks the numbers of the texts and compares every result file with the commit using `tools/compare_results.py`.

- On Ubuntu 24.04 (x86-64), with Python 3.11.18, the version the author used, the comparison accepts a relative difference of 10^{-12} on floats. It requires equality of counts, verdicts, rankings and all strings. The p-values move in their fifteenth or sixteenth digit with the mathematics library.
- On macOS 15 (arm64), the family of machines that produced the results, the comparison requires exact equality.

A third job builds the image of the `Dockerfile` and runs the tests and the number check in it. The image starts from `python:3.11-slim`, pinned by hash, installs the

same locked file and mounts the repository at launch instead of copying it, so the same image serves any commit.

8.7 Tests

The `tests/` directory holds eight test files and one file of shared helpers (`conftest.py`). The tests import the package and the thin layer `sim/analysis/runs.py`, never an analysis script, which computes as soon as it is imported. The tests combine three kinds of checks: values published in the literature, an independent computation (statsmodels, SciPy, NumPy) and reference values on synthetic data, frozen to a relative 10^{-12}. A reference value that changes announces a result file that changes.

`test_stats.py`
: Wilson against a published value, against statsmodels and on its properties (the interval contains the share and stays within $[0, 1]$, it is symmetric, it widens at 99%); Newcombe against a published value and rebuilt from the two Wilson intervals; Fisher against SciPy; Holm computed by hand and its properties; Cochran-Armitage equal to $\sqrt{N}$ times the Pearson correlation between score and outcome; Cohen's kappa and its list of categories; the probability of improvement.

`test_bootstrap.py`
: the interquartile mean; the rliable intervals, identical under a fixed global random seed and dependent on that random seed alone; the bootstrap probability of improvement, which uses only its own generator.

`test_henderson.py`
: the three readings of Henderson's statistic against SciPy; the p-value of the mean t with the mean Welch degrees of freedom; the computation of the script, which ignores the spread between random seeds: on six synthetic random seeds, it reaches a t of 9 where a test over the random seeds sees nothing.

`test_runs.py`
: the format of a run: writing then reading, missing or unfinished runs returned as `None`, malformed files that raise an error, final scores and reference grid.

`test_grid.py`
: the two value semantics, the conversion of Y3, the harmonisation, the validator (valid file, each problem reported, scope rules, quotations) and the splitting of quotations.

`test_formatting.py`
: rounding half away from zero, percentages, thousands, intervals and the minus sign of both languages.

`test_pipelines.py`
: the reading of runs by `sim/analysis/runs.py` and the GRPO summary from end to end, on three synthetic random seeds: an unfinished run is set aside, while the mean gain and the bootstrap interval match their expected values.

`test_draw.py`
: the 2026 draw comes back from its rule (`study/draw_2026.py`) and depends on its random seed.

8.8 Frozen environments

Four Python environments ran the project, all under Python 3.11.18. Each file in `envs/` is the `pip freeze` of one of them, taken on 15 November 2026, with the Python version and the platform at the top. The `*.lock` files add the SHA-256 hash of each package. Chapter D gives the versions. The training environment (`envs/train.txt`) serves `run.py` and `dose.py`, the analysis environment (`analysis.txt`) the analyses, the verification and continuous integration, the Zoo environment (`zoo.txt`) the official pipeline of experiment E6, the language-model environment (`llm.txt`) GRPO and the figures of the summary document. A fifth file, `test.txt`, adds pytest to the analysis environment.

8.9 The trained policies

The final policies let anyone re-evaluate an agent without retraining it, as the graded re-evaluation in modified physics did (`sim/dose.py`). Those of PPO, TRPO and experiment E6 are in the repository. The weights of SAC and CrossQ, too heavy for git, are attached to the GitHub release `policies-v1`, as tar archives listed with their SHA-256 hash in a `SHA256SUMS` file. The script `sim/fetch_models.py` downloads each archive, checks its hash, unpacks it under `models/`, then checks each policy file against `sim/models/MANIFEST.sha256`, the hashes taken on 15 November 2026 on the machine that trained them. The downloads run one after another, separated by a random pause of 1.5 to 4 seconds. A refused request waits according to an exponential backoff with full jitter, five attempts at most.

```
cd sim && python3 fetch_models.py          # download, unpack, check
    everything
cd sim && python3 fetch_models.py --check  # only check the policies already
    present
```

Listing 8.4. Fetching and checking the policies of SAC and CrossQ.

8.10 Hardware and compute

All runs ran on one laptop, a Mac with an Apple M2 Max chip under macOS 27.2 (arm64), shared with other projects. Each simulation training runs on a single thread. The number of jobs in parallel changed over the project without affecting the results, since each run is determined by its random seed: four cores at the start, up to eight granted every hour according to the load of the machine (deviation 15 of the simulations), then two cores for the whole project from 19 November 2026 (deviation 19). Section 5.11 describes the launchers, the watchdog and the job queue. Table 5.3 gives, per program and per task, the number of runs, the number of steps, the median duration of a run and the core-hours consumed.

The GRPO runs have very different durations depending on the device. According to the `wall` field of their logs at step 100, a Qwen run took 42 to 52 minutes on GPU and 4.3 to 11.4 hours on CPU, a Llama run 69 to 148 minutes on GPU. These spreads also reflect the varying load of the machine.

8.11 What is not in the repository

The PDFs of the papers read are protected by copyright. The script `study/fetch.py` downloads them again from the public sources (`cd study && python3 fetch.py final_2026.json pdf`). Pineau's slides are not in the repository for the same reason: they are on the websites of ICLR 2018 and NeurIPS 2018. The file `slides-en.md` describes them one by one. The logs of the working sessions with the agents that read the papers are not public. The exact instructions, the model and the dates of each reader, reconstructed from these logs, are public (`study/readers/`).

8.12 Licence and citation

The code, the protocols, the derived data and the documents are published under the BSD 3-Clause licence, the licence of scikit-learn (file `LICENSE`, Copyright (c) 2026, Warith Harchaoui). The file `CITATION.cff` gives the reference, which GitHub offers in APA and BibTeX.

CHAPTER 9

Deviations from the protocols

A protocol written in advance protects a result only if every later change is visible. This chapter describes the rule we followed, what a reader can check of it and what rests on our word. It then gives the complete list of deviations from the three protocols and discusses those that weigh most on the results.

9.1 The principle

The three protocols date from 8 November 2026: the survey protocol (`study/PROTOCOL.md`), written before any paper was read, the simulation protocol (`sim/SIM_PROTOCOL.md`) and the protocol of the GRPO strand (`sim/llm/LLM_PROTOCOL.md`), both written before any training. Each one fixes the question, the population or the runs, the measures and the analyses. Every later change is appended at the bottom of the file, in a numbered list of 'deviations'. The body of the protocol is never rewritten.

Each deviation records three things: its date, its reason and what we had already seen when we decided it. The third matters most. A rule fixed before a result is seen cannot have been chosen to suit that result. A rule fixed afterwards may have been. The deviations therefore say 'BEFORE' or 'AFTER', in capitals, with the exact object of the comparison: before any training, after the PPO and SAC results but before any CrossQ result, after the values at step 100 were displayed but before any gain was computed. Three rules complete the arrangement. No run is removed. A run that crashes is relaunched once with the same random seed, then the incident is recorded. Once an analysis has been announced, its result is published whatever it is.

The three protocols list 66 deviations: 29 for the survey, 20 for the simulations, 17 for the GRPO strand. Table 9.1 summarises all of them. Figure 9.1 places them in time, from 8 to 21 November 2026.

9.2 What rests on our word

A reader can recompute every number from the committed results (chapter 8) and check every survey decision against the quotation and the page it rests on. A reader cannot check, unaided, the order in which things were written.

The repository became public on 15 November 2026. Its first commit, dated 9 November at 06:07, already holds results next to the protocols. For everything before 15 November, the order of writing rests on our word and on the commit dates, which their author could have forged. A critical review of the public repository also found several 'before'/'after' statements that were wrong in their details. The corrections are dated and published: deviation 14 of the simulations, deviation 15 of the GRPO strand. They show that the rule was applied with errors. The reader learns this before deciding how far to trust the other statements.

The repository offers three supports for that trust. The evaluation logs of every run are committed, including those of stopped runs when they were not overwritten. The attempt ledger lists every training launch. The provenance of the survey readers (instructions, model, dates, papers written) is reconstructed from the transcripts and published in `study/readers/` (survey deviation 28).

9.3 The attempt ledgers

The simulation protocol states that a run that crashes is relaunched once and the incident recorded. Until 15 November, no ledger kept track of every launch. The reconstructed ledger, `sim/logs/attempts_reconstructed.csv`, rests on the creation dates of the logs, the outputs of the launchers, the committed versions and our session logs. It gives one line per attempt: algorithm, task, random seed, attempt number, start, end, outcome, score seen before the relaunch and evidence. Simulation deviation 14 (e) draws up the balance: 292 launches for the 250 training runs of the first phase, 42 attempts stopped (only 2 of them declared) and none crashed. Every stop hit all the runs in progress at once. No run was relaunched or discarded after its final score was known. For PPO PyTorch and PPO JAX, the relaunches reproduce the evaluation points of the stopped attempts identically. For SAC and CrossQ, the stopped curves were overwritten and do not allow this check.

Since 15 November, the launcher `queue.sh` keeps one log per attempt, never overwritten. It appends each attempt to the ledger `sim/logs/attempts.jsonl`: start, end, exit code, SHA-256 fingerprint of the result. The watchdog writes into the ledger every job it stops. A stop by signal is no longer relaunched automatically. The stopped GRPO runs are recorded there too.

9.4 The list of deviations

Table 9.1. The 66 deviations from the three protocols, in the order of their lists: survey (`study/PROTOCOL-en.md`), experiments (`sim/SIM_PROTOCOL-en.md`), GRPO strand (`sim/llm/LLM_PROTOCOL-en.md`). Date: day of November 2026 given by the protocol ('undated' when the deviation carries no date; survey deviations 1 to 10 were noted during coding, before aggregation). Timing: what the protocol declares had already been seen when the deviation was written (before or after the results concerned, or no effect on the results for the organisation of the computation). The summary takes one line; the full text, which prevails, is in the protocol (the French files are the dated originals). The number of deviations, their numbering and their dates are checked against the files by `report/tools/tables.py`.

