# CHEAP JUDGES FOR DENSE REWARDS IN REINFORCEMENT LEARNING

**Jai Pal**

4 October 2026

**ABSTRACT**

This pilot tested Clef-Flash as a source of rewards after each tool call in one small, synthetic workflow. A table of action values controlled the policy; no language model weights changed. Four reward conditions used five paired runs and 6,000 tool calls per run. Clef progress plus an exact terminal reward raised mean curve success from 83.41% to 95.88%. Both conditions reached 100% final success. A handwritten exact progress score matched Clef at every evaluation checkpoint.

Clef terminal rewards alone produced 35.04% mean curve success and 80% final success. One policy omitted required verification. The results support a narrow role for intermediate feedback with an independent final checker. They do not establish an advantage over exact progress, a larger judge, or language-model reinforcement learning.

## 1 Question and scope

A judge is a model that assigns scores to task states. A terminal reward arrives at the end of an attempt. A dense reward supplies feedback after each action. This pilot asked whether a small judge could supply useful dense rewards while an exact checker retained control of final success. The separate terminal-judge condition tested the consequence of trust in the judge alone.

It used the real `Cloudflare/clef-flash` weights on one NVIDIA A100 graphics processing unit (GPU) with 40 GB of memory. The judge stayed fixed throughout the experiment. The trained policy was a small action-value table, not a language model. This report uses the original saved results; it adds no experiment runs.

## 2 Method and controls

### 2.1 Task and policy

The task required four actions: fetch the values $[1,2,3]$, calculate their sum, verify the result, and submit it. Each attempt allowed eight tool calls. The action set also included `wait` and `reset`. An action out of order consumed a call without progress. Submission ended the attempt, and only a verified submission counted as success.

The policy used tabular Q-learning, which updates a stored value for each state and action. Its state identified the workflow stage and available call budget. The update rate was $\alpha = 0.2$, the discount was $\gamma = 0.95$, and random action choice had probability $\varepsilon = 0.25$. A seed sets a run's random choices. Each of the four reward conditions used seeds 0–4 and exactly 6,000 training tool calls per seed. The same seed values paired the conditions.

Evaluation ran before training and after each 500 calls. Each checkpoint used 100 attempts with highest-value actions and random tie resolution. The exact checker measured success. Evaluation used no judge calls and did not update the policy.

### 2.2 Reward conditions

The two terminal conditions used either exact success or the judge's terminal score. The two dense conditions retained exact success and added a change in progress:

$$r_t = r_t^{\text{exact}} + \beta\,[\gamma\Phi(s_{t+1}) - \Phi(s_t)], \qquad \beta = 0.5, \quad \Phi(s_{\text{end}}) = 0. \tag{1}$$

Here, $\Phi$ is a potential: a numerical score for progress at a state. The exact potential equaled the completed fraction of three stages: fetch, calculate, and verify. The Clef potential came from the model's progress score. The judge saw task rules and tool evidence, without the exact success or progress labels. The code set terminal potential to zero for submission, call exhaustion, and budget truncation. This boundary rule leaves exact terminal success as the objective.

## 3 Results

### 3.1 Success during training

Figure 1 uses all 260 saved checkpoint rows. Let $p_{s,k}$ denote exact success for seed $s$ at checkpoint $c_k$. The curve measure is the trapezoidal checkpoint area divided by the tool-call budget:

$$A_s = \frac{1}{6000} \sum_{k=1}^{12} (c_k - c_{k-1}) \frac{p_{s,k-1} + p_{s,k}}{2}, \qquad c_k = 500k. \tag{2}$$

Table 1 reports the mean of $A_s$ across five seeds. This measure summarizes checkpoint success during training; it is not the fraction of successful training attempts.

