

Language Judges as Reward Sources for PPO in MuJoCo: Kev-4B versus an Autoregressive Judge of the Same Size

DC-Jev-RL project, Part I

October 2026

Abstract

Can a 4-billion-parameter language judge that reads a text summary of a robot’s behaviour replace the engineered reward of a MuJoCo task for PPO? We compare Kev-4B, a decision model that answers questions about a JSON state in one forward pass, with an autoregressive (AR) judge of the same size and family (Qwen3.5-4B) that reads the same text. We train PPO on seven reward sources across five tasks, with five seeds each (165 runs of 10^6 steps). With its healthy gate, Kev-4B’s reward reaches an interquartile mean normalised score of 1.00 [0.81, 1.09], against 1.05 for the engineered reward. A reward network distilled from Kev-4B’s labels on a tenth of the windows reaches 0.98, and the AR judge’s reward 0.35. The aggregate hides a failure: on Reacher every judge’s reward ends below a random policy. The AR judge’s reward teaches Hopper and Walker2d to stand still, although its scores rank windows almost as well as Kev-4B’s. On Hopper it pays an upright, slow window 0.87 of what it pays a fast one, against 0.48 for Kev-4B and 0.31 for the engineered reward. No judge’s reward let the cheetah flip onto its back, which the engineered reward did in 1 of five runs. Waiting for a judge shared by up to 24 runs made each run about 17 times slower. Code, logs and every number in this paper are in the repository.

1 Introduction

Reinforcement learning needs a reward, and for most tasks someone has to write it. The MuJoCo locomotion benchmarks come with engineered rewards: forward velocity plus a survival bonus minus a control cost, with weights that were tuned by hand. A language model that judges behaviour from a description of the task would remove that step. This idea has been pursued in several forms. Language models have written reward code [7, 16], and they have served directly as reward functions in text environments [5]. Vision–language models have scored images of a robot’s pose [10]. And a language model’s preferences over event captions have been distilled into an intrinsic reward [4].

This paper asks a narrow version of the question, with the controls needed to answer it. Suppose a 4-billion-parameter judge reads a short text summary of what the robot did in a window of 32 steps. Can its score replace the engineered reward of a standard benchmark when PPO is trained on it, and what does that cost? We compare two judges of the same size and family. One is Kev-4B [8], a decision model that answers questions about a JSON state in a single forward pass, with a pointer head and a calibration temperature. The other is an autoregressive (AR) judge, Qwen3.5-4B, whose next-token distribution over the answer options is read out the same way. Both judges see identical text. A control with the true reward at the judges’ temporal resolution (`env_window`) separates the effect of window-level feedback from the effect of the judge.

Contributions.

- A controlled comparison of seven reward sources for PPO on five MuJoCo tasks: the engineered reward, its window-averaged version, Kev-4B’s score with and without a “healthy” gate, a distilled reward model

- labelled by Kev-4B on 10% of the windows, and the AR judge with expected and with sampled readout. It uses 1M steps and 5 seeds per task (3 for the sampled AR judge), and statistics after Agarwal et al. [1] (§7).
- An offline agreement study that relates each judge signal to the environment’s reward on the same windows (§6).
 - An analysis of what the final policies do, which explains the main differences (§7.1). The AR judge’s reward teaches Hopper and Walker2d to stand still, because the judge pays an upright, slow window almost as much as a fast one. No judge’s reward lets the cheetah flip onto its back, which the engineered reward does. On Reacher, every judge’s reward makes the arm flail.
 - Measured costs: judged windows, judge time and wall-clock per run (§8).

2 Related work

Language models as reward designers and reward functions. Eureka [7] and Language to Rewards [16] use a language model to write or parameterise reward code. This keeps the policy’s reward cheap but moves the design effort into the model’s coding ability and into an outer search loop. Using the model as the reward itself, as Kwon et al. [5] do for negotiation and other text games, needs no code, but the model is then called during training. VLM-RM [10] scores rendered frames with CLIP for humanoid poses. Motif [4] distils a language model’s preferences between NetHack captions into a reward network, which our `kev_distill` condition resembles. Our judges read text summaries of telemetry instead of pixels. This isolates the judgment from perception, at the price of a summary that a person had to design.

LLM judges. Using language models to grade outputs [6, 18] has become standard in evaluation. As reward models they are known to be exploitable [2, 12]. Generative verifiers [17] read a judge’s yes/no probability, as our AR judge does. Kev-4B differs in that its answers are distributions from a dedicated head, calibrated with a temperature, and computed in one pass for several questions.

Evaluation of deep RL. With few seeds per task, point estimates of mean return are unreliable. We follow Agarwal et al. [1]: interquartile means over all runs with stratified-bootstrap confidence intervals, and the probability of improvement. We report per-task returns as well.

