Interoception: Teaching LLMs to Reason on a Wallclock Budget

Harsh Singh
Department of Computer Science
Stanford University
singhh@stanford.edu

Extended Abstract

Large language models reason in *tokens*, not *seconds*. They have no internal clock, so under a wallclock deadline they cannot trade off thinking longer (more accurate) against answering now (lower latency). This matters as test-time compute grows: inference now drives both accuracy and serving cost, and production systems answer within latency budgets. A model that **paces its reasoning to a time budget** T is *resource-rational*. We ask whether RL can teach this calibration on **Countdown** (reach a target integer using $+, -, \times, \div$), a search task where extra thinking genuinely buys accuracy, so the speed/accuracy trade-off is real rather than cosmetic.

We build a **simulated-time, multi-turn RL environment**: the model reasons in 128-token chunks; a hardware-latency simulator (**hwprop**) converts tokens to a deterministic wallclock time t; and the prompt reports elapsed and remaining time each turn against a budget $T \sim \mathcal{U}(1, 40)$ s. We train Qwen3-4B with GRPO and LoRA on top of the Prime Intellect `prime-rl/verifiers` stack, and measure calibration by the correlation $r(t, T)$ between commit time and budget (paced $\Rightarrow$ t tracks T). All headline numbers are computed on a fixed held-out probe of 498 problems, with correctness re-scored by the task's exact solver.

Our central finding is a **discovery, then a wall**. The discovery: a "remaining-budget" prompt makes the *untrained* model pace—commit time tracks T with $r \approx +0.79$ while it already solves 15% of problems—so the bottleneck is not time *awareness* but *reward shaping*. The wall: *every* outcome-reward RL configuration we tried trades this pacing for accuracy. The original multiplicative reward $c \cdot f$ raises accuracy $0.15 \to 0.41$ but erodes pacing ($r : 0.79 \to 0.16$); correctness-only RL is the extreme, reaching 0.55 accuracy while r collapses to -0.01 and the model abandons the budget on 42% of problems (committing at $\sim$4$\times$ the budget). We then ran a reward-engineering campaign to recover pacing without losing accuracy: (i) an **additive** reward $c + \lambda f$, a knife-edge ($\lambda \leq 0.15$ ignores time and reaches 0.39–0.50 accuracy; λ=0.30 restores pacing, r=+0.90, but caps accuracy at 0.17); (ii) a **windowed asymmetric-Gaussian** reward peaked at t=T with a sweep over the early-commit penalty σ_{under}; and (iii) a **two-stage KL curriculum** that first learns to pace, then trains correctness-only while a KL anchor pulls back toward the pacing policy.

None broke the trade-off. Across six reward families, runs fall into one of **two basins**: a *pacing* basin ($r \approx 0.8$–0.9, accuracy 0.15–0.21) or a *correctness* basin (accuracy 0.32–0.55, $r \approx 0$, commit time $\approx$ the model's T-independent natural solve time), with no run in the interior. The KL curriculum fails at *every* anchor strength $\beta \in \{0, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$: accuracy rises to ~ 0.34 but r returns to ≈ 0 regardless of β. We also flag a methodological caution: an apparent "σ_{under}=0.17 inverts calibration" result is, on inspection, a *basin/seed* effect (a monotone knob cannot produce a non-monotone flip mid-range; n=1 per setting, shared seed) plus a binned-mean artifact over a bimodal commit-time distribution—not a property of σ.

Contributions. (1) To our knowledge, the first study of RL for *wallclock-time* (not token-count) calibration, with a reusable simulated-time multi-turn environment. (2) The finding that time-pacing is a prompt-elicited capability of the *base* model that outcome-reward RL systematically un-learns. (3) A controlled comparison of six reward/curriculum designs revealing a robust two-basin correctness–calibration frontier, diagnosed as the precedence of the correctness gradient. (4) A caution on how single-seed sweeps and binned-mean plots can manufacture a spurious parameter effect.

Abstract

We study whether reinforcement learning can teach a language model to pace its reasoning to a wallclock time budget. We introduce a simulated-time, multi-turn RL environment in which a latency simulator converts generated tokens to deterministic elapsed seconds and the prompt reports elapsed and remaining time against a per-episode budget T. Training Qwen3-4B with GRPO and LoRA on Countdown, we find that the *untrained* model already paces well when prompted with the remaining budget ($r(t, T) \approx +0.79$ at 15% accuracy), but that outcome-reward RL systematically erodes this calibration as it improves accuracy. We design and evaluate six reward families intended to preserve pacing—a multiplicative gate, an additive bonus, a windowed asymmetric-Gaussian reward with an early-commit-penalty sweep, a value-of-computation cost, and a two-stage KL curriculum—and find a robust two-basin structure: policies either pace ($r \approx 0.9$, accuracy ≤ 0.21) or maximize correctness ($r \approx 0$, T-independent commit time, accuracy up to 0.55), with no run in the interior. We diagnose the cause as the precedence of correctness under outcome reward and the conflation of the two objectives under a multiplicative gate, show that KL-anchoring to a pacing policy does not help at any strength, and caution against a spurious single-seed parameter effect in our own windowed sweep. All numbers are computed on a fixed held-out probe with correctness re-scored by the exact task solver.

1 Introduction

Reasoning language models obtain much of their accuracy from *test-time compute*: they think in long chains before answering [Snell et al., 2024, DeepSeek-AI, 2025]. But that thinking is measured, controlled, and rewarded in *tokens*, whereas the resource that actually constrains a deployed system is *time*. The two are not interchangeable. Wallclock latency depends on hardware, batch size, sequence length, and serving conditions, and it is not a monotone function of token count. A model asked to "answer within T seconds" has no way to perceive T: it has no clock, and nothing in standard training couples its decoding length to elapsed real time.

We call the missing capability *interoception*—a model's awareness of, and ability to act on, its own resource consumption as it reasons. Concretely, a time-calibrated model should **think longer when it has more time and commit sooner when it has less**, trading accuracy for latency in a controlled way. This is the resource-rational behavior a serving system needs, and it is exactly what current reasoning models lack.