No.	Date	Timing	Summary
Survey (`study/PROTOCOL-en.md`, 29 deviations)			
1	undated	during coding, before aggregation	X4 harmonised: `yes` requires a described tuning of the baselines; 'same configuration' with no tuning becomes `no`.
2	undated	during coding, before aggregation	X7 harmonised: `yes` requires a quantity of compute (hours, GPU-hours, FLOP); the hardware name alone is `no`.
3	undated	during coding, before aggregation	Papers with no experiment (Q1 `no`): Q2 to Q8 and X1 to X4 set to `na`, out of the denominators.
4	undated	during coding, before aggregation	Q7 for a theory paper with toy experiments: `no`, not `na`.

continued on next page

Table 9.1, continued

No.	Date	Timing	Summary
5	undated	during coding, before aggregation	X3: a selection unstated but visible in the paper's own tables (LR10) counts as `yes`.
6	undated	during coding, before aggregation	Q5: code 'to be made public' is `no`; a project page is `yes` only if it leads to the code.
7	undated	during coding, before aggregation	Calibration on 12 papers from 2018: our grid stricter than Pineau's on Q3, more lenient on Q7; a change must hold against both references.
8	undated	during coding, before aggregation	Separate 2018 appendices read (ICML, NeurIPS); in 2026 they are inside the PDF; C21 has no published appendix.
9	undated	during coding, before aggregation	Contact address removed from the User-Agent header of the downloads; no effect on the data.
10	undated	during coding, before aggregation	Working scripts overwritten in the readers' shared space; the codings, in separate folders, are untouched.
11	9 Nov.	before the results	Grid v2 (Y1 to Y12) for the flaws grid v1 leaves unmeasured; Y11 and Y12 added the same day, before any coding.
12	9 Nov.	before the results	Enlargement: 2018 brought to 50 papers (A2), classical RL 2026 to 50 (A3, title screen); analysis fixed (Newcombe, Fisher, Holm); double reading of 10 papers.
13	9 Nov.	before the results	Blind human review of 10 papers planned (A1), abandoned in deviation 23.
14	9 Nov.	before the results	Decision rules of grid v2 (scope, edge cases of Y1 to Y12); sheet `GRILLE.md`, check `check_codes.py`.
15	9 Nov.	after the results	Rules of deviations 1, 2 and 4 applied to the 12 papers from 2018 as well, left out until then; no new rule.
16	9 Nov.	before the results	Pilot of three papers (LR08, ML15, C218) to find ambiguous rules; its readings enter no analysis.
17	9 Nov.	before the results	Rules clarified after the pilot (2018 appendices, Y1, Y3, Y5 to Y9, Y11, format); 9 unreachable ICLR 2026 papers skipped in the A3 queue.
18	9 Nov.	after reading A3, before any percentage	X5 (family) read on the model the RL trains, to decide which A3 papers enter the classical sample.
19	9 Nov.	after the audit of the readings, before any percentage	Harmonisation of grid v2: 80 codes changed, each with its reason in `overrides_v2.json`, for both years.
20	9 Nov.	after the results	Verdict rule: only the same-grid comparison (classical RL 2026 against our reading of 2018) decides; Pineau's figures remain context.
21	10 Nov.	after the results, before this computation	Exploratory checks (provenance, PDF source, Mantel-Haenszel, reader removed) and blind test of the cost and risk thesis.
22	10 Nov.	before the results	Checklists of the 30 NeurIPS 2025 papers read, extracted and set against our codes (A5).
23	10 Nov.	before the results	Human review (A1) abandoned, decision of Warith Harchaoui; stated among the limitations.
24	10 Nov.	before the results	Sensitivity to the title filter (A6): 400 papers outside the filter, abstracts read, then 20 hidden RL papers read with both grids.
25	10 Nov.	before the results	Trajectory from 2018 to 2026 (A4): 20 classical RL papers per year from ICML and NeurIPS 2019, 2020, 2022 and 2024; Cochran-Armitage.
26	11 Nov.	before the results	Blind second reading of 10 A4 papers (reader R3).
27	15 Nov.	after the results, before any search	Keyword search against the 'no' decisions of the main comparison, blind arbiter; decisions turned, then doubtful ones.

continued on next page

Table 9.1, continued

No.	Date	Timing	Summary
28	15 Nov.	after the results	Provenance of the readers published (`readers/`): two readers finished on `claude-opus-5` (13 papers); the readings do not change.
29	15 Nov.	after the results, before this computation	Main comparison and trajectory redone without the 13 papers read partly by `claude-opus-5`.
Experiments (`sim/SIM_PROTOCOL-en.md`, 20 deviations)			
1	undated	before the first training run	SAC and CrossQ: 200,000 steps instead of 300,000, for lack of compute; TD3 not launched.
2	undated	after the results	E6: an error on our side (normalisation gamma 0.99 instead of 0.999); corrected recipe rerun, diagnosis on Hopper v4 and v5.
3	9 Nov.	after the results	E1: a second statistic, the mean t along the curve as in Henderson et al. (`e1_curve.py`).
4	9 Nov.	before any TRPO run	B1: 30 TRPO runs on HalfCheetah and three readings of Henderson et al.'s statistic.
5	9 Nov.	after `e6_diag`, before any official run	E6: ten agents trained with the Zoo's official script, evaluated with our protocol.
6	9 Nov.	after `b1_trpo`, before this computation	B1: the script Henderson published in 2017 (`from_same_dist.py`) applied to our curves.
7	10 Nov.	after PPO and SAC, before looking at CrossQ	Analysis details of E2, E4 and E5 (final score, IQM, probability of improvement, reversals, draws).
8	10 Nov.	after PPO and SAC, before looking at CrossQ	B3: three measures (final score, best score, mean of the curve) and ranking reversals.
9	11 Nov.	no effect on the results	Organisation: the second worker takes the CrossQ queue from the other end.
10	12 Nov.	no effect on the results	Stop by the watchdog (swap); two CrossQ runs relaunched from scratch.
11	13 Nov.	after the results	End of the simulations; descriptive complement to E7 (ranking by IQM in both physics, no test).
12	15 Nov.	before any new run	Consolidation: seeds 10 to 19 (160 runs), E7 at several strengths, reading at equal budget (200,000 steps).
13	15 Nov.	after the results	Intervals made reproducible: rliable ignores `random_state`, NumPy's global generator is seeded; verdicts unchanged.
14	15 Nov.	after the results	Corrections after a critical review: date of `dose.py`, counts in deviations 7 and 8, SB3 2.9.2, actual steps, undeclared relaunches, seeds read, ledger.
15	15 Nov.	no effect on the results	Up to eight cores granted to the project depending on the machine's load.
16	16 Nov.	no effect on the results	Night of 15 to 16 November: stop by the watchdog, surviving worker fixed, battery rule.
17	16 Nov.	no effect on the results	Battery condition lifted.
18	16 Nov.	after the results, before this computation	E2: the 24 probability-of-improvement intervals made simultaneous by Bonferroni (99.79%).
19	19 Nov.	no effect on the results	Cap of two cores for the whole project; watchdog restarted.

continued on next page

Table 9.1, continued

No.	Date	Timing	Summary
20	21 Nov.	after the runs, before any computation on seeds 10 to 19	Computation of the additions of deviation 12; defect of `e5.py` fixed (right order over all the chosen seeds).
GRPO (`sim/llm/LLM_PROTOCOL-en.md`, 17 deviations)			
1	undated	after reading outputs, before any training	Length limit raised from 64 to 160 tokens.
2	undated	before any training	Measures reduced to 150 problems for greedy decoding and 50 for sampling, for cost.
3	undated	after the pilot, before any training	Task `mul2` chosen; main measure: sampled pass@1.
4	8 Nov.	report, no retained result	The learning-rate pilot saturates memory (swap); killed, no usable result.
5	9 Nov.	report, no retained result	Diagnosis without training: 23 answers out of 32 cut off at 160 tokens at step 10 of the pilot.
6	9 Nov.	before any retained run	Reduced version: 100 steps, seeds 0 to 2, learning rate 3e-6, updates in batches of 8 answers, memory cap.
7	9 Nov.	rule fixed before any result for the seed	Pause at Warith's request; `nan` crash of a Qwen run; a single relaunch planned.
8	9 Nov.	report, no retained result	Resumption: none of the nine remaining runs completes (`nan`, macOS cut-off, MPS memory).
9	9 Nov.	before any result of these runs	Night relaunch (`night.sh`) with a control run first.
10	10 Nov.	report, no retained result	Cause of the `nan` found: non-finite gradients on MPS, clean on CPU; recomputation safeguards.
11	10 Nov.	before any retained run with this code	The remaining Qwen runs move to CPU; Llama stays on GPU at night.
12	10-11 Nov.	corrections before any relaunch	Qwen finished on CPU; the six Llama runs run out of MPS memory; cache emptied at every step, cap raised.
13	11-12 Nov.	statistic details set after the results	All twelve runs finished; percentile bootstrap over the seeds, 10,000 draws.
14	15 Nov.	before any new run	Five seeds: seeds 3 and 4 added for both models and both rewards.
15	15 Nov.	after the results	Corrections after a critical review (counts of deviation 8, dates); no result changes.
16	17 Nov.	after the step-100 values, before any gain	Llama, random reward, seed 4 fails twice: counted as a result; statistic over the finished runs, sensitivity to the missing gain.
17	19 Nov.	no effect on the results	The Llama CPU run starts after the queue, under the two-core cap.

9.5 Deviations of the survey

Harmonisation (deviations 1 to 6, 15 and 19). The decision rules of the protocol left borderline cases open. Deviations 1 to 6, recorded during reading and before any percentage, fix the most frequent ones: X4 requires a described setting for the baselines, X7 an amount of compute, Q5 a link that leads to code. Each change is recoded from the quotation, in a harmonisation file (`overrides.json`, `overrides_v2.json`) that gives the reason for every modified value. Agreement between readers is always computed on the raw readings, before harmonisation. Deviation 15 corrects a mistake of ours,



Figure 9.1. The protocols and their dated deviations, from 8 to 21 November 2026. One line per protocol (survey, simulations, GRPO strand). Each mark is a deviation, placed at its date. Its shape shows whether it was written before or after the results it concerns.

seen after the percentages: the harmonisation rules had been applied only to the 2026 papers. The same rules, with no new rule added, were then applied to the twelve 2018 papers of the calibration. Deviation 19 applies the harmonisation of grid v2 to both years, after an audit of the raw readings and before any percentage: 80 changes, each with its reason. For X3, X4 and X7, no new rule was added, so that both years are read with the same grid. Borderline cases stay coded as the reader read them.

Grid v2 and its planning (deviations 11, 14, 16 and 17). Grid v1 measured only part of the flaws. Grid v2, Y1 to Y12, was written on 9 November 2026 before any v2 reading (deviation 11), its decision rules the same day (deviation 14). They were made more precise after a pilot on three papers (deviations 16 and 17), still before the main coding. The work plan that precedes these deviations, `study/planification_v2.md`, was kept as it was and made public on 15 November 2026. For each flaw, it states which v2 criteria are meant to move the verdict out of the 'to be measured' state.

The enlargement (deviation 12). The power audit of the first draw shows that twelve 2018 papers and fourteen classical RL papers from 2026 are too few to detect a gap of thirty points. Deviation 12, written before any draw and before any of the added papers was read, raises each of the two samples to 50 papers and fixes the main comparison: classical RL in 2026 against our own reading of 2018, with the same grids. The papers of the 2026 queue go through a title screen that sets aside titles presumed to concern large language models. Chapter 10 discusses its effect.

The verdict rule (deviation 20). Of all the survey deviations, this one weighs most on the conclusions. It was decided after the first results of the enlarged reading had been seen. Until then, a criterion counted as changed when the gap held against Pineau's figures. The secondary comparison shows that our grid reads 2018 differently from her: 55% against 90% for the hyperparameters of the method (Q3), 33% against 10% for the held-out evaluation (Q7). Comparing 2026 with her figures would mix the change in practices with the change in reading. Since this deviation, a verdict rests only on the main comparison of deviation 12, fixed before any of the added papers was read. The comparisons with Pineau are still shown, as context. The change of rule is declared as decided after the results. The comparison it retains had been fixed before them.

The human review dropped (deviations 13 and 23). Deviation 13 planned for Warith Harchaoui to reread ten 2026 papers alone, blind, on seven criteria. Deviation 23 dropped that plan on 10 November 2026, before any rereading. As a result, no human reread the papers. Every decision can still be checked against its quotation. The folder `study/relecture_humaine/` is kept as a trace of the plan.

Analyses written in response to review (deviations 21, 22, 24 to 29). After the results, several analyses were added to answer objections. Each was written before its own computation and published whatever it showed: the robustness cuts and the test of the cost thesis (deviation 21), the checklists declared by the authors (deviation 22), the title filter (deviation 24), the yearly trajectory and its second reading (deviations 25 and 26), the mechanical search against the 'no' decisions (deviation 27), the provenance of the readers (deviation 28) and the removal of the thirteen papers partly read by another model (deviation 29). Deviation 21 removed one reading from the document: the thesis 'Pineau was heard where it cost little' fails the correlation test that the deviation announced (chapter 4).