3 Judges

Kev-4B. Kev-4B [8] is a “System One” decision model: a LoRA adapter (rank 16) and a pointer head on Qwen3.5-4B-Base [9], a hybrid model whose layers mix Gated DeltaNet [15] and full attention. A request holds a *state* (a JSON object, rendered to text by `kev.api`) and one or more *questions*. A question of type `score` lists ordinal criteria; one of type `noul` asks for yes or no. Kev-4B does not generate text. Each question is a branch appended to the rendered state, and the pointer head reads the hidden states at a decision position and at the positions of the answer options. A softmax with the temperature stored in the checkpoint ($T = 2.406$) turns these into a distribution over the options. All questions of one request are answered in one forward pass of the shared state and their short branches. We use kev’s code for the schema, the rendering, the encoder and the head unchanged (revision 6cfce5c). Our wrapper only changes how rows are batched. On 300 questions its bf16 probabilities differ from kev’s reference serving path by at most 0.040 (argmax agreement 99.7%), and by at most 0.0009 in fp32 (`experiments/validation/kev_parity.json`).

Autoregressive (AR) judge. The baseline judge is Qwen3.5-4B [9] (revision 851bf6e), the instruction-tuned model of the same family and size, with thinking disabled. It gets the same rendered state and the same

task	task description	reference	healthy question
Hopper	A one-legged hopping robot (MuJoCo Hopper) must hop forward along the x-axis as fast as possible without falling over.	When the robot stands upright its torso is about 1.25 m above the ground. A strong policy for this robot hops forward at about 3 m/s.	Is the robot making genuine forward progress with a controlled, upright gait (not fallen, not falling, not standing still)?
Walker2d	A two-legged walking robot (MuJoCo Walker2d) must walk or run forward along the x-axis as fast as possible without falling over.	When the robot stands upright its torso is about 1.25 m above the ground. A strong policy for this robot runs forward at about 4 m/s.	Is the robot making genuine forward progress with a controlled, upright gait (not fallen, not falling, not standing still)?
HalfCheetah	A planar cheetah-like robot with two legs (MuJoCo HalfCheetah) must run forward along the x-axis as fast as possible.	The robot cannot fall, but it can flip onto its back, which is undesirable. A strong policy for this robot runs forward at about 10 m/s.	Is the robot running forward with an upright, coordinated gait (not flipped onto its back, not stuck in place)?
Swimmer	A three-segment swimming robot in a viscous fluid (MuJoCo Swimmer) must swim forward along the x-axis as fast as possible.	Motion in the fluid is slow: a strong policy for this robot swims forward at about 0.3 m/s.	Is the robot swimming forward with a coordinated undulating motion (not drifting backward, not stuck)?
Reacher	A two-joint robot arm (MuJoCo Reacher) must move its fingertip to a target point and keep it there, using as little control effort as possible.	The arm is about 0.21 m long. A fingertip within about 0.01 m of the target is on target; control effort ranges from 0 (none) to 1 (maximum).	Is the arm bringing its fingertip to the target or holding it there (not moving away, not flailing)?

Table 1: The task text both judges read, per task. Each request also holds the window’s summary (the quantities of §4). The **progress** question (“How well is the robot accomplishing its task in this segment?”) has five task-specific levels, from *failing* to *excellent*; all are in `experiments/part1_control/prompts/judge_prompts.json`.

question text in a chat prompt, followed by the numbered options and the instruction to answer with the option number only (or with yes or no). We read the next-token distribution once, restrict it to the tokens of the valid answers and renormalise. The mass on valid answers before renormalisation is logged (§6). It is at least 1.00 for the yes/no question, but only 0.75 to 0.96 for the five-level **progress** question, so the renormalisation is not a formality there. The AR judge costs one forward pass per question, like Kev-4B. It needs one row per question instead of one shared state, and it is given no calibration.

Questions. For every task the state is {task description, reference, segment summary} (Table 1); the questions are fixed in `experiments/part1_control/prompts/judge_prompts.json` and were frozen before the grid was run. **progress** is a five-level score question (*failing*, *poor*, *fair*, *good*, *excellent*, each with a one-line description), read out as the expected level divided by four, in $[0, 1]$. **healthy** is a noul question (“Is the robot making genuine forward progress with a controlled, upright gait. . . ?”) read out as $p(\text{yes})$.

4 Method

What the judges see. A judge never sees the environment’s reward. It sees a short summary of a *window* of consecutive steps of one episode (32 steps; 25 for Reacher), computed from physical quantities read from the simulator: the forward velocity of the torso (mean, minimum and maximum over the window), the torso height and tilt for Hopper and Walker2d, the torso pitch for HalfCheetah, the fingertip–target distance for Reacher (at the start and end of the window, and its mean), the mean squared action (“control effort”), and, for Hopper and Walker2d, whether the episode terminated in the window (`fell_over`). The reference field gives two facts per task, for example “A strong policy for this robot hops forward at about 3 m/s”. Windows

are cut at episode ends and at the end of each PPO rollout, so a window can be shorter than 32 steps.

This is a deliberately favourable setting for a language judge. The summary is built from privileged simulator state, it names the quantities that matter, and the reference states a target speed. The experiment tests whether a 4B judge can turn a task description and a numeric behaviour summary into a reward that PPO can optimise. It does not test perception from pixels or from raw observations.