We study this on **Countdown**: given a set of source numbers and a target, produce an arithmetic expression (using $+, -, \times, \div$, each number once) that evaluates to the target. Countdown is a search problem where additional reasoning genuinely raises the success probability, so a time budget creates a *real* speed/accuracy trade-off rather than a cosmetic one; it has become a standard small-scale RL-for-reasoning benchmark [Gandhi et al., 2024, Pan et al., 2025]. Our question is narrow and concrete: *can RL teach a model to calibrate its commit time to a stated wallclock budget, and if so, with what reward?*

Our investigation produced a discovery and then a wall. The discovery is that the bottleneck is **not** time-awareness: with a prompt that simply reports the remaining budget each turn, the *untrained* model already paces—its commit time tracks T with $r \approx +0.79$, while solving 15% of problems. The wall is that *outcome-reward RL erodes this*: optimizing for correctness (alone, or gated by a timing term) systematically flattens the commit time toward a T-independent constant. We then engineered and compared six reward/curriculum designs aimed at preserving pacing while improving accuracy, and found a robust **two-basin** structure that none escaped. This report documents the environment, the reward designs, the controlled results, and—importantly—a methodological caution about how easily a single-seed sweep can appear to show a parameter effect that is really basin/seed variance.

2 Related Work

Test-time compute and reasoning. Much of the recent accuracy gain in LLMs comes from spending more compute at inference: chain-of-thought prompting elicits multi-step reasoning [Wei et al., 2022], and RL-trained reasoning models extend this to long deliberate chains [DeepSeek-AI, 2025, Kimi Team, 2025]. Compute-optimal scaling shows that allocating this test-time compute well can beat scaling parameters [Snell et al., 2024]. But this compute is highly variable and directly drives serving cost and latency, which motivates *controlling* how much a model spends—the problem we study, in wallclock terms.

Controlling reasoning length (by tokens). A large body of work controls test-time compute, but almost always in token units. *Budget forcing* in s1 [Muennighoff et al., 2025] caps or extends decoding by truncating the chain or appending "Wait" tokens; length-controlled RL (L1/LCPO) [Aggarwal and Welleck, 2025] trains a model to hit a prompt-specified *token* length; token-budget prompting (TALE) [Han et al., 2024] elicits answers within a stated token allowance; and RL recipes such as Kimi k1.5 [Kimi Team, 2025] add an explicit length penalty to the reward. These make length *controllable*, but the controlled quantity is tokens, not wallclock time.

Overthinking and efficient reasoning. A parallel line observes that reasoning models *overthink*—spending far more tokens than a problem warrants [Chen et al., 2024]—and trains them to reason more efficiently, e.g. by rewarding shorter correct traces [Arora and Zanette, 2025]; recent surveys catalogue the rapidly growing space [Sui et al., 2025]. This work targets *average* token efficiency rather than adherence to an externally specified, per-query *time* budget, and again does not perceive wallclock time.

Difficulty-aware computation and metareasoning. A third line scales the thinking penalty by how much computation is *worth* on a given input. Adaptive Length Penalties [Xiang et al., 2025] weight a per-token cost by a problem's online pass rate, so easy problems are pushed to answer fast while hard ones may think; rational metareasoning for LLMs [Sabbata et al., 2024] charges a cost for unnecessary computation derived from a value-of-computation analysis, echoing the classical metareasoning framing of bounded rationality. We adapt the ALP idea from token-length to time (Section 3.6), but our central results concern calibration rather than raw efficiency.

Adaptive computation: models that decide their own compute. Predating LLM reasoning, a line of work lets a network learn *how much* computation to spend: Adaptive Computation Time [Graves, 2016] and PonderNet [Banino et al., 2021] learn a halting policy over recurrent steps, and Confident Adaptive Language Modeling [Schuster et al., 2022] exits decoding early when confident. These models choose their own compute, but none is trained to meet an *externally imposed* wallclock budget that it must perceive and pace against—the setting we introduce.

RL, outcome reward, and reward hacking. Our training follows the now-standard recipe of outcome-reward RL with GRPO [Shao et al., 2024, DeepSeek-AI, 2025] and parameter-efficient LoRA adapters [Hu et al., 2021], on the Prime Intellect `prime-rl` trainer and its `verifiers` environment library [Intellect, 2025]; our task setup follows Countdown RL reproductions [Gandhi et al., 2024, Pan et al., 2025]. The phenomenon we report—that outcome reward optimizes the verifiable objective (correctness) and *erodes* an un-rewarded but desirable behavior (pacing)—is a calibration analogue of reward hacking and specification gaming [Skalse et al., 2022], and of the alignment tax familiar from RLHF [Ouyang et al., 2022].

Gap. All of the length-control work above targets a *token* budget. We instead target **wallclock time**—hardware-dependent and not a monotone function of tokens—by (i) converting tokens to time with a latency simulator and (ii) injecting elapsed *and* remaining time into the prompt every turn. To our knowledge this is the first study of RL for wallclock-time calibration, and the first to report that the capability is present in the base model and *removed* by outcome-reward RL.

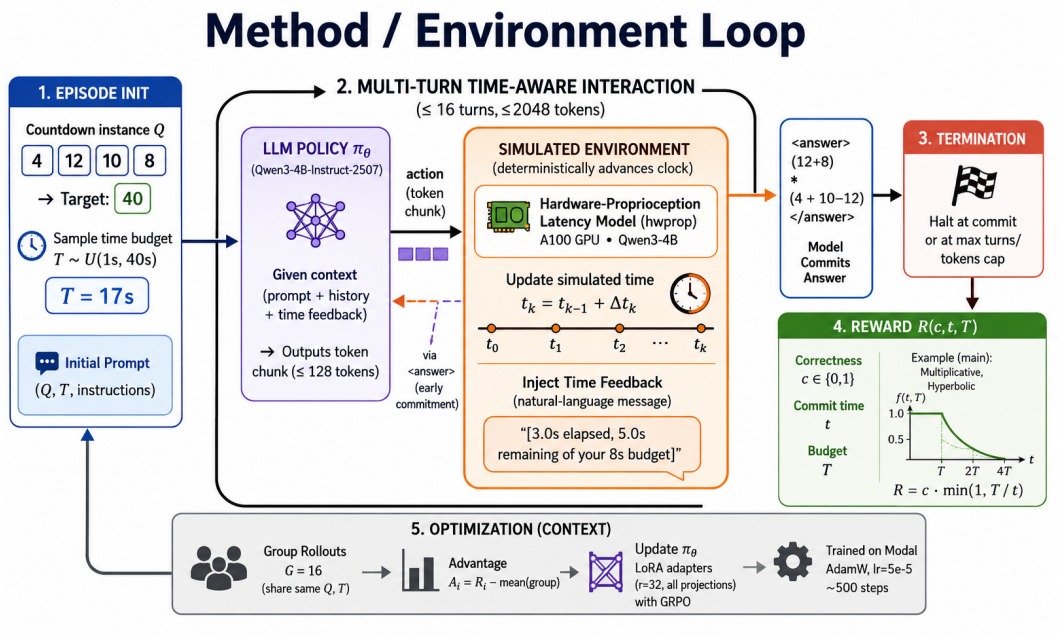

Figure 1: **The simulated-time RL environment and training loop.** Each episode samples a Countdown problem and a budget T. The policy reasons in 128-token chunks; the **hwprop** latency simulator converts tokens to deterministic elapsed seconds and injects elapsed and remaining time back into the prompt each turn; the episode terminates on commit, context cap, or time ceiling; the reward scores correctness and timing; GRPO updates a LoRA adapter on Qwen3-4B.