9.6 Deviations of the simulations

The budget (deviation 1). SAC and CrossQ train for 200,000 steps instead of 300,000, a decision taken before the first training to keep the compute within reach. PPO keeps its million steps. This deviation creates the comparison at unequal budgets, which deviation 12 (iii) then corrects with a reading at equal budget.

B3, written along the way (deviation 8). Experiment B3, which asks whether the measure (final score, best score, mean of the curve) changes the ranking, was written on 10 November 2026, after the PPO and SAC results. It trains nothing: it rereads the curves already recorded. Deviation 8 dated it 'BEFORE any CrossQ result'. Deviation 14 (b) corrects that account, since seven CrossQ runs had finished when B3 was written. Our session logs, which are not public, show that no CrossQ score had been read. The exact wording becomes 'before looking at any CrossQ result'. Deviation 7, which fixes the details of E2, E4 and E5, receives the same correction.

The descriptive ranking under modified physics (deviation 11). The most quoted reversal of the blog article, PPO JAX first on Hopper in the training physics and last in the modified physics, comes from a complement decided after the E7 results had been seen, without a test (`analysis/e7_rank.py`). The document presents it as such. The graded re-evaluation of deviation 12 (ii), written before any new computation, then put this reversal to the test: PPO JAX is no longer first on Hopper in 10 of the 12 modified physics.

The consolidation (deviation 12). Written on 15 November 2026, before any new training and any new evaluation, this deviation adds three analyses with the same code and the same settings: twenty random seeds per pair (160 training runs), the re-evaluation of every policy under masses and friction modified by degrees, plus the reading of PPO at 200,000 steps, the budget of SAC and CrossQ. The results at ten random seeds remain the main results. Chapter 6 gives both next to each other and states every divergence.

The random seed of the bootstrap (deviation 13). While the public repository was being prepared, six scripts did not reproduce their intervals identically. rliable ignores its `random_state` argument and draws from NumPy's global generator. Each script now seeds this generator at the start. Two runs now give files identical byte for byte. The point estimates and the verdicts do not change. The bounds move at the level of Monte Carlo noise, by at most 16 points on an interquartile mean of 2,423 for E6.

Seven corrections after a critical review (deviation 14). A hostile review of the public repository established seven points, none of which changes a verdict. (a) The commit that described the dose script as 'written before the dose results' was wrong: 30 of the 160 re-evaluations already existed. The script applies a rule written before any computation, with no free parameter. (b) The counts of finished CrossQ runs in deviations 7 and 8 were wrong. (c) The protocol announced Stable-Baselines3 2.6.0. Every training ran with version 2.9.2, installed before the first training. (d) PPO and TRPO collect in batches of 2,048 steps, so each block of 50,000 steps is in fact 51,200 steps: the PPO and TRPO models have 1,024,000 steps, against 200,000 for SAC and CrossQ. (e) The undeclared relaunches, described in section 9.3. (f) Six scripts read every file present. With random seeds 10 to 19, E6 would have gone from ten to twenty

random seeds without saying so. Each script now names its random seeds. (g) The ledger `attempts.jsonl` and the per-attempt logs, from 15 November on.

Organising the compute (deviations 9, 10, 15 to 17 and 19). These deviations record stops by the watchdog, relaunches, the move to eight cores and then to two, plus a suspension tied to the Mac's battery, lifted the same morning. None changes a result: each run uses a single thread and depends only on its own random seed.

Computing the additions (deviation 20). Written on 21 November 2026, after every training had finished and before any computation on random seeds 10 to 19, this deviation fixes a defect of `analysis/e5.py`, which took the mean of the first ten runs as the 'correct' ordering even with twenty random seeds. It then lists the computations, all part of `make analysis`. Two of them, the descriptive ranking and the Bonferroni correction at twenty random seeds, were not in deviation 12: they apply rules already written and are published whatever they show.

9.7 Deviations of the GRPO strand

The plan reduced, then restored (deviations 6 and 14). Deviation 6, decided before any retained run, reduces the plan to 100 steps and three random seeds, sets the learning rate to 3×10^{-6} without a full pilot and splits the update into sub-batches. Deviation 14, written on 15 November 2026 before any new run, adds random seeds 3 and 4. Of all the decisions, this restoration changed the reading of the strand most: Qwen's gain with the true reward, clear at three random seeds, is no longer clear at five (chapter 7).

The crashes and their cause (deviations 7 to 11). The first retained runs crashed on non-finite probabilities or ran out of GPU memory. Deviations 7 and 8 set a relaunch rule, decided before any result of these runs was seen. Deviation 9 moves the runs to the night. Deviation 10 finds the cause: a GPU computation interrupted by macOS returns with corrupted values. It adds the safeguards of `grpo.py`. It also resets the failure count to zero, since the past crashes came from the machine. This change of rule was decided after the crashes were seen, at a time when no result of these runs existed. Deviation 11 moves the remaining Qwen runs to the CPU. It is the origin of the confounding between hardware and random seed discussed in section 7.5.3.

Corrections after review (deviation 15). Written on 15 November 2026, after the results, this deviation corrects the breakdown of the nine failures of 9 November and the date of one run, pins the exact revision of both models in `grpo.py`, documents the stops of 15 November and commits to comparing the CPU runs alone, since the device changes between Qwen's random seeds.

The run that fails twice (deviation 16). This deviation was written on 17 November 2026, after the step-100 values of the seven other new runs had been displayed during a progress check, but before any gain or interval at five random seeds was computed. The Llama run with the randomly drawn reward and random seed 4 fails twice on the gradient safeguard. Under deviation 14, this second failure is a result. The configuration is summarised over its four finished runs, with a sensitivity analysis for the missing gain. The deviation also plans one CPU run outside the protocol, whose result is kept apart.

The order of the CPU run (deviation 17). With the two-core ceiling of simulation deviation 19, the Llama CPU run starts after all the other computations. It has no effect on the results.

9.8 What these deviations say about the work

Most deviations have one of three causes: the compute available on one laptop, our own mistakes, objections raised in review. The deviations decided after the results are declared as such. The three that weigh most on a conclusion are the verdict rule of the survey (deviation 20), the descriptive ranking under modified physics (simulation deviation 11) and the reset of the GRPO failure count (deviation 10). For the first, the retained comparison had been fixed before the results. The second was put to the test by an analysis written in advance. The third affects no published number, since no crashed run had a result. A reader who rejects these three decisions can redo every computation without them from the committed files.

CHAPTER 10

Limitations and threats to validity

This chapter gathers the limitations of the work, so that no reader has to hunt for them. It sorts them into the four families of threats of Shadish et al. (2002). Internal validity asks whether the measured effect really comes from the presumed cause. External validity asks to which populations, tasks and conditions the result extends. Construct validity asks whether the measure captures the notion it claims to capture. Statistical conclusion validity asks whether the tests and intervals license the conclusion drawn. A final section states what a new complete training run on another machine would test.

10.1 Internal validity

10.1.1 The survey readers are language-model agents

Every paper was read by language-model agents, copies of one model (`claude-opus-5-5`). Two readers finished on a neighbouring model (`claude-opus-5`) for thirteen papers, after a response was stopped by a content filter of the provider (survey deviation 28). No human reread the papers blind: the rereading planned by deviation 13 was dropped before it began (deviation 23). Five safeguards reduce the risk without removing it.

1. **The mandatory quotation.** Every ‘yes’ decision rests on an exact quotation and its page. Anyone can check every decision against the PDF. The script `study/check_codes.py` checks mechanically that every quotation appears in the text.
2. **The quotation audit.** After the first percentages, we looked for the quotations that the mechanical check did not find (deviation 15, file `study/v1_quotes_audit.md`). Of 89 quotations, none is invented: 83 appear word for word in the paper, one is a faithful paraphrase and 5 give values that contradict the rules.
3. **The blind double readings.** Twenty papers of the survey were reread by a second agent that did not see the first reading. On the ten reread papers of the first draw, the two readers agree on 127 decisions out of 130 (98%). For every criterion, two agents agree on at least 80% of the papers, with one exception: the performance measure (Y1), at 60% on the first draw, before its rules were made more precise. Ten papers of the yearly trajectory, reread in the same way, agree at 80% or more on each of the 23 criteria (table 4.7).
4. **The search against the ‘no’ decisions.** A ‘no’ rests on no quotation. Every ‘no’ on the verdict criteria was confronted with the text of the paper through keywords fixed in advance. Each excerpt found was then judged by an arbiter that saw neither the paper nor the year (deviation 27). Of 377 ‘no’ decisions, 10 become ‘yes’ (2 in 2018, 8 in 2026) and no verdict changes. If the 24 decisions judged doubtful are flipped as well, uncertainty (X2) and sensitivity (Y9) end up just above the threshold ($p = 0.07$ and $p = 0.054$).

5. **Removing the neighbouring model.** Without the thirteen papers partly read by `claude-opus-5`, the verdicts hold, except reported uncertainty (X2), which no longer passes the Holm correction ($p = 0.07$, deviation 29).

These safeguards leave one threat whole. The two readers of a double reading are copies of the same model: their agreement measures the stability of that model, which differs from the agreement of two independent readers. A systematic bias of the model, shared by all its copies, escapes the double readings. The exact instructions, the model and the dates of every reader are published (`study/readers/`), so that another reader, human or not, can redo the reading.

10.1.2 Decisions taken after the results

Several rules were fixed after results had been seen: the verdict rule of the survey (deviation 20), the harmonisation of the 2018 calibration (deviation 15), the descriptive ranking under modified physics (simulation deviation 11), the details of the GRPO bootstrap (deviation 13) and the reset of the GRPO failure count (deviation 10). Chapter 9 discusses each of them. The order in which the deviations before 15 November 2026 were written rests on our word and on the commit dates (section 9.2).

10.1.3 The simulation runs

The reconstructed ledger counts 42 stopped attempts that the protocol did not declare (section 9.3). Every stop hit all the runs in progress at once, with no link to a score. For SAC and CrossQ, the curves of the stopped attempts were overwritten: we cannot show that the relaunch reproduces the same curve. The protocol announced Stable-Baselines3 2.6.0. Every training ran with 2.9.2, the same version for all runs. These deviations favour no program. They are stated here so that the reader knows them.

10.1.4 Hardware and random seed in the GRPO strand

For Qwen, the hardware changed with the random seed: the two true-reward runs that show the gain ran on the GPU, the three that do not show it ran on the CPU. Five runs cannot separate these two causes (section 7.5.3). The first three runs ran before the safeguards against a corrupted computation existed. A gradient that was corrupted but finite, clipped to 1, may have gone through them without a trace. The MPS computation is not deterministic from one attempt to the next, so the same random seed does not guarantee the same run on the GPU.

10.2 External validity

10.2.1 The population of papers

- **The population has changed.** 27 of the 50 papers of the first 2026 draw deal with large language models, a family absent in 2018. Our verdicts apply to classical RL.
- **The title filter.** The regular expression applied to titles misses RL papers whose title does not say so, in 2018 as in 2026. We measured this for 2026 with a plan

written in advance (deviation 24). Of 400 papers drawn outside the filter, 25 have an abstract that mentions RL, which comes to about 1,037 missed papers (from 708 to 1,503), 45% of the whole. This is an upper bound, since the mechanical rule also counts false positives. Read with both grids, 20 of these papers resemble the first draw: code linked in 60% of them (67% in the draw), an uncertainty in 50% (47%). The 7 classical RL papers among them are too few to test our verdicts.