Reward conditions. Every step of a window receives the window’s score. The conditions are:

- **env**: the environment’s own per-step reward, the engineered reference.
- **env_window**: the environment’s reward averaged over each window and given to every step of it. This control has the judges’ temporal resolution and the true reward’s information. It separates the effect of window-level feedback from the effect of the judge.
- **kev_score**: Kev-4B’s progress score.
- **kev_gated**: Kev-4B’s progress score multiplied by its $p(\text{healthy})$, a multiplicative gate that should suppress windows that look fast but are not genuine progress.
- **kev_distill**: a small reward network (two hidden layers of 64 units) trained online, with a confidence-weighted squared loss, on **kev_gated** labels of a random 10% of the windows of each rollout (all windows of the first rollout). It predicts the remaining windows from numeric features of the same quantities the summary reports. The weight of a label is the confidence of Kev-4B’s progress distribution, clipped to $[0.05, 1]$. This condition calls the judge about ten times less often.
- **ar_gated**: the AR judge’s expected progress level times its $p(\text{healthy})$, the AR counterpart of **kev_gated**.
- **ar_sampled**: as **ar_gated**, but with a progress level sampled from the AR judge’s distribution instead of its expectation. This is the reward an AR judge gives when it answers in text at temperature one (3 seeds).

Every condition goes through the same reward normalisation (gymnasium’s `NormalizeReward`, reproduced step by step, and clipping to $[-10, 10]$). The judges are frozen, so no condition adapts the judge to the policy.

PPO. We use a single-environment port of CleanRL’s `ppo_continuous_action.py` [3, 11] with its default hyperparameters: 10^6 environment steps, rollouts of 2048 steps, 10 epochs of 32 minibatches, learning rate $3 \cdot 10^{-4}$ annealed linearly to zero, $\gamma = 0.99$, $\lambda_{\text{GAE}} = 0.95$, clip coefficient 0.2, value coefficient 0.5, no entropy bonus, gradient-norm clipping at 0.5, and running observation normalisation. The hyperparameters are the same in every condition and were not tuned for any of them.

Evaluation. A policy is evaluated with its deterministic action (the mean of the Gaussian), frozen observation statistics and the environment’s own undiscounted return. This is 10 episodes every 10^5 steps and 20 episodes at the end, from fixed evaluation seeds ($10,000 + k$ and $20,000 + k$ for episode k). The judges never see these returns. We normalise a run’s final return R per task as $(R - R_{\text{rand}})/(R_{\text{env}} - R_{\text{rand}})$, where R_{rand} is the mean return of a uniform random policy and R_{env} is the mean final return of the **env** condition. A score of 1 means “as good as PPO on the engineered reward”. Aggregates over the 5 tasks $\times$ 5 seeds follow Agarwal et al. [1]: the interquartile mean (IQM) with 95% stratified-bootstrap confidence intervals (10 000 resamples), and the probability of improvement of one condition over another.

Judge agreement. Every run logs a random 2% of its windows with the summary and the environment reward over the window. The windows logged by the **env** runs (all seeds, all of training) cover the failing, mediocre and good behaviour of the reference learner. Offline, both judges score up to 4000 of them per task, through the same server and prompts as in training. We report each signal’s Spearman correlation with the window’s per-step environment reward. We also report it with the physical quantity the task is about: the mean forward velocity, or for Reacher the reduction of the fingertip–target distance over the window. For

task	windows	Kev-4B			AR judge			
		progress	healthy	gated	progress	sampled	healthy	gated
Hopper	3233	0.73	0.56	0.70	0.66	0.22	0.50	0.65
Walker2d	3270	0.73	0.58	0.71	0.69	0.22	0.58	0.68
HalfCheetah	3167	0.93	0.85	0.88	0.64	0.02	0.76	0.79
Swimmer	3167	0.73	0.83	0.86	0.67	0.26	0.78	0.76
Reacher	3997	0.37	0.11	0.25	0.09	0.04	-0.05	0.02

Table 2: Spearman correlation of each judge signal with the per-step environment reward of the window, on windows logged by the `env` runs (all seeds, all of training). *progress*: the expected level divided by four; *healthy*: $p(\text{yes})$; *gated*: their product; *sampled*: a level drawn from the AR judge’s distribution, as an AR judge answering in text at temperature one would give it.

Hopper and Walker2d we also report the AUROC of $p(\text{healthy})$ for windows that did not end in a fall. During training, each judge-reward run also logs, per rollout, the correlation between the window scores it trained on and the environment reward of the same windows.