3 Method

Figure 1 overviews the environment and training loop, which we specify component by component below.

3.1 Problem formulation

A Countdown instance $Q = (\{n_1, \ldots, n_k\}, g)$ provides source numbers and a target g; a solution is an arithmetic expression over $\{+, -, \times, \div\}$ that uses each n_j exactly once and evaluates to g. We frame each episode as a budgeted multi-turn decision process. The policy π_θ (Qwen3-4B with a LoRA adapter) receives Q and a wallclock budget T and reasons over up to $N{=}16$ turns of at most 128 generated tokens each; after any turn it may continue or *commit* by emitting `<answer>` e `</answer>`. An episode terminates when (i) the model commits, (ii) it reaches N turns, or (iii) the context fills (sequence length 2048). We deliberately *disable* the optional hard time cutoff at $m{\cdot}T$: an early time wall truncates long rollouts at small T and superficially *looks* like pacing while being mere eviction, confounding the calibration signal. The committed expression is scored by an exact arithmetic solver (Unicode-normalized operators, tolerance 10^{-5}): correctness $c = 1$ iff e is a valid solution of Q, else 0.

3.2 Simulated wallclock time

To make per-turn latency cheap and reproducible inside the RL loop, we advance a simulated clock with our **hwprop** latency model. After turn k emits a chunk of d_k tokens on a context of p_k tokens, elapsed time advances by

$$\Delta t_k \;=\; \mathbf{1}[k{=}0]\,\tau_{\text{prefill}}(p_k) \;+\; \tau_{\text{decode}}(p_k, d_k), \qquad t \;=\; \textstyle\sum_k \Delta t_k,$$

where $\tau_{\text{prefill}}, \tau_{\text{decode}}$ come from `hwprop.simulate_latency` for an A100-80GB serving Qwen3-4B. Decode costs $\approx$35 ms/token and is nearly flat in context (so a 128-token chunk $\approx$4.5 s); prefill is charged only on the first turn, assuming prefix caching across turns. With $N{=}16$ turns and 2048-token contexts the clock saturates at $\approx$50–72 s. Time is therefore a smooth, deterministic,

monotone function of tokens generated—which gives GRPO low-variance group baselines and lets us substitute counterfactual hardware—and it is surfaced to the model only in *seconds*, never tokens. (At evaluation we can instead use measured per-turn wallclock; the calibration conclusions are unchanged.)

3.3 Prompt and per-turn time injection

The budget enters *entirely* through the prompt. The system message states the budget ("`You have only` T `seconds total`"); the user message states the problem ("`Using the numbers [...] find an expression that equals` g `...put the final expression inside <answer>...</answer>`"). After each turn the environment appends a natural-language clock message. We compare three injection variants: **base** ("`[X.Xs elapsed]`"), **remaining-budget** ("`[8.5s elapsed, 3.5s remaining of your 12s budget]`"—restating the budget and remaining time, with an instruction to commit when nearly out of time), and **strict-pace** (the same per-turn information without the commit-now instruction). The remaining-budget variant is what makes T salient enough for the base model to pace, and all main results use it.

3.4 Budget assignment and group invariance

Each problem i is assigned a single budget $T_i \sim \mathcal{U}(1, 40)\,\mathrm{s}$ by a seeded RNG (`seed` = `dataset_seed` $\oplus\, h(i)$), so the assignment is reproducible and the per-problem T is the dominant source of cross-seed variance. Crucially, *all G rollouts of problem i in a GRPO group share T_i*. GRPO's baseline is the within-group mean reward; if T were resampled per rollout, some rollouts would be "easier" purely by budget and would poison the advantage estimate. Sharing T within the group keeps the group-relative advantage a clean estimate of *policy* quality at a fixed budget.

3.5 Training: GRPO with LoRA

We optimize π_θ with Group Relative Policy Optimization [Shao et al., 2024]. For each problem we sample a group of G rollouts $\{\tau_i\}_{i=1}^{G}$, score each with the reward R_i (below), and form the group-relative advantage

$$A_i \;=\; \frac{R_i - \mu}{\sigma}, \qquad \mu = \tfrac{1}{G}\textstyle\sum_j R_j, \quad \sigma = \mathrm{std}_j(R_j),$$

then take a clipped policy-gradient step that maximizes $\sum_i A_i \log \pi_\theta(\tau_i)$. Only a rank-32 LoRA adapter is trained; the base weights stay frozen. The rollout loop, prompt assembly, time injection, and scoring are implemented as a `verifiers` `MultiTurnEnv`, and rewards are composed in a `verifiers` `Rubric` as a weighted sum of reward functions. Per-rollout rewards are scored independently; *group* rewards (the value-of-computation design) are routed through the Rubric's group path, which hands one reward function all G rollouts of a problem at once so it can compute the online pass rate $\rho_q = \frac{1}{G}\sum_i c_i$.

3.6 Reward designs

Let $c \in \{0, 1\}$ be correctness, t the commit time, T the budget, and $f(t, T) \in [0, 1]$ a time-fit term. The families differ in f and in how c and f combine:

- **Multiplicative gate** (the original reward): $R = c \cdot f$ with $f = \min(1,\, T/t)$ (1 in budget, T/t past T). A correct-but-late answer is discounted; because a *wrong* answer pays 0 regardless of timing, this gives *no* pacing gradient when $c = 0$.
- **Correctness-only**: $R = c$ (the time term dropped). Isolates what pure outcome reward does to pacing.
- **Additive**: $R = c + \lambda_f\, f$ with $f = \min(1,\, T/t)$. Decouples the terms so timing is a marginal target even when c is high; we sweep $\lambda_f \in \{0.10, 0.15, 0.30\}$.
- **Windowed asymmetric Gaussian**: $R = c + \lambda_f\, f$ with $f = \exp\!\big[-\big((t-T)/(\sigma T)\big)^2\big]$ and $\sigma = \sigma_{\text{under}}$ if $t < T$ else σ_{over}. This replaces the flat-in-budget ceiling with a peak *at* $t{=}T$, penalizing early-commit and overrun asymmetrically; we fix $\lambda_f{=}0.30,\ \sigma_{\text{over}}{=}0.10$ and sweep $\sigma_{\text{under}} \in \{0.25, 0.17, 0.10\}$.