- **The title screen of the 2026 queue.** A classical RL paper whose title contains ‘reasoning’, ‘preference’ or ‘math’ could not enter the classical 2026 sample. The 9 ICLR 2026 papers at the head of the queue, unreachable on 9 November 2026, were skipped.
- **The source of the PDFs.** The 2026 papers were read mostly in their arXiv version (45 of the 50 classical RL papers), the 2018 papers in the conference proceedings. An arXiv version is sometimes longer or more recent. With only 5 papers read on a conference site, this effect cannot be measured.
- **The trajectory without ICLR.** The years 2019 to 2024 cover only ICML and NeurIPS: OpenReview refuses automated downloads.
- **Pineau’s draw.** Her 2018 sampling method is not described. Our comparisons with her figures assume a draw comparable to ours and serve only as context.

10.2.2 The simulations

The experiments are limited to the MuJoCo simulator and to four locomotion tasks (HalfCheetah, Hopper, Walker2d, Swimmer), those of Henderson et al. (2018). They say nothing about image-based games, real robotics or offline RL. E1 and B1 are limited to one task (HalfCheetah) and two algorithms (PPO and TRPO). The four programs run with their default hyperparameters, except in E6: another tuning might change the rankings. The modified physics changes only two quantities, the masses and the friction.

10.2.3 The budgets

PPO trains for 1,024,000 steps, SAC and CrossQ for 200,000 (simulation deviations 1 and 14 (d)). The equal-budget reading (deviation 12 (iii)) compares PPO at 200,000 steps, read off its one-million-step curve: a PPO tuned for a budget of 200,000 steps might do better. The budget of SAC and CrossQ, reduced to keep the compute within reach, places them in the ‘little data’ regime. A longer budget might change their order.

10.2.4 One platform

Every run comes from one Mac with an Apple M2 Max chip, shared with other projects, under macOS and for arm64. MuJoCo, JAX and PyTorch round differently from one processor to another: the same random seed gives another curve on another machine (section 10.5).

10.2.5 The scale of the GRPO strand

The GRPO strand is an illustration: two models of 0.5 and 1 billion parameters, one multiplication task, 100 steps, groups of 8 answers, answers limited to 160 tokens, no KL term. Shao et al. (2025) work on 7-billion-parameter models specialised in mathematics, on MATH-500, with far more compute. Our strand neither replicates nor refutes their experiment.

10.3 Construct validity

- **Our grid reads 2018 differently from Pineau.** Our reading finds 55% of 2018 papers giving all the hyperparameters of their method, where Pineau counted 90%. The criteria share their wording but have different implicit decision rules. This is the reason for the same-grid verdict rule (deviation 20).
- **A satisfied criterion does not mean a practice done well.** Q5 is yes for a link to code, with no check that the code runs. X2 is yes for an error bar, with no judgement of whether it measures the right spread: the bar may summarise episodes instead of random seeds, which experiment B1 shows to be dangerous. Q8 is yes for an applied test, with no judgement of whether the test is appropriate.
- **The performance measure (Y1)** is our weakest criterion: 60% agreement on the first draw, before its rules were made more precise. The verdict on flaw F8 is read with this caveat.
- **The borderline cases of X3, X4 and X7** received no new harmonisation rule, so that both years are read with the same grid. A compute figure limited to inference time, or failed random seeds that were discarded, stay coded as the reader read them.
- **What a checklist declares.** The authors' answers to the NeurIPS checklist do not guarantee the practice they name: none of the 18 papers that answer yes to the question on statistical significance applies a test. The survey measures what the paper does, the checklist what it declares.
- **The score of a simulation run** is the mean return of ten deterministic episodes at the last step. Experiment B3 shows that another measure (best score, mean of the curve) changes the order of some pairs.
- **The pass@1 of the GRPO strand** mixes correctness and concision, since a truncated answer counts as wrong (section 7.6).
- **The cost and risk ratings** that tested the cost thesis come from three language-model agents of three different sizes. Their agreement with one another does not guarantee that a researcher would rate the same way.

10.4 Statistical conclusion validity

The sample sizes of the survey. With 50 papers on each side, a change of ten points still often goes unnoticed: 'no detectable change' does not mean 'no change'. The first draw had 14 classical RL papers, too few to detect a gap of thirty points, hence the

enlargement of deviation 12. The trajectory rests on 20 papers per year: a share there has an uncertainty of about twenty points on either side.

Multiple comparisons. The survey compares seventeen criteria in the main comparison. The Holm correction protects the family-wise error rate at the cost of power: a gap whose Newcombe interval excludes zero without passing Holm, such as Q6 or Y12, is reported but makes no verdict. For E2, 24 intervals of the probability of improvement are also given at the simultaneous 95% level through the Bonferroni correction (simulation deviation 18, table 6.4).

The fragile verdicts. The rise of reported uncertainty (X2) no longer passes the Holm correction in two checks ($p = 0.07$ each time). It remains the most fragile of the four changes. Code and compute pass the threshold in all our checks, sensitivity to a hyperparameter in nearly all of them (table 4.3).

The number of random seeds in the simulations. The main results rest on ten random seeds per pair. Experiment E5 shows what this number allows: between two programs whose means differ by less than 20%, five random seeds recover the right order only 71 times in a hundred at the median, 62 times as measured on twenty random seeds. On the tasks where the first program does not clearly beat the second, the rankings at ten random seeds are no more reliable than that. The twenty random seeds of deviation 12 confirm or qualify each result. Chapter 6 states every divergence.

The number of random seeds in the GRPO strand. With five random seeds, the bootstrap has only 126 distinct resamples, ten with three random seeds. The intervals of chapter 7 are very coarse. An interval that contains zero says that we cannot conclude. It does not say that the effect is zero.

The bootstrap intervals. The rliable intervals depend on NumPy's global generator and on the order of the calls. Their bounds move with the random seed of the bootstrap, at the level of Monte Carlo noise (simulation deviation 13): a verdict whose bound grazes the threshold stays fragile.

10.5 What a new training run on another machine would test

A new complete training run on another processor would not reproduce our curves. MuJoCo, JAX and PyTorch round differently from one processor to another, so the same random seed gives another trajectory from the first steps on. The GRPO strand illustrates this: for the same model and the same random seed, the starting pass@1 differs between GPU and CPU (section 7.5.3).

Such a training run would therefore test the statistics of the report. Its curves would differ from ours. It would tell whether the spread between random seeds, the rankings, the inflation due to selection and the losses under modified physics stay of the same order with other draws. A conclusion that survived only our own random seeds would say nothing about the algorithms. Continuous integration already runs the equivalent test for the analyses: on Linux, the counts, the verdicts and the rankings come back exactly, the floating-point values to within 10^{-12} in relative terms (section 8.6). No training was redone on another machine.

CHAPTER 11

Conclusion

11.1 The answer

> Joelle Pineau was heard on what is published next to the result: the code, the compute, the effect of a setting. She was heard much less on how the result itself is judged. The statistical test remains rare, the settings of the baselines remain poorly described, evaluation still often takes place in the training world.

This answer applies to classical RL, read by language-model agents in fifty papers on each side, then compared with our own reading of 2018 using the same grids (chapters 3 and 4). It also rests on experiments rerun with the algorithms of 2026 under protocols written in advance (chapters 5 to 7).

11.2 The main numbers

Four practices have changed. A link to the code goes from 31% to 74% of papers, the sensitivity to a setting shown from 44% to 80%, the compute stated in figures from 27% to 58%, a reported uncertainty from 63% to 88% (tables 4.1 and 4.2). Code and compute survive every cut we tried, sensitivity nearly every one. Uncertainty is fragile: it no longer passes the Holm correction once the thirteen papers partly read by another model are removed ($p = 0.07$). Read on 20 papers per year between the two dates, these four practices rise slowly, with no jump right after the checklist that NeurIPS has required since 2019 (table 4.6).

Others have not moved detectably. A statistical test goes from 6% to 12%, baselines described with all their settings from 35% to 29%, a held-out evaluation from 33% to 45%, the version of the benchmarks from 23% to 26%. Nearly every paper now draws error bars, very few apply a test.

What a ticked box guarantees. The 18 NeurIPS 2025 papers that answer yes to the checklist question on statistical significance apply no test. On code, 5 papers out of 22 answer yes without a link to their code. On compute, 6 papers out of 21 answer yes without giving a compute time.

Random seeds and selection. Over thirty PPO runs on HalfCheetah, two groups of five drawn at random fool a statistical test in only 1.2% of cases. Sorted, they fool it in 31% of cases. Keeping the best runs inflates the score by 33% (tables 6.1 and 6.6). The t of 9 from 2018 most probably comes from the script published by Peter Henderson, which measures the spread of the episodes and ignores the spread of the random seeds. Applied to our TRPO runs, the script exceeds 9 in 38% of random draws (table 6.2).

How far the rankings reach. Across our four programs, the best one changes with the task. Yet PPO has five times more steps than SAC and CrossQ: at equal budget and with twenty random seeds, CrossQ leads on all four tasks (table 6.3). A physics modified by 20% overturns the ranking on Hopper: the first program finishes last (table 6.9).

The language-model strand. With five random seeds, Qwen's mean gain with the true reward is $+14$ points, from -3 to $+34$, Llama's $+6$ points, from -0.4 to $+15$: both intervals contain zero. Qwen's gain rests on the two runs done on the GPU. The hardware changed with the random seed. With a coin-flip reward, success falls to 1% or less in the nine finished runs. Part of the apparent gain may come from the 160-token limit (table 7.1).

11.3 Recommendations

The following recommendations follow from the measurements of this report. Each names the flaw it targets and the result that justifies it.

11.3.1 For authors

1. **State the number of random seeds and run at least ten.** Of 41 experimental RL papers in the first 2026 draw, 20 do not say how many runs they launch. The median of the others is five. With five random seeds, a gap of less than 20% reads backwards nearly one time in three, more than one time in three when twenty random seeds set the true order (E5, flaws F9 and F10).
2. **Never sort or select runs.** Report every run, including those that fail. State which were excluded (E1, E4, criteria X3 and Y6).
3. **Test across random seeds.** The independent unit is the complete training run. A test that counts episodes instead declares a difference between two groups of the same code (B1, flaw F11).
4. **Show an interval next to every curve and say what it represents.** Name the standard deviation, the standard error or the confidence interval, then the spread it summarises (criteria X2, Y2, Y11).
5. **Compare at equal budget, or state the difference in budget.** A ranking at unequal budgets also compares the budgets (E2 at equal budget).
6. **Tune the baselines with the same care as the method and say so.** Give the tuning budget of each baseline and its final hyperparameters (E6, criteria Q4, X4, Y5).
7. **Evaluate under held-out conditions.** Environment random seeds never seen, modified dynamics or a distinct test distribution (E7, criteria Q7, Y8).
8. **Give the versions.** The version of each benchmark and of each dependency, in a frozen environment file (criteria Y7, Y10).
9. **For a language model: several model families and the share of truncated answers.** A result obtained on one family informs about that family. A gain in correctness is read together with the length limit, the truncated share before and after and the correctness of the complete answers (chapter 7, criterion X6).

10. **Write the protocol before the first run and date every deviation.** For each change, state what had already been seen (chapter 9).
11. **Recompute every published number from its files.** A script that checks every number in the text against its result file costs little and catches drift (section 8.4).

11.3.2 For reviewers

The seven questions of the blog article are meant for any reader. A reviewer can ask them of every submission.

1. How many random seeds? With fewer than five runs, or with 'the best runs', the result says little.
2. What uncertainty? A mean with no interval and no band around the curve does not allow a gap to be judged.
3. Which baseline, tuned how? If the method got a month of tuning and the baseline its default values, the gap measures the month of tuning.
4. What budget? Comparing two methods trained on different numbers of steps also compares those numbers of steps.
5. Evaluated where? An agent judged on the situations of its training may have learned them by heart.
6. What code? Without published code, the reader does not know what the paper really did.
7. What compute? A result that nobody else can afford to recompute cannot be reproduced.

The reviewer can add an eighth question, which this report justifies: which unit does the test compare? A test over episodes, or over the stacked points of a curve, does not measure the variability between training runs.