5 Experimental setup

The tasks are Hopper, Walker2d, HalfCheetah, Swimmer and Reacher, all MuJoCo v5 through Gymnasium 1.3 [13, 14]. There are five seeds per task and condition (three for `ar_sampled`). Seed s fixes the policy initialisation, the environment’s seed and the run’s own random choices, the windows sampled for logging and for distillation included. Runs that query a judge are still not bitwise reproducible. The judge’s floating-point results depend slightly on how the server batched the request with those of other runs, and the level drawn for `ar_sampled` comes from the server’s random generator, which all runs share. All runs used one Colab virtual machine with one NVIDIA RTX PRO 6000 Blackwell Server Edition GPU (96 GB) and 48 vCPUs. PPO runs on the CPU, one process per run. The judges run in one GPU server process that holds both Kev-4B (bf16, LoRA merged) and the AR judge. The server merges concurrent requests from all runs into batches (at most 4096 windows, 15 ms gathering). Up to 24 runs trained at the same time. A run’s wall-clock time therefore depends on how many other runs shared the judge, and we report it as measured. Software versions are pinned in the repository (`scripts/setup_colab.sh`, with the full package lists of both environments). A run interrupted by a runtime reset resumes from its last periodic checkpoint. The simulator state is not checkpointed, so the episode in progress is dropped and a new one starts. The log records every resume, and the number of resumed runs is reported with the results.

6 Do the judges’ scores track the reward?

Both judges scored the windows logged by the `env` runs: 3233 for Hopper, 3270 for Walker2d, 3167 for HalfCheetah, 3167 for Swimmer and 3997 for Reacher, from all five runs and all of training. Table 2 gives each signal’s Spearman correlation with the window’s per-step environment reward. Intervals are 95% intervals from a two-stage bootstrap that resamples the runs and then the windows of each run, so the windows of one run are not counted as independent.

Kev-4B tracks the reward more closely than the AR judge. On every task, Kev-4B’s `progress` correlates more strongly with the environment’s reward than the AR judge’s expected `progress`. The difference is smallest on Walker2d (+0.04 [0.03, 0.06]) and largest on HalfCheetah (+0.29 [0.18, 0.41]) and Reacher (+0.28

[0.25, 0.31]). No interval includes zero. The gated signals keep the order on every task (+0.02 on Walker2d to +0.23 on Reacher).

Reacher is the exception that qualifies this. Reacher’s per-step reward is mostly the distance between the fingertip and the target, a state. The progress levels describe approaching the target and holding it. Against the reduction of the distance over the window, the quantity the question is mostly about, the two judges are close, with the AR judge slightly ahead (Kev-4B 0.61, AR 0.64; difference -0.03 [$-0.06, -0.00$]). Kev-4B’s higher correlation with Reacher’s reward is therefore not a better reading of the approach the question asks about. On the four locomotion tasks the reward and the forward velocity rank the windows almost identically, and the two correlations nearly coincide.

Sampling the AR judge’s answer discards most of the signal. A level drawn from the AR judge’s distribution correlates only 0.02 (HalfCheetah) to 0.26 (Swimmer) with the reward, against 0.09 to 0.69 for the expected level of the same distribution. At the scale of one window, the information is in the distribution, and a single draw from it is mostly noise.

Falls are seen; the gate is mixed. $p(\text{healthy})$ separates the windows without a fall from those with one almost perfectly: AUROC 1.00 and 0.99 for Kev-4B on Hopper and Walker2d, 1.00 and 1.00 for the AR judge. The summary states whether the robot fell, so this checks reading rather than judgment. Multiplying Kev-4B’s progress by its $p(\text{healthy})$ lowers the agreement with the reward on four tasks and raises it only on Swimmer (0.86 against 0.73). For the AR judge the gate also helps on HalfCheetah (0.79 against 0.64).

The AR judge puts mass outside the answer options. Before renormalisation, the AR judge’s next-token distribution puts only 0.75 to 0.96 of its mass on the five progress options, depending on the task, and at least 1.00 on yes or no. Kev-4B has no such loss, since its head scores only the options.

What this shows and what it does not. The windows come from policies that optimise the true reward. They show how well each judge ranks the behaviour such policies produce. They do not show how a judge ranks the behaviour of a policy that optimises the judge itself; the training runs measure that (§7).

7 Training on the judges’ rewards

All 165 runs finished: seven conditions on five tasks, five seeds each (three for `ar_sampled`). A reset of the virtual machine interrupted 24 runs (of `kev_score`, `kev_gated`, `kev_distill`, `ar_gated`). They resumed from their checkpoint at step 301,056, and each lost the episode in progress, at most 703 steps. Table 3 gives the final returns per task, Table 4 the normalised aggregate, and Figure 1 the learning curves.