- **Value-of-computation (ALP-style [Xiang et al., 2025])**: a *group* reward $R_i = c_i - \gamma\, \rho_q\, \kappa_i$ with difficulty weight ρ_q (the online pass rate) and time cost $\kappa_i = \max(0,\, t_i - T_i)/t_{\max}$. Easy groups (high ρ_q) are pushed to commit fast while hard groups (low ρ_q; ρ_q=0 for an all-wrong group ⇒ no cost) feel little pressure. De-prioritized before producing calibration results; reported as backlog.

Each family is realized at the `Rubric` level from small reward functions—e.g. additive is `is_correct` (weight 1) plus `f_term` (weight λ_f)—alongside a battery of zero-weight diagnostics (c, f, t/T, turn count, commit rate) that are logged but excluded from the loss.

Two-stage KL curriculum. As a curriculum alternative to a single reward, we (i) train the additive λ_f=0.30 pacing policy, (ii) *merge* its LoRA adapter into the base weights to form π_{stage1}, then (iii) run a second stage with the correctness-only reward R=c plus a penalty $\beta\, \mathrm{KL}(\pi_\theta \,\|\, \pi_{\text{stage1}})$ anchoring the policy to the pacing solution. We sweep $\beta \in \{0, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$ to test whether KL alone can hold pacing in place while correctness improves.

3.7 Calibration metric

A policy paces well if it commits later when given more time. Our primary metric is the Pearson correlation $r(t, T)$ between commit time and budget over committed rollouts: perfect pacing ($t \approx T$) gives $r \to 1$, while a T-independent commit time gives $r \approx 0$. We report it alongside the commit-time-vs-T curve, the commit rate (fraction emitting an answer), the mean overshoot t/T, and accuracy (exact-solver correctness), all on a fixed held-out probe (Section 4).

4 Experimental Setup

Model and training. We fine-tune Qwen3-4B-Instruct [Team, 2025] with GRPO [Shao et al., 2024] and a rank-32 LoRA adapter (α=32, all projections) [Hu et al., 2021] using the `prime-rl/verifiers` stack [Intellect, 2025] on Modal (2×A100-80GB per run: vLLM generation on one GPU, the FSDP trainer on the other). Optimization uses AdamW at learning rate $5{\times}10^{-5}$ with a 2-step warmup, sequence length 2048, sampling temperature 1.0, group size G=16, batch size 128, and 200 optimizer steps (the correctness-only and early hyperbolic controls use a 500-step schedule), checkpointing the adapter every 25 steps. Training uses the deterministic `hwprop` clock (`timing_source = sim`, A100-80GB / Qwen3-4B) with the hard time wall disabled (`enforce_max_time = false`) and budgets $T \sim \mathcal{U}(1, 40)$ s assigned per problem (Section 3.4; `dataset_seed` = 0). Reward is computed by our `verifiers` environment, scoring correctness and timing per rollout (and, for the ALP design, per GRPO group).

Evaluation protocol. Evaluation is *decoupled* from the training reward: a probe re-runs each saved checkpoint on a **fixed held-out set of 498 Countdown problems** spanning the budget range, logging $(T, t, \text{commit?}, \text{completion})$ per rollout independently of which reward shaped training. For every reported policy we compute accuracy, commit rate, mean overshoot t/T, and $r(t, T)$ on this same probe. To guarantee comparability we **re-score correctness for all policies with the task's exact arithmetic solver** (parsing each rollout's committed expression and validating it against the problem's numbers and target); this also corrected two early checkpoints whose logged reward field was unreliable. All numbers in Section 5 are from this probe.

5 Results

5.1 Quantitative Evaluation

The base model paces; RL erodes it. Under the remaining-budget prompt, the *untrained* Qwen3-4B already paces: its commit time rises with T (r=+0.79, Figure 2), while solving 15% of problems and committing on all of them. The bottleneck is therefore not awareness. The original multiplicative reward $c \cdot f$ raises accuracy to 0.41 but *flattens* the commit-time curve (r=+0.16, mean overshoot t/T=2.8): it buys correctness by spending the budget. Correctness-only RL (the strong-prompt R=c control, 500 steps) is the extreme (Figure 3): accuracy climbs to 0.55, but r collapses to −0.01, mean overshoot grows to 4.3×, and the model *stops committing on* 42% *of problems*

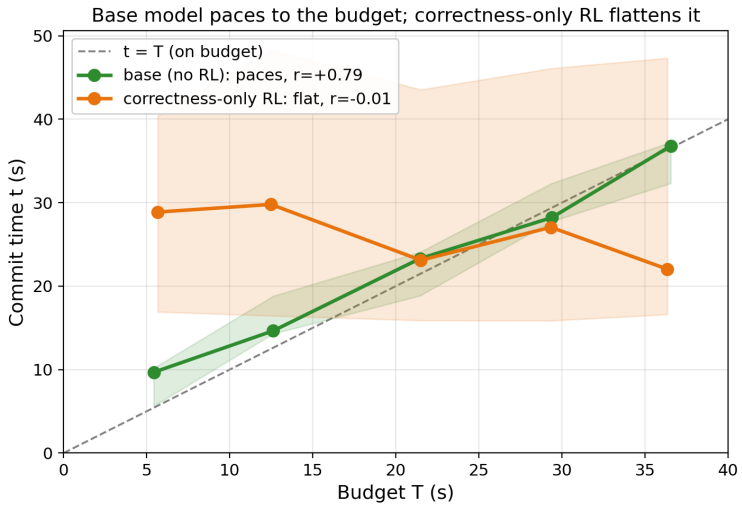

Figure 2: **The base model paces; RL erodes it.** Commit time vs. budget T on the held-out probe (per-bin median; shaded band = inter-quartile range). The untrained model (green) tracks $t{=}T$ with a tight spread ($r{=}{+}0.79$); correctness-only RL (orange) commits at a near-constant time irrespective of the budget ($r{=}{-}0.01$), with a wide, budget-insensitive spread.