11.3.3 For conferences

1. **Ask for the test as well as the bars.** The NeurIPS question on statistical significance accepts error bars 'or other appropriate information'. The 18 papers that answer yes to it apply no test. A question that asks for the unit of the test and the number of independent training runs would demand more.
2. **Check a ticked box on a sample.** Some papers answer yes on code with no link, on compute with no figure. A check drawn at random on a share of the accepted papers would measure the gap between declaration and practice.
3. **Require the link to the code in the camera-ready version.** 'Code will be released' is no substitute for a link.
4. **Ask for the version of each benchmark.** This practice has not moved detectably since 2018 (23% then 26%), although the three AI agents that rated the cost of each practice (chapter 4) judge it nearly free.
5. **Support reproducibility challenges.** The ICLR 2018 challenge presented by Pineau showed that reproduction by third parties is feasible: among the participants who answered the survey, 54.5% reproduced part of a paper and 32.7%

most of it. Running it again by broad families of results, with a protocol written in advance, would measure what the checklist does not.

11.4 Future work

- **A blind human rereading** of a sample of papers, on the verdict criteria, would measure the agreement between a human and our readers, which the double readings between copies of one model cannot measure. The plan of survey deviation 13 is still ready.
- **A second family of readers**, another AI model, would reread the same papers with the same instructions, published in `study/readers/`.
- **ICLR in the trajectory.** The years 2019 to 2024 lack ICLR. A reading of its proceedings, through a download route that OpenReview accepts, would complete the series.
- **The papers that apply what they call RL to large language models, read with the same grid** on a sample of comparable size, would tell whether held-out evaluation, which is progressing there, comes with the other practices.
- **A new complete training run on another platform**, Linux on x86-64, would test the statistics of our experiments with other draws (section 10.5).
- **Other families of tasks** than MuJoCo locomotion: image-based games, offline RL, real robotics, where the conclusions of E2 to E7 have yet to be tested.
- **The GRPO strand on one device**, with more random seeds, a wider length limit and the correctness of complete answers measured at every evaluation, would separate the effect of the hardware from that of the random seed. The same protocol on 7-billion-parameter models would bring the strand closer to Shao et al. (2025).

In 2018, Pineau asked researchers to be as critical of their good results as of their bad ones. This report tried to hold to that. While checking our surprising results, we found two traps: an error in our code in E6, whose correction changed nothing, then the length limit of the GRPO strand. Two more random seeds undid the Qwen gain that we read at three random seeds. The repository keeps a dated record of every correction for anyone who wants to redo the work.

APPENDIX A

The complete reading grid

This appendix reproduces the reading grid of the survey, criterion by criterion, with the question put to the reader and its decision rule. It combines the reader's sheet (`study/GRILLE.md`) and the rules of the protocol (`study/PROTOCOL.md`): deviations 1 to 6 for the harmonisation of grid v1, deviations 11, 14, 17 and 19 for grid v2. When in doubt, the protocol prevails.

A.1 What the reader does

The reader reads the whole paper, appendices included, before filling in one JSON file per paper (section 8.3). For each criterion, the file gives the value, an exact quotation from the paper, copied word for word from the PDF, then the page. If nothing justifies a `yes`, the quotation is 'absent après lecture des sections expériences, annexes et liens' (absent after reading the experiments, appendices and links sections) and the page is `null`. A quotation spread over two pages receives a list of pages. In a two-column PDF, a quotation that runs across lines is split into one-line pieces joined by '[...]'. The reader never sees another reader's codes, the results or the synthesis document. They run `study/check_codes.py` on their own files and fix them until it reports zero problems. The possible values are `yes`, `no` and `na` (not applicable), except for the integer criteria and for the family X5.

A.2 Pineau's grid (Q1 to Q8)

Joelle Pineau presented these eight criteria at NeurIPS 2018 (slides 29 and 30), with the share of the 50 RL papers from 2018 she had read that met each one. Her sampling method is not described. The labels below are hers, word for word.

Table A.1. Pineau's eight criteria and her 2018 figures.

Code	Pineau's label	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. Grid v1, Pineau's criteria: question and decision rule.

Code	Question	Decision rule
Q1	Does the paper have experiments?	no: every other criterion is na and leaves the denominators (deviation 3). Pineau's denominators are the experimental papers.
Q2	Does it use neural networks?	Read directly.
Q3	Are all the hyperparameters of the method given?	yes if a table or a paragraph gives them for all the main methods. 'Default values of [library]' without listing them counts yes only if the version is given.
Q4	Are all the hyperparameters of the baselines given?	Same rule as Q3. na if no baseline is compared.
Q5	Is the code linked?	A link to code in the paper. 'On request', 'will be released', 'will be made public at <URL>': no. A project page counts yes only if it leads to code, checked by visiting it on 9 November 2026. An anonymous mirror that holds the code counts yes (deviations 6 and 19).
Q6	Is the way the hyperparameters are chosen described?	The selection method is stated: grid, random search, budget, criterion. 'Tuned' and nothing more: no.
Q7	Does evaluation run on held-out conditions?	Tasks, environments, environment random seeds or datasets seen neither in training nor in tuning. For an llm paper, benchmarks distinct from the RL training data count yes. A theory paper with toy experiments: no (deviation 4). New training random seeds, used to train, are not held-out conditions (deviation 17).
Q8	Is a statistical test applied to a comparison?	A test (t, Welch, Mann-Whitney, bootstrap with interval, rliable interquartile mean with stratified intervals) applied to a comparison. Error bars alone: no for Q8, yes for X2.

A.3 Our additions in grid v1 (X1 to X7)

These criteria come from the ICLR 2018 talk and from the 2025 debate on the spurious rewards of language models (section 7.2). They were written with the protocol, on 8 November 2026, before any reading.

Table A.3. Grid v1, our additions: question and decision rule.

Code	Question	Decision rule
X1	How many independent training runs behind the main results?	An integer, or null if not stated: the smallest number announced for a main table or curve. For an llm paper, count the independent RL training runs alone: pass@k and '16 samples' are evaluation samples, which do not count.
X2	Is an uncertainty reported on the main results?	Standard deviation, standard error, confidence interval, band around the curve.

Code	Question	Decision rule
X3	Are the best runs selected?	'Top-k', 'best of N', best checkpoint chosen on the test set. `yes` also when the paper's own tables show that the main result is the best of several identical training runs, even if the text does not say so (deviation 5).
X4	Does the paper say that the baselines are tuned with the same care as the method?	`yes` requires a tuning procedure described for the baselines, or an equal tuning budget announced. 'Same configuration as the method', with no tuning described for anyone: `no` (deviation 1).
X5	Which family does the paper belong to?	`llm` if the optimised policy is a pretrained generative model conditioned on language (a language model of any size, a vision-language model, a text-to-image model). `classic` if RL trains something else, even in the service of a large model, or a transformer trained from scratch on RL tasks. `theory` for a paper with no experiments or with toy experiments only. A paper with theoretical results that compares methods on standard benchmarks, even in an appendix, is `classic` (deviation 18).
X6	For `llm`: how many distinct base-model families?	An integer, or `null`. Qwen, Llama, Mistral, Gemma, DeepSeek, OLMo... Two sizes of Qwen count as one family.
X7	Is the compute used quantified?	`yes` requires at least one quantity: hours, GPU-hours, FLOP, or a number of GPUs with a duration. The name of the hardware alone: `no` (deviation 2).

A.4 Grid v2 (Y1 to Y12)

Grid v2 was written on 9 November 2026, before any v2 reading (deviation 11), together with its decision rules (deviation 14). The rules were made more precise after a pilot on three papers (deviation 17), then harmonised after the audit of the raw readings, before any percentage (deviation 19). Each criterion points to the 2018 source that motivates it.

Scope rules. If Q1 is `no`, Y1 to Y12 are `na`. Y2 and Y11 are `na` if and only if X2 is `no`. Y4 is `na` if Y3 is `1`. Y5 is `na` if and only if Q4 is `na`. For a paper already read with grid v1, the reader receives the v1 values of Q1, Q4 and X2, which set these rules, without seeing the other v1 codes.

Table A.4. Grid v2: question, source and decision rule.

Code	Question	Decision rule
Y1	Is the performance measure defined? (ICLR 2018, slide 27)	yes if the paper states the quantity and the point of measurement: final return, mean over the last N episodes, best point, area under the curve, pass@k with k and the number of samples, success rate and over what. 'Return' or 'accuracy' and nothing more: no. A normalised score with no statement of the point at which it is taken: no. The number of evaluation episodes is not required (deviation 17). A curve whose evaluation points are stated counts as a point of measurement. 'Trained for N steps' followed by one figure counts as the final point. For a quantity other than the score of a trained policy (error of an estimator, compute time), yes if the quantity is defined (deviation 19).
Y2	Are the error bars defined? (checklist v1.2)	yes if the paper states what the band or the interval represents: standard deviation, standard error, 95% confidence interval, minimum and maximum. Bars present but undefined: no.
Y3	How many tasks does the main conclusion cover? (Henderson et al., table 3)	An integer: the number of distinct training setups on which the method is trained and compared, an environment or a training dataset for `llm`. Two difficulty levels of one game, or the quality variants of a dataset from one environment (D4RL medium, replay, expert), count as one task. Several evaluation benchmarks of one trained model do not multiply the tasks (deviation 17). A single meta-training counts as one task. A setup present only in an appendix does not count (deviation 19). Derived value for the analysis: yes if the integer is at least 2.
Y4	Is a task where the method does not win shown? (Henderson et al., table 3)	yes if a task-by-task result is given and at least one task where the method does not win is shown or stated. no if the paper gives only means, or shows only tasks where the method wins.
Y5	Is the implementation of the baselines identified? (Henderson et al., figure 6)	yes if, for each main baseline, the paper says where the code comes from: a named repository, library or software framework, with or without version, 'reimplemented by us', figures copied from papers with their source, a released model designated by its exact name, the authors' training framework if the baselines are said to run in it (deviations 17 and 19). A single main baseline with no origin gives no.
Y6	Are exclusions stated? (checklist v1.2)	yes if the paper says which runs, random seeds or data were removed, or says explicitly that none was. An omitted combination with its reason counts (deviation 17). A filtering of the training data stated with what it removes counts too (deviation 19). Silence is no.

Code	Question	Decision rule
Y7	Is the version of the environments or benchmarks given? (checklist v1.2)	`yes` if the version of each main environment or benchmark is given: Gym or Gymnasium suffix, suite version, exact name and date of the test split for `llm`. 'MuJoCo' alone: `no`. 'MATH-500' counts `yes`, 'MATH' alone `no`. A dataset published on a named repository counts as versioned, a split built by the authors only if it is published (deviation 17). A name that covers several sets without saying which ('AMC', 'Minerva') does not count. An environment fully specified in the paper counts (deviation 19).
Y8	Is a generalisation test outside the training conditions carried out? (NeurIPS 2018, roadmap)	`yes` if at least one evaluation runs on conditions absent from training and tuning: other environment random seeds announced as such, new tasks, modified dynamics, a distinct test distribution. Stricter than Q7: for `llm`, a different benchmark counts only if it is presented as an out-of-distribution test. A test split from the same distribution falls under Q7. Y8 `yes` presupposes Q7 `yes` (deviation 19).
Y9	Is the sensitivity to at least one hyperparameter shown? (ICLR 2018, slides 24 to 26)	`yes` if at least one hyperparameter of the method, presented as such, is swept and the result is shown through a performance measure. A component ablation does not count. A parameter set at inference counts if it is presented as a hyperparameter. A condition of the environment does not count (deviations 17 and 19).
Y10	Are the software dependencies listed with their versions? (checklist v1.2)	`yes` if the versions of the main libraries are in the paper, or if an environment file is cited in the linked repository. The repository is not visited, unless the paper points explicitly to the file.
Y11	Is the type of uncertainty named? (comparable to Gorsane et al., 2022)	`yes` or `no`, plus a `kind` field: `std`, `se`, `ci`, `iqr`, `minmax` or `other`. Broader than Y2: it is enough that the type is named somewhere. Percentile bands other than the interquartile range: `other` (deviation 17).
Y12	Is a statistical tool from after 2018 used? (adoption of Agarwal et al., 2021)	`yes` if the paper uses at least one of these tools: interquartile mean, stratified bootstrap interval, rliable, performance profiles, probability of improvement. A plain bootstrap on a mean counts if it produces a reported interval.