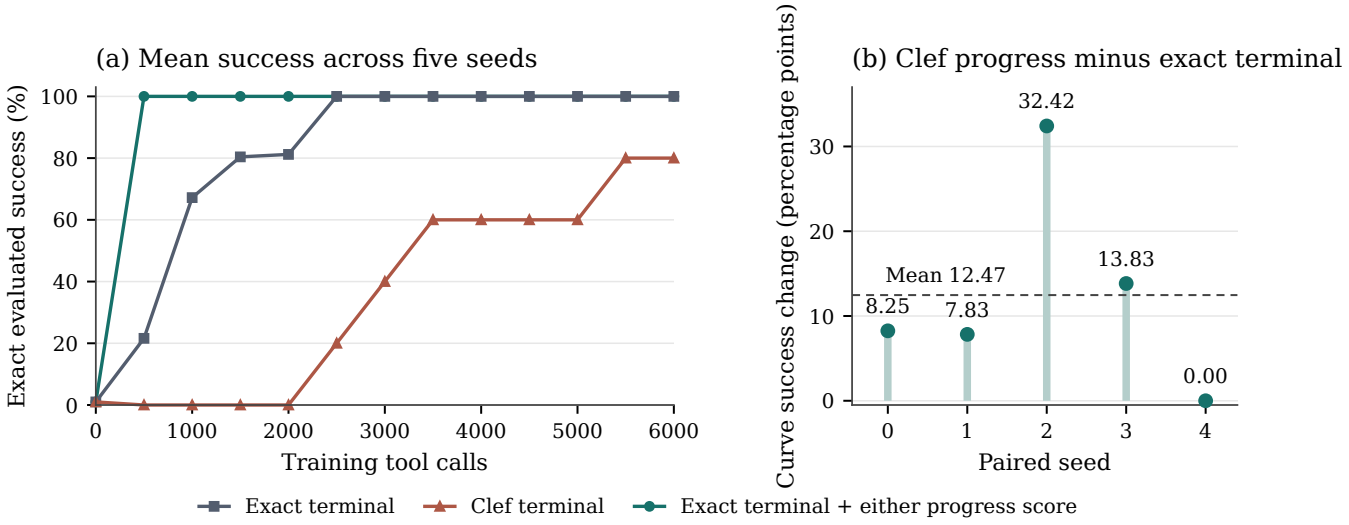



**Figure 1:** Recorded success and paired differences. Panel (a) shows means across five seeds; lines join measured checkpoints. The exact-progress and Clef-progress curves coincide at every checkpoint. Panel (b) shows each seed's change in normalized checkpoint area. The dashed line marks the mean change of 12.47 percentage points.

**Table 1:** Exact evaluation results across five paired seeds.

| Reward condition | Mean curve success | Final success |
|---|---|---|
| Exact terminal | 83.41% | 100% |
| Clef terminal | 35.04% | 80% |
| Exact terminal + exact progress | 95.88% | 100% |
| Exact terminal + Clef progress | 95.88% | 100% |

Both dense conditions reached 100% success at the first 500-call checkpoint in every seed. Their success curves matched, although their action sequences did not always match. The checkpoint interval cannot resolve the order of success within the first 500 calls. All exact-terminal seeds reached 100% by 2,500 calls.

Clef progress improved curve success by 12.47 percentage points on average. The paired changes ranged from 0 to 32.42 points, with a sample standard deviation of 12.19 points. Five seeds on one task do not support a broad claim of reliable improvement.

### 3.2 Judge errors and the failed policy

The judge audit used 72 clean states and 12 stress variants from the same workflow. Terminal accuracy was 95.83%, exactly equal to a baseline that always predicted failure. Only three clean states represented success. The judge found all three but falsely accepted three submissions without verification. Its Brier score, the mean squared error of its success probability, was 0.03139. The constant-prevalence baseline scored 0.03993.

The progress score had mean absolute error 0.06433 against exact stage progress. The judge correctly ordered all 52 tested state pairs by exact success probability under uniform random future actions. These pairs came from one workflow, not 52 independent tasks. False success claims changed progress scores by 0.02528 on average; key order caused no change in the tested pairs.

The terminal-judge condition retained one failed policy at the end of training. Seed 3 followed `transform`, `fetch`, `transform`, and `submit`. The first action had no effect, and the policy omitted `verify`. The final judge reward was 0.8129 despite exact failure.

The original analysis reports a higher discounted judge reward for a verified continuation. This observed failure does not prove that the optimal judge-reward policy must omit verification. It shows that the available training budget did not remove a consequential judge error.

## 4 Compute and the cost claim

The audit made 168 model requests, and training made 250. Their combined inference time was 85.59 seconds across 418 requests. Mean uncached latency was 204.77 ms; median latency was 202.53 ms and the 95th percentile was 206.15 ms. These times exclude model load, download, package installation, and idle time. The run used PyTorch's fallback implementation for linear attention rather than optional optimized kernels. It did not measure maximum model throughput.

Training reused 63,299 scores from 63,549 logical judge requests. Thus, 99.61% of requests used a cache, which stores scores for reuse. Each condition and seed began with an empty cache. The small set of distinct states made score reuse unusually effective. The measured latency describes fresh requests; the short overall run also depends on extensive reuse.

The listed GPU rate was $1.99 per hour. The original estimate from launch to the termination request was $0.36, with setup and idle time included. This estimate was not an invoice. All reward conditions had equal tool-call budgets, not equal dollar budgets. No larger judge ran in this comparison. The pilot does not establish a cost advantage over another judge.

## 5 Limits and the next test

The exact progress control explains the central limit: a simple task-specific score supplied the same measured benefit as Clef. The pilot establishes useful feedback for this policy and task; it does not isolate a unique benefit from model judgment. The task had a short, fixed sequence and a small state space. The five seeds measured random variation within that task, not variation across tasks. Progress scores also do not establish calibrated probabilities of future success.

A later plan targeted retail tasks in the $\tau^2$ benchmark with a language-model policy. That follow-up stopped before language-model training and contributes no results here. The present evidence supports no claim about better language-model weights or replacement of a conventional language-model judge.

A useful next test needs three controls:

1. Unseen tasks need independent final checks and distinct intermediate traces.
2. Clef needs comparison with a larger judge and a task-specific progress rule where one exists.
3. Policy tests need separate comparisons at equal tool-call budgets and equal total cost, with fresh judge requests included.

These are proposed tests, not completed work.

## 6 Evidence and reproducibility

The archived source report and data share commit `bd2023c14803`. The experiment code used commit `d3f9ca41069a`. The Clef model revision was `17f0b0ad64efb65d273590632833508766b2aae6`. The figure script recalculates all curve measures from the original checkpoint file and checks them against the saved summary. The data files preserve the source values and full revision identifiers. The original report placed raw logs outside Git; this report does not reconstruct absent traces.