In aggregate, Kev-4B’s gated and distilled rewards come close to the engineered reward, and the AR judge’s do not. The IQM of the normalised score is 1.00 [0.81, 1.09] for `kev_gated` and 0.98 [0.82, 1.10] for `kev_distill`, against 1.05 [0.91, 1.16] for `env`. The intervals overlap widely. A run with either reward beats a run with the engineered reward on the same task with probability 0.44 (`kev_gated`) and 0.44 (`kev_distill`). Without the gate, Kev-4B’s score does worse (0.78 [0.60, 0.91], probability 0.28). The AR judge’s rewards reach about a third of the engineered reward’s normalised score: 0.35 [0.28, 0.46] with the expected readout and 0.31 [0.19, 0.45] with a sampled level. A `kev_gated` run beats an `ar_gated` run on the same task with probability 0.78. Per task this probability is 1.00 on Hopper, 1.00 on HalfCheetah and 0.88 on Walker2d. On Swimmer (0.52) and Reacher (0.48) it is close to even.

task	env	env_window	kev_score	kev_gated	kev_distill	ar_gated	ar_sampled
Hopper	2820 $\pm$ 969	3419 $\pm$ 450	3061 $\pm$ 314	3169 $\pm$ 174	2833 $\pm$ 909	1165 $\pm$ 309	1127 $\pm$ 293
Walker2d	3150 $\pm$ 1276	4068 $\pm$ 1909	2593 $\pm$ 509	2536 $\pm$ 712	2943 $\pm$ 562	1321 $\pm$ 189	1330 $\pm$ 165
HalfCheetah	3441 $\pm$ 1152	2376 $\pm$ 1349	2845 $\pm$ 638	4241 $\pm$ 242	4117 $\pm$ 460	1069 $\pm$ 1046	1050 $\pm$ 824
Swimmer	81 $\pm$ 22	278 $\pm$ 104	99 $\pm$ 148	141 $\pm$ 124	91 $\pm$ 41	141 $\pm$ 120	118 $\pm$ 166
Reacher	-6.4 $\pm$ 1.6	-28.0 $\pm$ 9.4	-48.9 $\pm$ 21.2	-75.8 $\pm$ 10.9	-88.4 $\pm$ 21.9	-75.2 $\pm$ 21.3	-68.8 $\pm$ 21.3

Table 3: Final return per task: mean $\pm$ standard deviation over seeds of the mean return over 20 evaluation episodes of the deterministic policy after 10^6 steps. A uniform random policy scores 13 (Hopper), -0.40 (Walker2d), -280 (HalfCheetah), -0.13 (Swimmer) and -42 (Reacher).

condition	runs	IQM [95% CI]	mean	median	$P(> \text{env})$	$P(> \text{env_window})$
env	25	1.05 [0.91, 1.16]	1.00	1.03	0.50	0.46
env_window	25	1.10 [0.89, 1.26]	1.41	1.21	0.54	0.50
kev_score	25	0.78 [0.60, 0.91]	0.76	0.82	0.28	0.27
kev_gated	25	1.00 [0.81, 1.09]	0.79	1.05	0.44	0.31
kev_distill	25	0.98 [0.82, 1.10]	0.59	1.02	0.44	0.27
ar_gated	25	0.35 [0.28, 0.46]	0.40	0.36	0.18	0.10
ar_sampled	15	0.31 [0.19, 0.45]	0.37	0.36	0.12	0.11

Table 4: Normalised score $(R - R_{\text{rand}})/(R_{\text{env}} - R_{\text{rand}})$ over all tasks and seeds: interquartile mean (IQM) with a 95% stratified-bootstrap interval, mean, median, and the probability that a run of the condition beats a run of env or env_window on the same task, averaged over tasks.

The aggregate hides Reacher. The IQM leaves out the lowest and the highest quarter of the runs. The means are lower: 0.79 for kev_gated and 0.59 for kev_distill. On Reacher every judge’s reward ends below the random policy, so these runs have negative normalised scores (Table 3). We therefore go through the tasks one by one.

Task by task. On Hopper, Kev-4B’s rewards end close to the engineered reward (3169 for kev_gated against 2820; probability of improvement 0.44). On Walker2d they end lower (2536 for kev_gated and 2943 for kev_distill, against 3150; probabilities 0.32 and 0.36). On HalfCheetah, kev_gated (4241) and kev_distill (4117) end above the engineered reward (3441; probability 0.72 for both), for a reason we come to in §7.1. On Swimmer, PPO on the engineered reward barely moves the swimmer (81). Every other reward does better on average, the window-averaged true reward most (278), and the judges’ rewards vary widely between seeds. On Reacher, nothing but the engineered reward learns the task: -6.4 with it, -28.0 with its window average, and -48.9 to -88.4 with the judges’ rewards. The learning curves show the judges’ Reacher policies getting worse as training proceeds (Figure 1).

Window-level feedback costs little on locomotion and a lot on Reacher. The true reward averaged over the judges’ windows (env_window) does at least as well as the per-step reward in aggregate: IQM 1.10 [0.89, 1.26], probability of improvement 0.54. It is ahead on Hopper (0.80), Walker2d (0.80) and Swimmer (0.92). It is behind on HalfCheetah (0.20), where 3 of its five runs end on their back (§7.1), and on Reacher (0.00). A Reacher episode has only two windows of 25 steps. The averaged reward no longer tells the policy which of its actions brought the fingertip closer, and its arms end no closer to the target than a random policy’s (Table 5). The judges’ temporal resolution therefore explains part of their failure on Reacher. It explains nothing of the AR judge’s failure on Hopper and Walker2d, where the window-averaged reward does well.