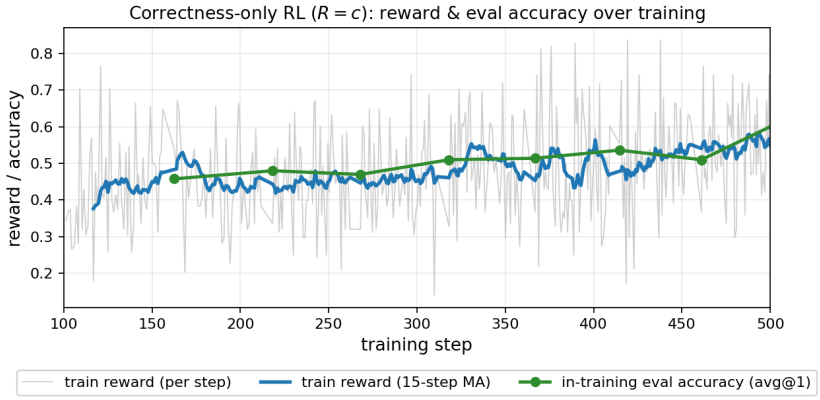

Figure 3: **Correctness-only RL ($R = c$, 500 steps).** Training reward (gray raw, blue 15-step moving average) and the in-training eval accuracy (green) both climb; on the held-out 498-problem probe the final policy reaches 0.55 accuracy. Meanwhile budget discipline degrades: the model increasingly spends the maximum available time and abandons the budget on 42% of problems (commit rate 58%, mean overshoot $t/T{\approx}4.3$).

(it exhausts the budget). The diagnosis is simple: outcome reward optimizes correctness first, and since more time only ever helps correctness on a search task, nothing preserves pacing once timing is unrewarded.

Additive reward: a knife-edge. Decoupling the terms with $R = c{+}\lambda f$ and sweeping λ (Figure 4) does not escape the trade-off; it merely chooses which side of it to sit on. At $\lambda{=}0.10$ and 0.15 the bonus is too small to compete with the correctness gradient: the runs land in the correctness basin (accuracy 0.39 and 0.50, but $r{=}{-}0.09$ and -0.06—no pacing). At $\lambda{=}0.30$ the time bonus dominates and pacing returns strongly ($r{=}{+}0.90$, $t/T{\approx}0.9$), but accuracy falls to 0.17. The usable band that achieves both is empty; the threshold between the two basins sits between $\lambda{=}0.15$ and 0.30.

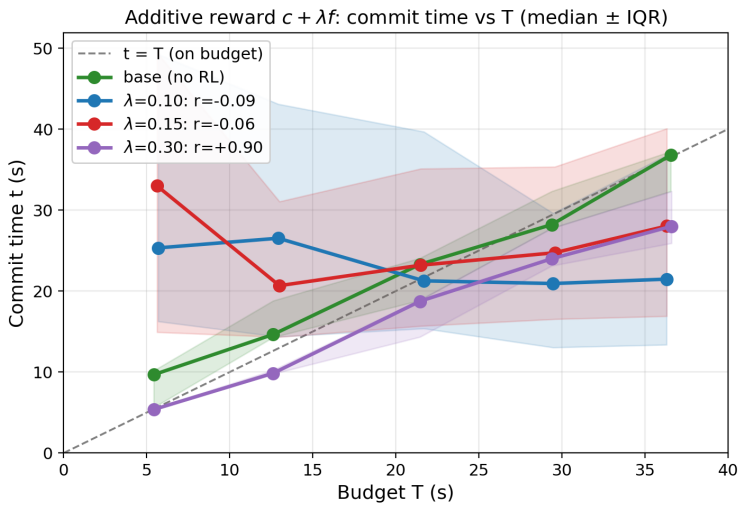

Figure 4: **Additive reward $c + \lambda f$ is a knife-edge.** Commit time vs. T (per-bin median; shaded band = IQR). $\lambda \in \{0.10, 0.15\}$ ignore the time term (flat, accuracy 0.39–0.50, no pacing); λ=0.30 (purple) tracks t=T (r=+0.90) but caps accuracy at 0.17; base (green) shown for reference. No setting achieves both.

Table 1: **Two-stage KL curriculum.** Stage-2 trains correctness-only with a KL anchor of strength β to the Stage-1 pacing policy. Accuracy rises but calibration $r(t, T)$ collapses to ≈ 0 at every β. (Base and Stage-1 shown for reference; held-out probe, n=498.)

Policy	β (kl_tau)	Accuracy	Commit rate	$r(t, T)$
Base (no RL)	—	0.15	100%	+0.79
Stage-1 (additive λ=0.30)	—	0.17	100%	+0.90
Stage-2 b0	0	0.33	39%	−0.02
Stage-2 b1	10^{-4}	0.25	79%	+0.03
Stage-2 b3	10^{-3}	0.34	46%	+0.02
Stage-2 b2	10^{-2}	0.35	44%	−0.04
Stage-2 b4	10^{-1}	0.32	45%	−0.04

Windowed asymmetric Gaussian and the σ_{under} sweep. Replacing the saturating $\min(1, T/t)$ with a Gaussian peaked at t=T, we fix λ=0.30, σ_{over}=0.10 and sweep the early-commit penalty $\sigma_{\text{under}} \in \{0.25, 0.17, 0.10\}$ at 200 steps (Figures 5, 6). Two of the three runs land in the pacing basin: σ_{under}=0.25 and 0.10 track T (r=+0.92 and +0.88) at accuracy 0.17 and 0.21. The third, σ_{under}=0.17, instead reaches the highest accuracy (0.34) with a commit-time curve that does not track T (r=−0.19). We analyze this apparent "inversion" in Section 5.2 and argue it is a basin/seed effect, not a property of σ.

Two-stage KL curriculum. If a single reward cannot hold both objectives, perhaps a curriculum can: first *learn* to pace, then improve correctness while a KL anchor prevents drift off the pacing solution. We merge the additive λ=0.30 (Stage-1) policy into the base weights, then run Stage-2 correctness-only RL with a KL penalty of strength β (Table 1, Figure 7). The curriculum fails at every anchor strength: Stage-1 alone is well-calibrated (r=+0.90, accuracy 0.17), but after Stage-2 accuracy rises to 0.25–0.35 while r returns to ≈ 0 for *all* β (including the heaviest anchor β=10^{-1}), and the commit rate drops to ∼40–46%. KL regularization toward a pacing policy is not sufficient to preserve pacing once the reward again rewards correctness only.