A.5 Harmonisation

A harmonisation rule is written into the protocol, dated, before the first percentage it affects. It applies to both years, from the quotations, in `study/overrides.json` (grid v1) or `study/overrides_v2.json` (grid v2), each modified value with its reason. Agreement between readers is computed on the raw readings, before harmonisation. For X3, X4 and X7, no rule was added after the enlarged reading, so that both years are still read with the same grid (deviation 19). The function `pineau_rl.grid.validate` checks the schema and the scope rules of every file (section 8.3).

APPENDIX B

Supplementary tables

This appendix gives the tables that do not appear in the chapters, then an index of every generated table and the list of result files they come from. Each generated table is written by `report/tools/tables.py` from a JSON result file: no number in it is typed by hand.

B.1 RL on large language models in the survey

The first 2026 draw holds 27 papers that apply what they call RL to large language models (family X5 `llm`, chapter A). The verdicts of the survey apply to classical RL. Table 4.5 sets these papers against classical RL in 2026, criterion by criterion, where the result file allows it. The 18 `llm` papers read along the way in the queue of the enlargement are left out: the title screen had filtered them, so they do not form a random sample (survey deviation 12). The table reads as a description, with no verdict: it compares two families of papers from the same year.

B.2 Index of the generated tables

Table B.1. Where to find each generated table.

Table	Content	Source file
Table 4.1	Pineau's criteria, 2018 against classical RL in 2026	`extension_stats.json`, `results_final.json`
Table 4.2	Our criteria Y1 to Y12, same columns	`extension_stats.json`
Table 4.5	RL on large models against classical RL, 2026	`extension_stats.json`
Table 4.6	Share per practice and per year, trend test	`trajectory.json`
Table 4.7	Agreement of the blind double readings	`agree_v2.json`
Table 4.3	Robustness checks of the survey	`robustness.json` and variants
Table 6.1	E1: thirty PPO runs, false positives	`e1.json`
Table 6.2	B1: the t of 9.09, two ways of computing it	`b1_script.json`, `b1_trpo.json`
Table 6.3	E2: ranking per task, ten and twenty random seeds, equal budget	`e2*.json`
Table 6.4	E2 with Bonferroni intervals	`e2_bonferroni*.json`
Table 6.5	E3: PPO PyTorch against PPO JAX	`e3*.json`
Table 6.6	E4: the best three out of ten	`e4.json`, `e4_r10.json`
Table 6.7	E5: how many random seeds recover the order	`e5*.json`

Table	Content	Source file
Table 6.8	E6: the tuned baseline	`e6*.json`
Table 6.9	E7: score kept under modified physics	`e7*.json`
Table 6.10	Physics modified by degrees	`dose*.json`
Table 6.11	B3: does the measure change the ranking	`b3*.json`
Table 7.1	GRPO: every run	`summary_5.json`, `gains_5.json`, `extra_5.json`
Table 9.1	The deviations of the three protocols	the three protocols
Table D.2	Versions of the environments	`envs/*.lock`
Table 5.3	Compute per program and per task	`sim/logs`, `sim/results`
Table C.1	The 2026 papers read	`final_2026.json` and queue A3
Table C.2	The 2018 papers read	`calib_2018_v2.json`

B.3 The result files

Each JSON result file comes from one script, rerun by `make analysis` (section 8.5). The criterion-by-criterion and pair-by-pair details that the tables summarise are in these files.

Table B.2. The analysis scripts and the content of their JSON file.

Script	Content of the result file
Survey, folder `study/`	
`analyze.py`	Shares in the first 2026 draw (`results_2026.json`).
`analyze_final.py`	Shares for 2026 and for the 2018 calibration, from the raw readings and the harmonisation (`results_final.json`).
`audit_stats.py`	Statistical audit of the comparison between 2018 and 2026: Newcombe differences, Fisher, Holm, reader agreement (`audit_stats.json`).
`agree_v2.py`	Agreement between readers on the raw readings, before harmonisation (`agree_v2.json`).
`extension_stats.py`	Main comparison: classical RL in 2026 against our reading of 2018, criterion by criterion, with Wilson, Newcombe, Fisher and Holm (`extension_stats.json`, with variants without the neighbouring model and with the flipped decisions).
`robustness.py`	Robustness cuts and test of the cost thesis (`robustness.json`).
`title_filter.py`, `title_filter_compare.py`	The title filter: share of hidden RL and comparison of the twenty hidden papers (`title_filter*.json`).
`trajectory.py`	The trajectory from 2018 to 2026: shares per year, Cochran-Armitage, immediate and delayed gaps (`trajectory.json`).
`falsify_no.py`	The search against the ‘no’ decisions (`falsify_no.json`).
`checklist.py`	The checklists declared at NeurIPS 2025 against our readings (`checklist.json`).
Simulations, folder `sim/analysis/`	
`e1.py`, `e1_curve.py`	E1: thirty PPO runs, Welch test on the final scores, then on the curves.

Script	Content of the result file
`b1_trpo.py`, `b1_script.py`	B1: the readings of the 2018 average t on TRPO and PPO, then the computation of Henderson's script.
`e2.py`, `e2_bonferroni.py`	E2: interquartile mean per task, probabilities of improvement, changes of ranking, then the same intervals at the simultaneous Bonferroni level.
`e3.py`	E3: the two implementations of PPO.
`e4.py`	E4: mean and spread of the ten runs, then of the best three, reversals.
`e5.py`	E5: share of draws that recover the order, for 3, 5, 10 and 20 random seeds.
`e6.py`, `e6_diag.py`, `e6_official.py`	E6: default against Zoo, diagnosis of the recipe, official policies evaluated with our protocol.
`e7.py`, `e7_rank.py`	E7: share of score kept under modified physics, then the descriptive ranking.
`dose.py`	The physics modified by degrees.
`b3.py`	B3: three measures per run and the reversals between measures.
GRPO strand, folder `sim/llm/`	
`summary.py`	Pass@1 at steps 0 and 100 of every run, gains, intervals, sensitivity to the missing gain, CPU runs alone (`summary*.json`, `gains*.json`, `extra*.json`).

The suffixes of the simulation files follow deviation 12: `_s20` for twenty random seeds, `_r10` for the replication of E4 on random seeds 10 to 19, `_equal` and `_equal_s20` for the equal budget. The files of the GRPO strand follow deviation 14: no suffix for three random seeds, `_5` for five.

APPENDIX C

The papers read

This appendix lists the papers of the two samples of the main comparison, with their identifier, title and venue. The identifier points to the reading file `study/codes/<reader>/<id>.json` or `study/codes_v2/<reader>/<id>.json`, where every decision rests on a quotation and its page. The PDFs are not in the repository (section 8.11). The script `study/fetch.py` downloads them from the addresses recorded in the sample files.

C.1 The 2026 papers

The population consists of the 1,252 papers of ICLR 2026, ICML 2026 and NeurIPS 2025 whose title matches the regular expression of the protocol, out of 17,836 accepted papers (chapter 3). The prefix of the identifier shows where the paper comes from.

- `LR`, `ML`, `NE`: the first draw (`random.Random(2018)`), 17 papers from ICLR 2026, then ICML 2026 and NeurIPS 2025 (`study/final_2026.json`).
- `MLR`, `NER`: the eight replacement papers from ICML and NeurIPS, drawn by `random.Random(2026)` in the same conference when the drawn paper had no freely accessible PDF (`replacements_2026.json`).
- `K`: the queue of the enlargement (A3), shuffled by `random.Random(2026)` and read in that order, after the title screen, until 36 classical RL papers had been read. The complete queue is in `classic_queue_2026.json`.

The classical RL sample of 2026 in the main comparison combines the 14 `classic` papers of the first draw and the 36 of queue A3. Table C.1 lists the papers read.

Table C.1. The 2026 papers read for the survey. First draw: 50 papers drawn at random from the 1,252 whose title mentions reinforcement learning (ICLR 2026, ICML 2026, NeurIPS 2025). A3 queue: the other papers of the population in the order of a `random.Random(2026)` shuffle, after a title screen that sets aside large language models (deviation 12); 70 papers read, of which the first 36 classical RL papers form, with the 14 classical papers of the first draw, the sample of 50 of the main comparison. Large-model and theory papers read before the last paper retained stay apart (18 and 3); 13 papers read after it enter no analysis. Family: criterion X5 harmonised (deviation 18), read on the model the RL trains. Identifiers: those of the reading files `study/codes/` and `study/codes_v2/`, where each decision quotes the paper with its page. The 80 papers of the 2019 to 2024 trajectory are listed in `study/trajectory_queue.json`.

Id	Title	Venue	Family, role
First random draw (50 papers, `study/final_2026.json`)			
LR01	Optimal Robust Subsidy Policies for Irrational Agent in Principal-Agent MDPs	ICLR 2026	theory, no experiment
LR02	Beware Untrusted Simulators – Reward-Free Backdoor Attacks in Reinforcement Learning	ICLR 2026	classical
LR03	MOBODY: Model-Based Off-Dynamics Offline Reinforcement Learning	ICLR 2026	classical