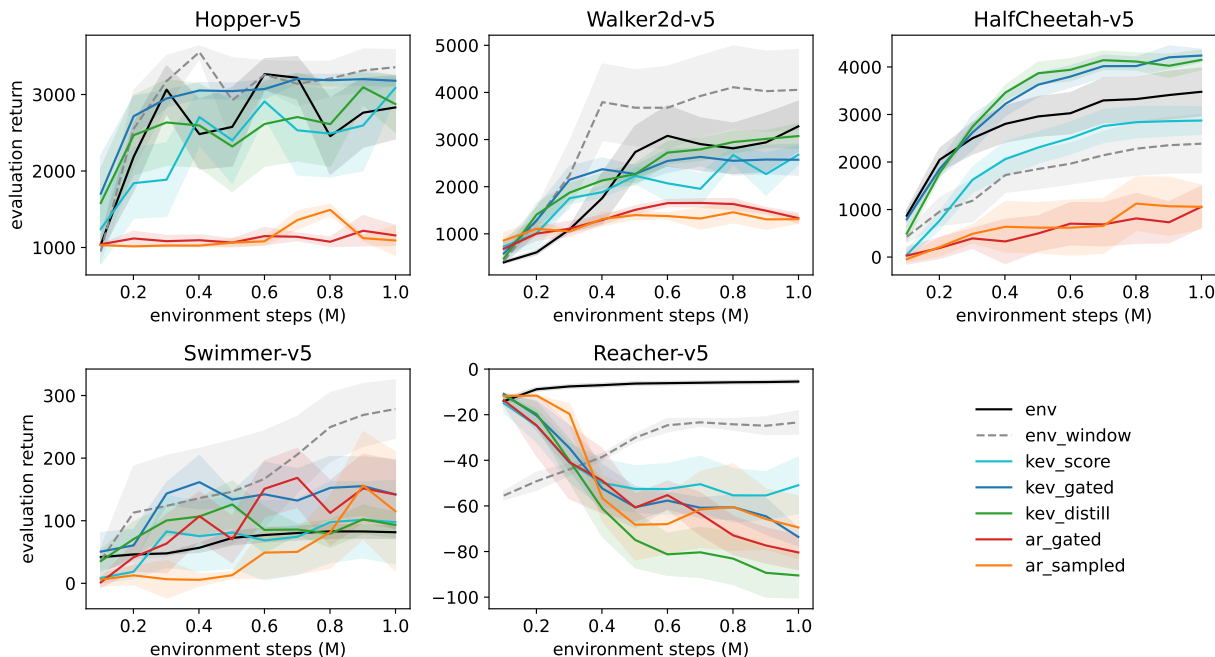


Figure 1: Evaluation return during training: 10 episodes every 10^5 steps, mean over seeds, shaded $\pm$ one standard error.

Distillation keeps Kev-4B’s reward at a tenth of the judge calls. `kev_distill` asks the judge about 3.5k windows per run instead of 33.9k, and ends level with `kev_gated`: IQM 0.98 against 1.00, probability of improvement 0.45. Its weakest task is Reacher (−88.4).

During training, both judges’ scores track the reward to a similar degree. For every rollout, each run logged the correlation between the window scores it trained on and the environment’s reward of the same windows. Averaged over rollouts and runs, it is 0.56 for `kev_gated` and 0.52 for `ar_gated`, and only 0.17 for `ar_sampled`, whose single draws are mostly noise (§6). The two gated rewards correlate with the reward about equally, yet they train very different policies, as the next section shows.

7.1 What the policies learned

A return does not say what a policy does. We ran every final policy again on its 20 evaluation episodes, and recorded what the robot did (Table 5). The re-runs reproduce the logged returns exactly (largest difference 0).

The AR judge’s reward teaches the hopper and the walker to stand. The final `ar_gated` policies stand still for 49% of the steps on Hopper and 40% on Walker2d. For `kev_gated` the figures are 1% and 10%, and for `env` 0% and 1%. Their mean forward velocity is 0.17 and 0.37 m/s, against 2.21 and 1.57 m/s for `kev_gated` and 2.59 and 3.02 m/s for `env`. Their Hopper return (1165) is close to the 1000 that the survival bonus alone pays for standing through an episode of 1000 steps. Sampling the level does not remove it: `ar_sampled` stands still for 26% and 17% of the steps.