Summary: a frontier with two basins. Table 2 collects every policy on the common probe. They separate cleanly into a *pacing* basin (high r, accuracy 0.15–0.21) and a *correctness* basin (accuracy

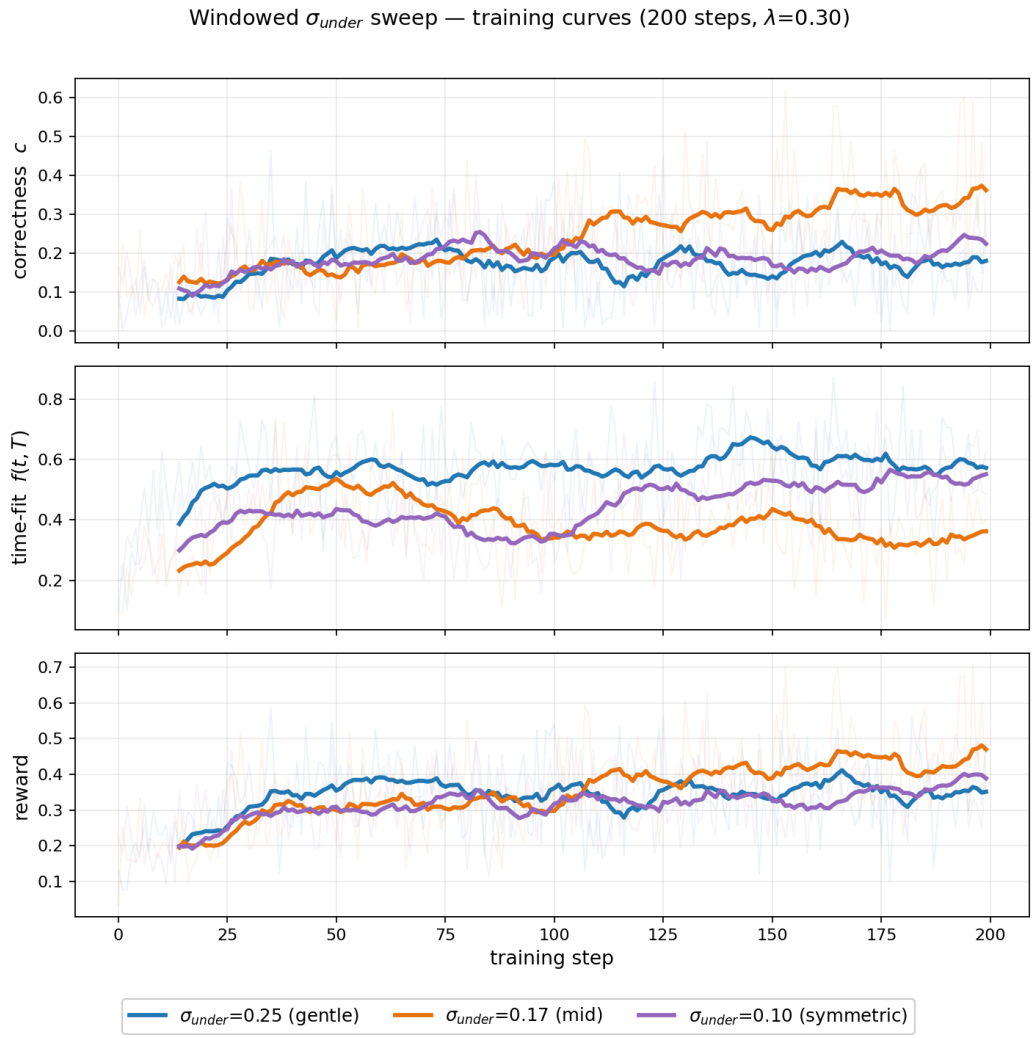

Figure 5: **Windowed σ_{under} sweep—training curves (200 steps, λ=0.30).** Correctness c (top), time-fit $f(t, T)$ (middle), total reward $c+\lambda f$ (bottom). σ_{under}=0.17 (orange) climbs to the highest correctness ($\sim$0.34); σ_{under}=0.25, 0.10 stay near 0.20.

0.32–0.55, $r \approx 0$); the interval between accuracy 0.21 and 0.32 is empty—no reward we tried reaches the interior with both high accuracy and high r.

5.2 Qualitative Analysis

Two behavioral modes. The quantitative split corresponds to two qualitatively different behaviors. *Pacing-basin* policies treat the budget as a deadline: they commit near t=T and almost always commit (commit rate 100%), so for a small budget they answer fast (often with a worse answer) and for a large budget they think longer. *Correctness-basin* policies treat the problem as the only objective: they "solve-then-stop," committing at roughly the model's natural solve time regardless of T, and when they cannot solve in the available context they simply never commit—hence the commit rate falling to 39–58% and the mean overshoot rising to 4× the budget. A concrete example from the probe: on a problem with budget T=9.9 s (numbers [29, 10, 8, 21]), the base model commits at 14.3 s—budget-aware, slightly over—whereas the correctness-only policy burns 39.5 s (4× over) before committing, having ignored the stated 10-second budget entirely.

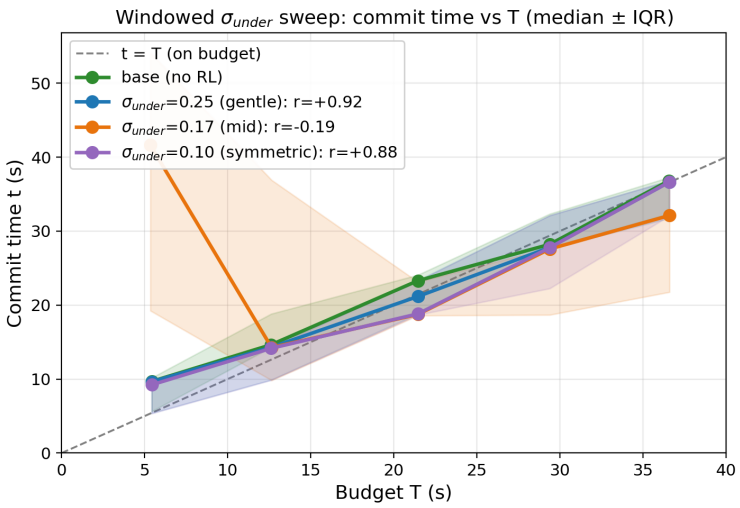

Figure 6: **Windowed σ_{under} sweep—commit time vs. T (median $\pm$ IQR).** σ_{under}=0.25, 0.10 track T with tight spreads (pacing basin, $r\approx+0.9$). The σ_{under}=0.17 median (orange) appears to slope the wrong way, but its *inter-quartile band is enormous* (e.g. $\sim$10–55 s at the smallest budget): the distribution is bimodal and T-independent (Section 5.2), so the median is not a reliable trend. With n=1 per setting and a shared seed this is a basin/seed effect, not a property of σ.