continued on next page

Table C.1, continued

Id	Title	Venue	Family, role
LR04	The Art of Scaling Reinforcement Learning Compute for LLMs	ICLR 2026	large models
LR05	SPIKE-RL: Video-LLMs meet Bayesian Surprise	ICLR 2026	large models
LR06	Near-Optimal Sample Complexity Bounds for Constrained Average-Reward MDPs	ICLR 2026	theory, no experiment
LR07	DeepSearch: Overcome the Bottleneck of Reinforcement Learning with Verifiable Rewards via Tree-based Search	ICLR 2026	large models
LR08	TROLL: Trust Regions Improve Reinforcement Learning for Large Language Models	ICLR 2026	large models
LR09	Wavelet Predictive Representations for Non-Stationary Reinforcement Learning	ICLR 2026	classical
LR10	Agnostics: Learning to Synthesize Code in Any Programming Language with a Universal Reinforcement Learning Environment	ICLR 2026	large models
LR11	Predictive CVaR Q-learning	ICLR 2026	theory
LR12	GAS: Enhancing Reward-Cost Balance of Generative Model-assisted Offline Safe RL	ICLR 2026	classical
LR13	MemAgent: Reshaping Long-Context LLM with Multi-Conv RL-based Memory Agent	ICLR 2026	large models
LR14	TimeSearch-R: Adaptive Temporal Search for Long-Form Video Understanding via Self-Verification Reinforcement Learning	ICLR 2026	large models
LR15	Asynchronous Policy Gradient Aggregation for Efficient Distributed Reinforcement Learning	ICLR 2026	classical
LR16	VeriRole: Verifiable Role-Awareness through Hint-Guided Reinforcement Learning	ICLR 2026	large models
LR17	MedVR: Annotation-Free Medical Visual Reasoning via Agentic Reinforcement Learning	ICLR 2026	large models
ML01	dTRPO : Trajectory Reduction in Policy Optimization of Diffusion Large Language Models	ICML 2026	large models
ML03	CoCoEdit: Content-Consistent Image Editing via Region Regularized Reinforcement Learning	ICML 2026	large models
ML06	MEAL: A Benchmark for Continual Multi-Agent Reinforcement Learning	ICML 2026	classical
ML07	PRPO: Paragraph-level Policy Optimization for Vision-Language Deepfake Detection	ICML 2026	large models
ML08	Towards Practical World Model-based Reinforcement Learning for Vision-Language-Action Models	ICML 2026	large models
ML10	Data- and Variance-dependent Regret Bounds for Online Tabular MDPs	ICML 2026	theory, no experiment
ML11	Randomized Advantage Transformation (RAT): Computing Natural Policy Gradients via Direct Backpropagation	ICML 2026	classical
ML12	VideoVeritas: AI-Generated Video Detection via Perception Pretext Reinforcement Learning	ICML 2026	large models
ML13	GenCircuit-RL: Reinforcement Learning from Hierarchical Verification for Genetic Circuit Design	ICML 2026	large models
ML14	Position: Deployed Reinforcement Learning should be Continual	ICML 2026	theory
ML15	Moment Matching Q-Learning	ICML 2026	classical
ML16	Causal Flow Q-Learning for Robust Offline Reinforcement Learning	ICML 2026	classical
ML17	InfoPO: Information-Driven Policy Optimization for User-Centric Agents	ICML 2026	large models
NE02	VisionThink: Smart and Efficient Vision Language Model via Reinforcement Learning	NeurIPS 2025	large models
NE04	OpenVLThinker: Complex Vision-Language Reasoning via Iterative SFT-RL Cycles	NeurIPS 2025	large models
NE05	Router-R1: Teaching LLMs Multi-Round Routing and Aggregation via Reinforcement Learning	NeurIPS 2025	large models
NE06	SATURN: SAT-based Reinforcement Learning to Unleash LLMs Reasoning	NeurIPS 2025	large models
NE07	Finite-Sample Analysis of Policy Evaluation for Robust Average Reward Reinforcement Learning	NeurIPS 2025	theory, no experiment
NE08	Pixel Reasoner: Incentivizing Pixel Space Reasoning via Curiosity-Driven Reinforcement Learning	NeurIPS 2025	large models
NE10	Towards Robust Zero-Shot Reinforcement Learning	NeurIPS 2025	classical
NE12	Rainbow Delay Compensation: A Multi-Agent Reinforcement Learning Framework for Mitigating Observation Delays	NeurIPS 2025	classical

continued on next page

Table C.1, continued

Id	Title	Venue	Family, role
NE13	Planning and Learning in Average Risk-aware MDPs	NeurIPS 2025	theory
NE14	Beyond Scalar Rewards: An Axiomatic Framework for Lexicographic MDPs	NeurIPS 2025	theory, no experiment
NE15	Information-Theoretic Reward Decomposition for Generalizable RLHF	NeurIPS 2025	large models
NE16	Sharp Analysis for KL-Regularized Contextual Bandits and RLHF	NeurIPS 2025	theory
MLR02	REAL: Regression-Aware Reinforcement Learning for LLM-as-a-Judge	ICML 2026	large models
MLR03	Stabilizing Reinforcement Learning for Diffusion Language Models	ICML 2026	large models
MLR06	Estimation of Treatment Effects Under Nonstationarity via the Truncated Policy Gradient Estimator	ICML 2026	classical
MLR07	Text Before Vision: Staged Knowledge Injection Matters for Agentic RLVR in Ultra-High-Resolution Remote Sensing Understanding	ICML 2026	large models
NER01	DAPO: An Open-Source LLM Reinforcement Learning System at Scale	NeurIPS 2025	large models
NER02	Text-to-Decision Agent: Offline Meta-Reinforcement Learning from Natural Language Supervision	NeurIPS 2025	classical
NER04	Asymmetric REINFORCE for off-Policy Reinforcement Learning: Balancing positive and negative rewards	NeurIPS 2025	large models
NER06	FairDICE: Fairness-Driven Offline Multi-Objective Reinforcement Learning	NeurIPS 2025	classical
A3 queue, papers read (70 papers, `study/classic_queue_2026.json`)			
K002	MolEditRL: Structure-Preserving Molecular Editing via Discrete Diffusion and Reinforcement Learning	ICLR 2026	classical sample
K003	Refined Analysis of Entropy-Regularized Actor-Critic	ICML 2026	apart (theory)
K004	Spotlight on Token Perception for Multimodal Reinforcement Learning	ICLR 2026	apart (large models)
K006	Joint Selection for Large-Scale Pre-Training Data via Policy Gradient-based Mask Learning	ICLR 2026	classical sample
K012	Deep RL Needs Deep Behavior Analysis: Exploring Implicit Planning by Model-Free Agents in Open-Ended Environments	NeurIPS 2025	classical sample
K014	Principled RL for Flow Matching Emerges from the Chunk-level Policy Optimization	ICML 2026	apart (large models)
K015	Tracking Drift: Variation-Aware Entropy Scheduling for Non-Stationary Reinforcement Learning	ICML 2026	classical sample
K016	Safety Generalization Under Distribution Shift in Safe Reinforcement Learning: A Diabetes Testbed	ICML 2026	classical sample
K017	Long-Horizon Model-Based Offline Reinforcement Learning Without Explicit Conservatism	ICML 2026	classical sample
K021	Beyond Distributions: Geometric Action Control for Continuous Reinforcement Learning	ICLR 2026	classical sample
K023	Sample-Efficient Tabular Self-Play for Offline Robust Reinforcement Learning	NeurIPS 2025	apart (theory)
K028	PoLi-RL: A Point-to-List Reinforcement Learning Framework for Conditional Semantic Textual Similarity	ICLR 2026	apart (large models)
K030	State Entropy Regularization for Robust Reinforcement Learning	NeurIPS 2025	classical sample
K032	Unbiased Dynamic Pruning for Efficient Group-Based Policy Optimization	ICML 2026	apart (large models)
K035	Learning Interactive World Model for Object-Centric Reinforcement Learning	NeurIPS 2025	classical sample
K036	Q-Insight: Understanding Image Quality via Visual Reinforcement Learning	NeurIPS 2025	apart (large models)
K037	MARC: Memory-Augmented RL Token Compression for Efficient Video Understanding	ICLR 2026	apart (large models)
K038	1000 Layer Networks for Self-Supervised RL: Scaling Depth Can Enable New Goal-Reaching Capabilities	NeurIPS 2025	classical sample
K039	DeepDiver: Adaptive Web-Search Intensity Scaling via Reinforcement Learning	NeurIPS 2025	apart (large models)
K042	Squeezing More from the Stream : Learning Representation Online for Streaming Reinforcement Learning	ICML 2026	classical sample

continued on next page

Table C.1, continued

Id	Title	Venue	Family, role
K043	Intrinsic Benefits of Categorical Distributional Loss: Uncertainty-aware Regularized Exploration in Reinforcement Learning	NeurIPS 2025	classical sample
K044	From Verifiable Dot to Reward Chain: Harnessing Verifiable Reference-based Rewards for Reinforcement Learning of Open-ended Generation	ICLR 2026	apart (large models)
K045	From Observations to Events: Event-Aware World Models for Reinforcement Learning	ICLR 2026	classical sample
K047	Adaptive Surrogate Gradients for Sequential Reinforcement Learning in Spiking Neural Networks	NeurIPS 2025	classical sample
K049	Safe Continuous-time Multi-Agent Reinforcement Learning via Epigraph Form	ICLR 2026	classical sample
K051	Hierarchical Reinforcement Learning for Sparse-Reward Search in Commutative Algebra	ICML 2026	classical sample
K052	RL-SPH: Learning to Achieve Feasible Solutions for Integer Linear Programs	ICML 2026	classical sample
K054	When Can You Poison Rewards? A Tight Characterization of Reward Poisoning in Linear MDPs	ICML 2026	classical sample
K055	Towards Large-Scale In-Context Reinforcement Learning by Meta-Training in Randomized Worlds	NeurIPS 2025	classical sample
K056	RAMAC: Multimodal Risk-Aware Offline Reinforcement Learning and the Role of Behavior Regularization	ICML 2026	classical sample
K058	Adaptive Neighborhood-Constrained Q Learning for Offline Reinforcement Learning	NeurIPS 2025	classical sample
K059	ReTabSyn: Realistic Tabular Data Synthesis via Reinforcement Learning	ICML 2026	apart (large models)
K060	ARM-FM: Automated Reward Machines via Foundation Models for Compositional Reinforcement Learning	ICLR 2026	classical sample
K061	Enhancing Geometric Perception in VLMs via Translator-Guided Reinforcement Learning	ICLR 2026	apart (large models)
K062	From Absolute to Relative: Rethinking Reward Shaping in Group-Based Reinforcement Learning	ICML 2026	apart (large models)
K063	Benchmarking the Limits of In-Context Reinforcement Learning for Ad-Hoc Teamwork	ICML 2026	classical sample
K064	Critique-Coder: Enhancing Coder Models by Critique Reinforcement Learning	ICLR 2026	apart (large models)
K065	Reinforcement Learning for Tool-Calling Agents in Fast Healthcare Interoperability Resources (FHIR)	ICML 2026	apart (large models)
K066	Bridging the performance-gap between target-free and target-based reinforcement learning	ICLR 2026	classical sample
K067	Reinforcement Learning Teachers of Test Time Scaling	NeurIPS 2025	apart (large models)
K068	WebSailor-V2: Bridging the Chasm to Proprietary Agents via Synthetic Data and Scalable Reinforcement Learning	ICLR 2026	apart (large models)
K069	Expanding the Capabilities of Reinforcement Learning via Text Feedback	ICML 2026	apart (large models)
K072	Continuous Q-Score Matching: Diffusion Guided Reinforcement Learning for Continuous-Time Control	NeurIPS 2025	classical sample
K073	SMAC: Score-Matched Actor-Critics for Robust Offline-to-Online Transfer	ICML 2026	classical sample
K075	CUDA-L1: Improving CUDA Optimization via Contrastive Reinforcement Learning	ICLR 2026	apart (large models)
K077	Reinforcement Unlearning via Group Relative Policy Optimization	ICLR 2026	apart (large models)
K078	Diffusion Guided Adversarial State Perturbations in Reinforcement Learning	NeurIPS 2025	classical sample
K079	From Static Constraints to Dynamic Adaptation: Sample-Level Constraint Relaxation for Offline-to-Online Reinforcement Learning	ICML 2026	classical sample
K081	Structured Reinforcement Learning for Combinatorial Decision-Making	NeurIPS 2025	classical sample
K082	Retrosynthesis Planning via Worst-path Policy Optimisation in Tree-structured MDPs	NeurIPS 2025	classical sample

continued on next page

Table C.1, continued

Id	Title	Venue	Family, role
K083	Use the Online Network If You Can: Towards Fast and Stable Reinforcement Learning	ICLR 2026	classical sample
K084	Approximating Shapley Explanations in Reinforcement Learning	NeurIPS 2025	classical sample
K085	Scalable Reinforcement Learning via Adaptive Batch Scaling	ICML 2026	classical sample
K086	Compute-Optimal Scaling for Value-Based Deep RL	NeurIPS 2025	classical sample
K089	Partially Equivariant Reinforcement Learning in Symmetry-Breaking Environments	ICLR 2026	classical sample
K090	Performative Policy Gradient: Optimality in Performative Reinforcement Learning	ICML 2026	apart (theory)
K093	Cross-Embodiment Offline Reinforcement Learning for Heterogeneous Robot Datasets	ICLR 2026	classical sample
K095	Robust In-Context Reinforcement Learning Under Reward Poisoning Attacks	ICML 2026	read after the 36th (classical)
K096	D²Evo: Dual Difficulty-Aware Self-Evolution for Data-Efficient Reinforcement Learning	ICML 2026	read after the 36th (large models)
K098	Towards Provable Emergence of In-Context Reinforcement Learning	NeurIPS 2025	read after the 36th (theory)
K099	On the Sample Complexity of Differentially Private Policy Optimization	NeurIPS 2025	read after the 36th (theory)
K102	Pretraining a Shared Q-Network for Data-Efficient Offline Reinforcement Learning	NeurIPS 2025	read after the 36th (classical)
K103	GUI Exploration Lab: Enhancing Screen Navigation in Agents via Multi-Turn Reinforcement Learning	NeurIPS 2025	read after the 36th (large models)
K105	The Surprising Difficulty of Search in Model-Based Reinforcement Learning	ICML 2026	read after the 36th (classical)
K106	Anchored Policy Optimization: Mitigating Exploration Collapse via Support-Constrained Rectification	ICML 2026	read after the 36th (large models)
K107	SOMBRL: Scalable and Optimistic Model-Based RL	NeurIPS 2025	read after the 36th (classical)
K109	On the Global Optimality of Policy Gradient Methods in General Utility Reinforcement Learning	NeurIPS 2025	read after the 36th (theory)
K110	Kevin: Multi-Turn RL for Generating CUDA Kernels	ICLR 2026	read after the 36th (large models)
K111	Polychromic Objectives for Reinforcement Learning	ICLR 2026	read after the 36th (classical)
K113	DAIL: Beyond Task Ambiguity for Language-Conditioned Reinforcement Learning	NeurIPS 2025	read after the 36th (classical)