Table 6 shows why. It compares what each signal pays for an upright window slower than 0.5 m/s with what it pays for a window faster than 3 m/s. The AR judge’s gated score pays the slow window 0.87 of the fast one on Hopper (0.85 against 0.97) and 0.87 on Walker2d. The engineered reward pays 0.31 and 0.27, and Kev-4B’s gated score 0.48 and 0.47. Every step of an episode receives the score of its window, and a fall

	Hopper	Walker2d	Swimmer	HalfCheetah	Reacher	
condition	still	still	still	runs flipped	effort	distance (m)
env	0%	1%	62%	1 of 5	0.01	0.101
env_window	0%	0%	14%	3 of 5	0.19	0.180
kev_score	0%	4%	11%	0 of 5	0.36	0.256
kev_gated	1%	10%	15%	0 of 5	0.63	0.263
kev_distill	1%	1%	11%	0 of 5	0.76	0.250
ar_gated	49%	40%	18%	0 of 5	0.63	0.253
ar_sampled	26%	17%	50%	0 of 3	0.56	0.252
random policy	31%	4%	26%	–	0.33	0.180

Table 5: Behaviour of the final policies, averaged over episodes and seeds (20 evaluation episodes per run). *still*: fraction of steps with $|v_x| < 0.1$ m/s. *runs flipped*: runs whose cheetah spends more than half of the steps on its back ($\cos(\text{pitch}) < 0$). *effort*: mean squared action, in $[0, 1]$. *distance*: mean fingertip–target distance over the episode. The random policy samples its actions uniformly.

task	windows (slow / fast)	env. reward	Kev-4B		AR judge	
			progress	gated	progress	gated
Hopper	42 / 430	0.31	0.59	0.48	0.89	0.87
Walker2d	386 / 1238	0.27	0.62	0.47	0.90	0.87
HalfCheetah	309 / 1368	−0.06	0.51	0.15	0.85	0.65

Table 6: What each signal pays for standing relative to running, on the windows of §6 (logged by the env runs). The table gives the mean over the windows without a fall that are slower than 0.5 m/s, divided by the mean over those faster than 3 m/s. *env. reward*: the per-step environment reward.

ends the episode. Under the AR judge’s reward, standing upright therefore earns almost as much per step as hopping, without the risk of a fall. PPO finds this early: the *ar_gated* curves on Hopper are flat from the first evaluation (Figure 1). Yet the AR judge’s scores rank the windows almost as well as Kev-4B’s. Their Spearman correlation with the reward is 0.65 against 0.70 on Hopper and 0.68 against 0.71 on Walker2d (Table 2). What differs is the margin: the AR judge rates almost every upright window near the top of its scale. Rank agreement with the reward did not predict which reward PPO could use.

The judges’ rewards keep the cheetah upright. On HalfCheetah, 1 of the five *env* runs and 3 of the five *env_window* runs end with the cheetah on its back. This is a local optimum that the engineered reward does not penalise. These runs return at most 1599, and the upright runs of the same two conditions at least 3049. None of the 23 runs on a judge’s reward flips. The task text both judges read says that flipping is undesirable, and the lowest *progress* level names it. Without its flipped run, *env* averages 3902 on HalfCheetah, against 4241 for *kev_gated*, so the flip accounts for much of Kev-4B’s advantage there. Here the judge’s reward carries a preference that the engineered reward leaves out, and the policy follows it.

On Reacher, the judges’ rewards make the arm flail. The arms trained on a judge’s reward keep the fingertip 0.250 to 0.263 m from the target on average. That is farther than a random policy’s (0.180 m) and the engineered reward’s (0.101 m). They also spend more effort than the random policy: 0.36 to 0.76, against 0.33, and 0.01 for *env*. Reacher’s reward charges the squared action at every step, which is why these policies end below the random policy. Both judges’ scores follow the change of the distance within a window (Spearman 0.61 for Kev-4B’s and 0.64 for the AR judge’s *progress*, §6). They follow the distance itself,

condition	judged windows	judge wait (h)	wall-clock (h)	corr. (score, env reward)
env	0	0.0	0.1	–
env_window	0	0.0	0.1	–
kev_score	33,914	2.0	2.1	0.57
kev_gated	33,942	1.9	2.1	0.56
kev_distill	3,477	1.9	2.0	0.54
ar_gated	33,861	1.9	2.0	0.52
ar_sampled	33,869	1.9	2.0	0.17

Table 7: Cost per run, averaged over tasks and seeds: windows sent to the judge, time spent waiting for its answers, wall-clock time of the run, and the mean correlation, per rollout, between the window scores the run trained on and the environment’s reward of the same windows. One judge server on one GPU served up to 24 concurrent runs, so the waits include the time it spent on the other runs’ requests.

which Reacher’s reward charges at every step, much less (0.37 and 0.09 with the reward). With two windows per episode, neither signal led PPO to the target.

8 Cost

Table 7 gives what the judges cost per run. A run that asks the judge about every window sends it about 33.9k windows over 10^6 steps, and `kev_distill` about 3.5k. A run without a judge took 0.1 h. A run with one took about 2.1 h, about 17 times as long, and spent nearly all of it (1.9 h) waiting for the judge. These times describe our setup, in which one judge server on one GPU answered up to 24 runs at once. A run’s wait includes the time the server spent on the other runs’ requests. That is why `kev_distill`, with a tenth of the windows, waited about as long (1.9 h): its requests queued behind those of the other runs. We did not measure a judge serving a single run, which is where the distilled reward’s saving would show.