Table 2: **All reward families on the common held-out probe** (n=498**, exact-solver accuracy).** Every policy is in either the pacing basin (high r, low accuracy) or the correctness basin (high accuracy, $r \approx 0$).

Reward / curriculum	Setting	Accuracy	Commit	$r(t, T)$	Basin
Base (no RL)	remaining-budget prompt	0.15	100%	+0.79	pacing
Multiplicative $c \cdot f$	hyperbolic, 500 steps	0.41	78%	+0.16	correctness
Correctness-only R=c	strict prompt, 500 steps	0.55	58%	−0.01	correctness
Additive c+λf	λ=0.10	0.39	52%	−0.09	correctness
Additive c+λf	λ=0.15	0.50	58%	−0.06	correctness
Additive c+λf	λ=0.30	0.17	100%	+0.90	pacing
Windowed Gaussian	σ_{under}=0.25	0.17	100%	+0.92	pacing
Windowed Gaussian	σ_{under}=0.10	0.21	100%	+0.88	pacing
Windowed Gaussian	σ_{under}=0.17	0.34	96%	−0.19	correctness
Two-stage KL	any β	0.25–0.35	39–79%	≈ 0	correctness

A cautionary reading of the σ_{under}=0.17 run. It is tempting to read Figure 6 as "σ_{under}=0.17 inverts calibration," but that conclusion does not survive scrutiny, and we flag it as a methodological caution. *First, the mechanism is implausible*: σ_{under} is a monotone knob ($0.25 > 0.17 > 0.10$), yet the outcome would be non-monotone (track, invert, track). A monotone parameter cannot mechanistically produce a flip in the *middle* of its range; such a pattern is the signature of run-to-run variance. *Second, the design confounds it*: we ran n=1 per setting with a shared seed, so σ and seed are not separable. *Third, the plot exaggerates it*: the curve connects binned *means* over highly dispersed, partly-bimodal commit-time distributions. In the smallest-budget bin ($\bar{T} \approx 5.4$ s) the σ_{under}=0.17 committed rollouts have median commit time 42 s with inter-quartile range $[19, 55]$ s—a mixture of fast and slow commits, not a single "commits at 35 s" point. Read at the rollout level, this run is not inverted but T*-independent*: its commit time clusters near the model's natural solve time ($\approx$ 23 s median, IQR $[15, 32]$) regardless of T. In other words, σ_{under}=0.17 is not a special σ; it is one run that happened to fall into the correctness basin—the same basin as the correctness-only and KL runs—while the other two fell into the pacing basin. Separating σ from seed would require multiple seeds per setting, and reporting commit-time *distributions* (not binned means) would have surfaced the mixture directly. We keep this as a cautionary result rather than a σ finding.

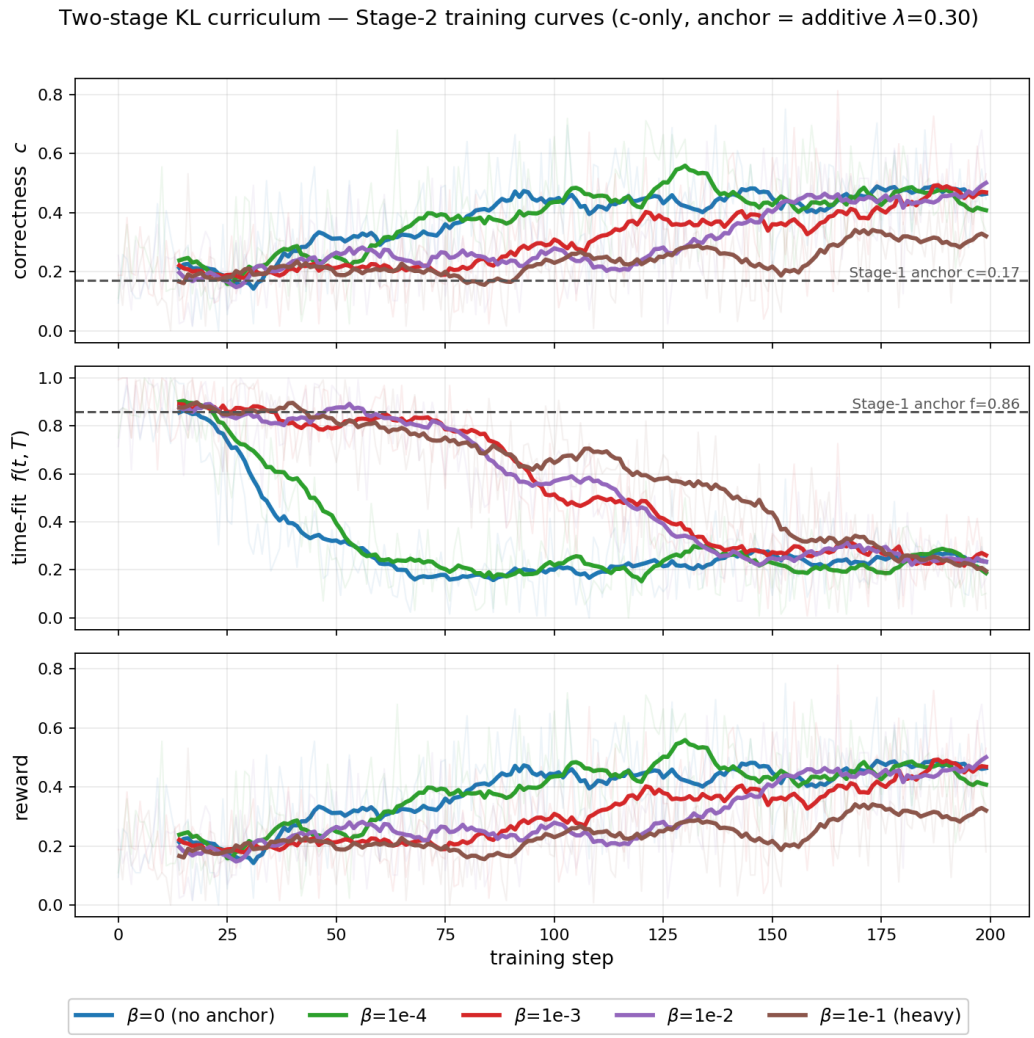

Figure 7: **Two-stage KL curriculum—Stage-2 training curves** across anchor strengths β. Correctness improves similarly for all β; none retains the Stage-1 pacing (Table 1).