C.2 The 2018 papers

Our reading of 2018 covers 50 RL papers from ICML 2018, NeurIPS 2018 and ICLR 2018, the three conferences Pineau named. The first 12 form the calibration of the protocol, 6 from ICML and 6 from NeurIPS (`study/calib_2018.json`). The other 38 come from enlargement A2, drawn in the order of a `random.Random(2018)` shuffle, with an allocation proportional to the size of each conference (deviation 12). The sample combines 21 papers from ICML 2018 (prefix `C1`), 20 from NeurIPS 2018 (`C2`) and 9 from ICLR 2018 (`C3`) (`study/calib_2018_v2.json`). The comparison for each criterion counts the papers where the criterion applies, so a denominator can fall below 50.

Table C.2. The 50 papers from 2018 read with both grids. The first 12 form the calibration of the first version; the other 38 (A2, deviation 12) are drawn in the order of a `random.Random(2018)` shuffle of the population of RL papers of ICML, NeurIPS and ICLR 2018 found by their title (131 papers, `study/rl_population_2018.json`), the three conferences Pineau names. 48 of the 50 papers have experiments (Q1); the other two leave the denominators. 2018 appendices read separately (ICML, NeurIPS). Source: `study/calib_2018_v2.json`.

Id	Title	Venue	Role
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), no experiment
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, no experiment
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

continued on next page

Table C.2, continued

Id	Title	Venue	Role
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

APPENDIX D

The software environments

This appendix gives the versions of the software that produced the results, the platform and the toolchain that builds the documents. The files in `envs/` are authoritative. Table D.2 extracts the key packages from them.

D.1 The platform

Every run and every published analysis comes from one Mac with an Apple M2 Max chip, under macOS 27.2, for the arm64 architecture, with Python 3.11.18. The simulations ran on the central processor, one thread per run. The GRPO runs ran on the chip's graphics processor (the MPS backend of PyTorch) or on the central processor, depending on the run (chapter 7). The original environments come from conda (`conda create -p sim/env python=3.11`). pip then installed each package. An ordinary `venv` environment receives the same packages.

D.2 The four environments

Each file is the `pip freeze` of an environment, taken on 15 November 2026, with the Python version and the platform at the top. The `*.lock` files give the same lists with the SHA-256 fingerprint of each package, for an installation with `pip install --require-hashes`. The four `*.txt` files repeat the lists that the protocols cite, `sim/requirements.lock` and its siblings, gathered in one folder.

Table D.1. The frozen environments of the project (`envs/README.md`).

File	Environment	What it runs
`train.txt`	`sim/env`	the training runs (`sim/run.py`) and the re-evaluations (`sim/dose.py`)
`analysis.txt`	`sim/env-an`	every analysis script, `docs/check_numbers.py`, continuous integration
`zoo.txt`	`sim/env-zoo`	the official RL Baselines3 Zoo pipeline of experiment E6
`llm.txt`	`sim/env-llm`	GRPO (`sim/llm/grpo.py`) and the figures of `docs/`
`test.txt`	on top of `analysis.txt`	the unit tests (`tests/`)

D.3 Installation pitfalls

- **Recreating an environment.** The paths matter, since the scripts and the launchers call `sim/env/bin/python` and its siblings:

Table D.2. Pinned versions of the key packages in the project's four Python environments, all on Python 3.11.18, macOS 27.2 on an Apple M2 Max chip (arm64). Each file `envs/<name>.txt` is the environment's `pip freeze`, taken on 15 November 2026; `envs/<name>.lock` is its twin locked by SHA-256 hashes (same versions). `train`: training (`sim/run.py`, `sim/dose.py`); `analysis`: every analysis script and the continuous integration; `zoo`: the official RL Baselines3 Zoo pipeline (E6); `llm`: GRPO (`sim/llm/grpo.py`). A dot: package absent from the environment. The unit tests add pytest 9.2.0 (`envs/test.txt`). The protocol announced Stable-Baselines3 2.6.0; every training run used 2.9.2 (deviation 14 (c) of `sim/SIM_PROTOCOL.md`). The revisions of the downloaded language models are not pinned. Source: `envs/*.txt`.

Package *environment*	`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	·
Pinned packages	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
```

The other three follow the same pattern. `sim/env-zoo` is a `venv` built on the Python of `sim/env`.

- **rliable 1.2.0** is distributed only as a source archive. pip builds it in a few seconds, since it is pure Python.
- **On Linux**, `analysis.txt` installs as is. Continuous integration does so on Ubuntu. The last digits of a bootstrap move with the processor: the byte-for-byte comparison is required on macOS arm64, the platform of the results. Linux is compared to within 10^{-12} in relative terms (section 8.6).
- **Training on Linux** works with `train.txt`, but the runs in the repository come from macOS arm64. A new run elsewhere gives other curves for the same random seed: MuJoCo, JAX and PyTorch round differently from one processor to another (section 10.5).

- **GRPO** downloads `Qwen/Qwen2.5-0.5B-Instruct` and `unsloth/Llama-3.2-1B-Instruct` from Hugging Face. Since deviation 15 (c) of the GRPO strand, `grpo.py` pins the exact revision of each model, the one used in our runs. `docs/figures/build.py` downloads Qwen's tokeniser.

D.4 The document toolchain

Outside the Python environments, building the PDFs and the figures uses the tools of table D.3. These tools draw and typeset without computing any number. `make check` and `make verify` need only `analysis.txt`.

Table D.3. The tools that build the documents.

Tool	Version	Used by
pandoc	3.6.1	`docs/build.sh`
md2star	3.13.5 (PyPI)	`docs/build.sh` (DOCX and a first 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 on PyPI)	`docs/figures/build.py`, `article/figures/build.py`
rsvg-convert (librsvg)	2.61.3	conversion of SVG files to PDF and PNG
git	2.50.0	`make verify`

This report is built by `make report`, which runs `report/build.sh`: the tables from the JSON files (`report/tools/tables.py`), the figures (`report/tools/figures.py`), then two passes of LuaLaTeX with biber, in French (`rapport.tex`) and in English (`report-en.tex`).

Bibliography

Agarwal, R. et al. (2021). 'Deep Reinforcement Learning at the Edge of the Statistical Precipice'. In: *Advances in Neural Information Processing Systems*. Vol. 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, pp. 375–386. DOI: 10.2307/3001775.

Baker, M. (2016). '1,500 scientists lift the lid on reproducibility'. In: *Nature* 533.7604, pp. 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 and 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. and D. L. Donoho (1995). 'WaveLab and Reproducible Research'. In: *Wavelets and Statistics*. Vol. 103. Lecture Notes in Statistics. Springer, pp. 55–81. DOI: 10.1007/978-1-4612-2544-7_5.

Chandak, N., S. Goel and 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*. Vol. 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, pp. 417–451. DOI: 10.2307/3001616.

Cohen, J. (1960). 'A Coefficient of Agreement for Nominal Scales'. In: *Educational and Psychological Measurement* 20.1, pp. 37–46. DOI: 10.1177/001316446002000104.

Colas, C., O. Sigaud and 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, pp. 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, pp. 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, pp. 1–26. DOI: 10.1214/aos/1176344552.

Eimer, T., M. Lindauer and R. Raileanu (2023). 'Hyperparameters in Reinforcement Learning and How To Tune Them'. In: *Proceedings of the 40th International Conference on Machine Learning*. Vol. 202. PMLR, pp. 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, pp. 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*. Vol. 35. arXiv: 2209.10485.

Grattafiori, A. et al. (2024). *The Llama 3 Herd of Models*. arXiv: 2407.21783.

Gundersen, O. E. and S. Kjensmo (2018). 'State of the Art: Reproducibility in Artificial Intelligence'. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1, pp. 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*. Vol. 80. PMLR, pp. 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*, pp. 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, pp. 65–70.

Irpan, A. (Feb. 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*. Vol. 119. PMLR, pp. 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*. Vol. 235. PMLR, pp. 22551–22569. arXiv: 2406.16241.

Kingma, D. P. and 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*. Vol. 100. PMLR, pp. 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, pp. 523–562. arXiv: 1709.06009.

Magnusson, I., N. A. Smith and J. Dodge (2023). *Reproducibility in NLP: What Have We Learned from the Checklist?* arXiv: 2306.09562.

Mann, H. B. and 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, pp. 50–60. DOI: 10.1214/aoms/1177730491.

Mantel, N. and W. Haenszel (1959). 'Statistical Aspects of the Analysis of Data From Retrospective Studies of Disease'. In: *Journal of the National Cancer Institute* 22.4, pp. 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, pp. 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*. Vol. 32. arXiv: 1912.01703.

Patterson, A. et al. (2024). 'Empirical Design in Reinforcement Learning'. In: *Journal of Machine Learning Research* 25.318, pp. 1–63. arXiv: 2304.01315.

Pineau, J. (May 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. (Dec. 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, pp. 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, pp. 1–8.

Schulman, J. et al. (2015). 'Trust Region Policy Optimization'. In: *Proceedings of the 32nd International Conference on Machine Learning*. Vol. 37. PMLR, pp. 1889–1897. arXiv: 1502.05477.

Schulman, J. et al. (2017). *Proximal Policy Optimization Algorithms*. arXiv: 1707.06347.

Shadish, W. R., T. D. Cook and 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. and M. B. Wilk (1965). ‘An Analysis of Variance Test for Normality (Complete Samples)’. In: *Biometrika* 52.3–4, pp. 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, pp. 484–489. DOI: 10.1038/nature16961.

Student (1908). ‘The Probable Error of a Mean’. In: *Biometrika* 6.1, pp. 1–25. DOI: 10.2307/2331554.

Sutton, R. S. and A. G. Barto (2018). *Reinforcement Learning: An Introduction*. 2nd ed. Cambridge, MA: MIT Press.

Tesauro, G. (1995). ‘Temporal Difference Learning and TD-Gammon’. In: *Communications of the ACM* 38.3, pp. 58–68. DOI: 10.1145/203330.203343.

Todorov, E., T. Erez and Y. Tassa (2012). ‘MuJoCo: A physics engine for model-based control’. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 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, pp. 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, pp. 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 and J. Pineau (2018a). *A Dissection of Overfitting and Generalization in Continuous Reinforcement Learning*. arXiv: 1806.07937.

Zhang, A., Y. Wu and J. Pineau (2018b). *Natural Environment Benchmarks for Reinforcement Learning*. arXiv: 1811.06032.