9 Discussion

What the experiment answers. The question was whether a 4B judge that reads a text summary of the robot’s behaviour can give PPO a reward to learn from. With Kev-4B’s gated or distilled reward, PPO learned to move forward on all four locomotion tasks. On Hopper the final returns are close to those of the engineered reward, and on HalfCheetah they are higher. On Swimmer they are higher on average but spread widely, and on Walker2d they are lower. On Reacher every judge’s reward failed. The autoregressive judge of the same size and family read the same text and answered the same questions, but its reward taught the hopper and the walker to stand.

Correlation is not enough. On Hopper and Walker2d the two gated judges rank windows by the reward almost equally well, offline (§6) and during training (Table 7). Yet one reward trained hopping and walking, and the other trained standing. The outcome was decided by how much more the judge pays for the behaviour the task wants than for safe but useless behaviour (Table 6). An offline test of a judge as a reward should report this margin as well as the correlation. Here a fall ends the episode and every step earns a positive reward, so a small margin favours survival over progress. A judge whose scores crowd at the top of its scale makes the margin small.

A judge’s reward follows the task text. It is not a noisy copy of the engineered reward. The HalfCheetah text calls flipping undesirable, and no judge’s reward let the cheetah flip, while the engineered reward did in 1

of five runs. The Reacher text asks for little control effort, and the summary reports the effort, yet the arms trained on the judges’ rewards flail. Both judges follow how the distance changes within a window more than how large it is, and with two windows per episode PPO did not find the target. A judge can add a preference that the engineered reward lacks, and it can miss a cost that the engineered reward charges.

Distillation. A reward network trained on Kev-4B’s labels for a tenth of the windows did as well as asking Kev-4B about every window. In our setup this saving did not shorten the runs, because the judge server was shared. We expect it to matter when a judge serves a single run, but we did not measure that.

Limitations. The judges read summaries that we designed from privileged simulator state. The summaries name the quantities that matter and state a target speed. This favours the judges, and the experiment says nothing about judging from pixels or raw observations. We used one PPO configuration (CleanRL’s defaults) for every condition, 10^6 steps, and five seeds (three for `ar_sampled`). The intervals are wide, and per-task probabilities of improvement between about 0.3 and 0.7 should not be read as differences. The AR judge had one prompt format and one readout (the expectation over the option tokens, renormalised), with no calibration and no reasoning. A different prompt, a calibrated or rescaled readout, or reasoning before the answer could change its margin. We did not try them, and we did not tune either judge’s prompts. The comparison involves two judges of one family and size. It does not show that decision models in general make better rewards than generative judges. The costs were measured with one judge server shared by up to 24 runs. Finally, 24 runs were resumed after a reset of the machine, and each lost one episode.

Reproducibility

Code, prompts, configuration, raw logs (every training episode, update, evaluation and a 2% sample of judged windows) and final policies of all runs are in the repository (`experiments/part1_control/`). The tables and every number in the text are generated from them by `dcjev/analysis/part1.py`, `dcjev/analysis/judge_agreement.py`, `dcjev/analysis/behaviour.py` and `dcjev/analysis/paper_numbers.py`.

References

- [1] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems*, volume 34, 2021.
- [2] Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning*, 2023.
- [3] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G. M. Araújo. CleanRL: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.
- [4] Martin Klissarov, Pierluca D’Oro, Shagun Sodhani, Roberta Raileanu, Pierre-Luc Bacon, Pascal Vincent, Amy Zhang, and Mikael Henaff. Motif: Intrinsic motivation from artificial intelligence feedback. In *International Conference on Learning Representations*, 2024.
- [5] Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. Reward design with language models. In *International Conference on Learning Representations*, 2023.

- [6] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-Eval: NLG evaluation using GPT-4 with better human alignment. In *Conference on Empirical Methods in Natural Language Processing*, 2023.
- [7] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *International Conference on Learning Representations*, 2024.
- [8] Jared Palmer. Kev-4B: a system one decision model. <https://huggingface.co/jaredpalmer/kev-4b>, revision 6cfce5c2fa4b4bd64026336ab649c5ca78857d52, 2026.
- [9] Qwen Team. Qwen3.5-4B and Qwen3.5-4B-Base. <https://huggingface.co/Qwen/Qwen3.5-4B-Base>, 2026.
- [10] Juan Rocamonde, Victoriano Montesinos, Elvis Nava, Ethan Perez, and David Lindner. Vision-language models are zero-shot reward models for reinforcement learning. In *International Conference on Learning Representations*, 2024.
- [11] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [12] Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking. In *Advances in Neural Information Processing Systems*, 2022.
- [13] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.
- [14] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.
- [15] Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *International Conference on Learning Representations*, 2025.
- [16] Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, Brian Ichter, Ted Xiao, Peng Xu, Andy Zeng, Tingnan Zhang, Nicolas Heess, Dorsa Sadigh, Jie Tan, Yuval Tassa, and Fei Xia. Language to rewards for robotic skill synthesis. In *Conference on Robot Learning*, 2023.
- [17] Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. *arXiv preprint arXiv:2408.15240*, 2024.
- [18] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2023.