6 Discussion

Why the frontier exists. Two design properties explain the two basins. (1) *Outcome reward optimizes correctness first.* On a search task, more time only ever helps correctness, so any reward that ultimately rewards correctness—directly, or after a curriculum—has a standing gradient toward spending the whole budget, and nothing opposes it once timing is unrewarded. (2) *The multiplicative gate conflates the objectives.* Because $c \cdot f$ folds timing into the correctness signal, the optimizer cannot separately preserve f; the additive and windowed forms decouple them but only relocate the operating point along the same trade-off. The KL curriculum shows that even an explicit pull back toward a pacing solution is overwhelmed by the correctness gradient. The implication is that a fix must make pacing *competitive with*, not merely additive to, correctness.

Limitations. (i) *Simulated, not measured, time*: hwprop is a deterministic proxy for A100 latency and does not model batching or contention. (ii) *One task, one model*: all results are Countdown with Qwen3-4B; the trade-off may differ where extra thinking helps less. (iii) *Single-seed sweeps*: as Section 5.2 shows, $n{=}1$ per setting cannot separate a parameter effect from basin/seed variance, and a binned-mean curve (or a single r) can hide a bimodal commit-time distribution; multi-seed runs and rollout-level distributional reporting are needed before any single-knob claim. (iv) The

difficulty-aware value-of-computation reward (ALP-style, $R = c - \gamma\rho_q t/t_{\max}$) was implemented and verified but de-prioritized before producing calibration results; it remains the most principled untested lever.

Future work. The diagnosis points to rewards that remove the correctness gradient *where it conflicts with pacing*: a difficulty-aware time cost that applies pressure only on problems the model can already solve (the ALP variant), so spending extra time on an easy problem is penalized while hard problems are left alone. Beyond reward shape, we would (a) run multi-seed sweeps and report commit-time distributions, not just r; (b) condition the budget on *sampled* hardware so the model must infer the token→time map; and (c) test whether a two-objective method that keeps pacing *competitive* (e.g. constrained RL with a pacing constraint) can reach the interior of the frontier.

7 Conclusion

We asked whether RL can teach a language model to pace its reasoning to a wallclock budget. The surprising answer is that the base model can already do so when prompted with its remaining budget ($r{=}{+}0.79$ at 15% accuracy), and that the harder problem is preventing outcome-reward RL from *un-learning* it. Across six reward and curriculum designs we find a robust correctness–calibration frontier with two basins and no interior solution, and we trace it to the precedence of the correctness gradient under outcome reward. Breaking the frontier requires making pacing competitive with correctness rather than merely additive to it—most promisingly via a difficulty-aware time cost that withholds time pressure exactly where thinking still helps.

8 Team Contributions

The CS224R project group is a single enrolled student, **Harsh Singh**, who is the sole author of this report. The broader research effort is shared with **Nicole Ma**, an undergraduate researcher in the Goodman lab who is *not* enrolled in CS224R and contributes as part of related lab work; per the course policy on shared projects, contributions to the full effort split roughly 50/50 (evidenced by the shared repository's commit history), with both contributors working on the environment, training infrastructure, and analysis, including the RL implementation. The portion reported here—the simulated-time environment and hwprop latency model, the wallclock-injection prompt and calibration metric, the windowed-Gaussian and value-of-computation reward designs and the σ_{under} sweep, and the verified cross-reward analysis—constitutes a complete CS224R deep-RL project on its own, comparable in scope to a standalone single-person submission.

Acknowledgments. We thank **Nicole Ma** for the equal collaboration on the broader project (baseline and control sweeps, the additive-reward λ sweep, the two-stage KL curriculum, and the `verifiers` environment refactor), and our external advisors **Michael Li** and **Kanishk Gandhi** for problem framing and methodology feedback. Compute was provided via Modal.

9 AI Tools Disclosure

AI tools (Anthropic's Claude, including the Claude Code CLI) were used as an aid during this project. Their use was limited to *standard coding tasks*—constructing data and evaluation pipelines, writing analysis and plotting scripts, LaTeX formatting, and debugging minor code issues—and to *writing assistance* in drafting and editing this report. The *essential components* of the project were developed independently by the author: the problem formulation (wallclock-time calibration), the simulated-time multi-turn environment and hwprop latency model, the reward designs and their analysis, and the experimental methodology. We did not use AI to implement core reinforcement-learning algorithms; GRPO is provided by the open-source `prime-rl/verifiers` framework [Intellect, 2025] rather than hand-implemented or AI-generated. I certify that I leveraged AI for standard coding tasks but not for the essential components of the project; AI assisted with writing and coding, but the core of the work was human effort.

References

Pranjal Aggarwal and Sean Welleck. L1: Controlling how long a reasoning model thinks with reinforcement learning, 2025.

Daman Arora and Andrea Zanette. Training language models to reason efficiently, 2025.

Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder, 2021.

Xingyu Chen et al. Do not think that much for 2+3=? on the overthinking of o1-like llms, 2024.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D. Goodman. Stream of search (sos): Learning to search in language, 2024.

Alex Graves. Adaptive computation time for recurrent neural networks, 2016.

Tingxu Han et al. Token-budget-aware llm reasoning, 2024.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

Prime Intellect. prime-rl: Agentic rl training at scale. `https://github.com/PrimeIntellect-ai/prime-rl`, 2025. with the `verifiers` environment library (`https://github.com/PrimeIntellect-ai/verifiers`); accessed 2026.

Kimi Team. Kimi k1.5: Scaling reinforcement learning with llms, 2025.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling, 2025.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.

Jiayi Pan, Junjie Zhang, Xingyao Wang, Lifan Yuan, Hao Peng, and Alane Suhr. Tinyzero. `https://github.com/Jiayi-Pan/TinyZero`, 2025. Accessed 2026.

C. Nicolò De Sabbata, Theodore R. Sumers, Badr AlKhamissi, Antoine Bosselut, and Thomas L. Griffiths. Rational metareasoning for large language models, 2024.

Tal Schuster et al. Confident adaptive language modeling, 2022.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking, 2022.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024.

Yang Sui et al. Stop overthinking: A survey on efficient reasoning for large language models, 2025.

Qwen Team. Qwen3 technical report, 2025.

Jason Wei et al. Chain-of-thought prompting elicits reasoning in large language models, 2022.

Violet Xiang, Chase Blagden, Rafael Rafailov, Nathan Lile, Sang Truong, Chelsea Finn, and Nick Haber. Just enough thinking: Efficient reasoning with adaptive length penalties reinforcement learning, 2025.