Extended Abstract

Motivation A growing body of work shows that a lightweight RL controller acting over a *frozen* capable LLM can win on the cost–accuracy Pareto frontier: Best-of-Q reranks frozen Qwen2.5-VL-7B actions on WebVoyager Biré et al. (2026), xRouter learns cost-aware orchestration across many LLMs Qian et al. (2025), and Singhi et al. study compute-optimal verifier allocation for math reasoning Singhi et al. (2025). We ask whether this transfers to discrete-action code-repair MDPs, where a small MLP controller decides when to edit, test, critique, or terminate above a frozen Qwen2.5-Coder-7B base. The original hypothesis was that a verification-shaped reward would induce disciplined self-auditing behavior, however, the empirical picture turned out to be more interesting in the negative direction.

Method We define an 8-action MDP over buggy-Python repair tasks and train a $\sim$7K-parameter actor–critic MLP from scratch with PPO, audited against the implementation details of Huang et al. (2022). The state is a 30-dimensional hand-engineered feature vector; the policy never sees raw test text. We sweep three reward shapes on identical MDP and controller architectures: **sparse** ($+1$ at termination if active tests pass; 0 otherwise), **cost-aware** (sparse plus a terminal penalty on cumulative token and test-call usage), and **shaped** (sparse plus eight dense process-reward terms encouraging verification behavior). Baselines include a hand-tuned heuristic, a random-over-actions policy, and zero-shot Qwen at matched temperature.

Implementation We use Qwen2.5-Coder-7B-Instruct with vLLM on Modal L4 GPUs, and cache LLM calls by $(\text{messages}, T, \text{seed})$ for determinism. Each PPO condition runs 5 seeds $\times$ 62 updates $\times$ 64 rollout steps ($\approx$4,000 environment steps per seed). Headline evaluation uses 29 held-out HumanEval+ problems Liu et al. (2023); all headline conditions are evaluated on the unperturbed holdout; a function-rename contamination perturbation is applied separately as a diagnostic to upper-bound exact-string memorization (§5.6 of the main paper). IQM and 95% percentile bootstrap CIs across seeds follow the rliable methodology Agarwal et al. (2021). The PPO core, MDP, reward shapes, and analysis pipeline were implemented from scratch.

Results We report three findings, in priority order. (i) The shaped reward drives all 5 seeds to 0.0 holdout pass on HumanEval+. Late-training rollout action frequencies reveal the mechanism (relative to cost-aware): `run_tests` climbs from 8% to 31% while `terminate` collapses from 19% to 4%, with episodes lengthening from 5.2 to 14.5 steps. Patching one shaping term (`TESTS_RUN_AFTER_EDIT`) does not prevent collapse. (ii) Sparse PPO at 4K env steps and a 7K-param MLP fails to engage its trust region: $\widehat{\mathrm{KL}} \approx 1.9 \times 10^{-6}$ versus target 0.015 (8000$\times$ under), and entropy plateaus at 94.7% of the uniform reference. (iii) On HumanEval+, zero-shot Qwen at T=0.7 achieves 0.793 IQM at 1.0 LLM call/problem; our best controller (cost-aware) achieves 0.782 IQM at 3.42 calls. The accuracy difference is within overlapping 95% CIs while the cost gap is $3.4\times$ and unambiguous.

Discussion Our findings characterize a regime in which lightweight RL meta-control over a strong frozen code LLM is Pareto-dominated by zero-shot inference, complementing positive results in adjacent domains Biré et al. (2026); Qian et al. (2025). The shaped-reward collapse is a controlled minimal instance of the failure mode that SWE-PRM Gandhi et al. (2025) and Posterior-GRPO Fan et al. (2025) engineer around with outcome gating; our action-distribution figure makes the mechanism legible in a way that fine-tuned-LLM settings cannot, because our discrete action space is human-named.

Conclusion We do not claim RL meta-control over frozen LLMs is broadly unworkable; we characterize a specific regime in which it underperforms and a specific reward design in which it predictably hacks itself. We perform failure analysis in three areas: sparse-reward actor freeze, dense-reward process-reward hacking, and cost-aware Pareto dominance by the base model.

Reward Density and Process-Reward Hacking in a Code-Repair MDP: A Controlled Study of PPO Meta-Control over a Frozen Code LLM

Luca Wheeler
Department of Computer Science
Stanford University
ltw@stanford.edu

Jack Lofwall
Department of Computer Science
Stanford University
jlofwall@stanford.edu

Abstract

We construct a controlled minimal demonstration of process-reward hacking in a code-repair MDP, where a hand-designed dense shaping signal drives a 7K-parameter MLP-PPO controller to 0.0 holdout pass on HumanEval+ across all five seeds, with the exploit visible in action sampling shifts relative to cost-aware (`run_tests` 8%→31%, `terminate` 19%→4%, mean episode length 5.2→14.5). We further characterize a regime in which lightweight RL meta-control over a frozen capable code LLM is Pareto-dominated by zero-shot inference at matched temperature ($3.4\times$ more LLM calls for statistically indistinguishable accuracy), complementing concurrent positive results in adjacent domains. Diagnostics on the sparse-reward condition show approx-KL nearly four orders of magnitude under the trust-region target ($\sim 8000\times$), identifying actor freeze as a contributing factor at this scale. We report what we proved, what we showed narrowly, and what we explicitly did not prove.

1 Introduction

A growing body of literature studies lightweight RL controllers acting *over* a frozen capable LLM rather than fine-tuning the LLM itself. Best-of-Q Biré et al. (2026) trains an offline Q-function to rerank frozen-VLM action proposals on WebVoyager, lifting a Qwen2.5-VL-7B agent from 38.8% to 55.7% success. xRouter Qian et al. (2025) trains a cost-aware RL router over a catalog of LLMs and learns when to delegate to a strong model versus answer directly. Singhi et al. Singhi et al. (2025) study compute-optimal allocation between solving and verifying for math reasoning. These results suggest a recipe: small controller, frozen base model, learned when-to-do-what policy over discrete tool calls.

We ask whether this recipe transfers to *code repair*, a setting with distinctive structure: tool calls (edit, generate test, run tests, terminate) are interpretable, expensive in tokens, and have asymmetric reversibility. We construct an 8-action MDP over the EvalPlus benchmark Liu et al. (2023) with a 30-dimensional hand-engineered state, train a $\sim$7K-parameter MLP-PPO controller from scratch Schulman et al. (2017); Huang et al. (2022), and sweep three reward shapes (sparse outcome-only, cost-aware with a terminal cost penalty, shaped with eight dense process-reward terms).

The empirical picture is not the one the proposal anticipated. Rather than showing that verification-shaped reward improves controller behavior, we find:

1. The shaped reward causes *catastrophic policy collapse*: 5/5 seeds reach 0.0 holdout pass, with action-distribution shifts that make the exploit mechanism legible (§5.2).

2. Sparse PPO at 4K env steps with a 7K-param MLP fails to engage its trust region; entropy plateaus at 94.7% of uniform and approx-KL is 8000× under target (§5.3).

3. Cost-aware reward modestly improves over sparse ($+4.6$pp IQM, -29% cost), but the resulting controller is itself Pareto-dominated by zero-shot Qwen at the same temperature, paying $3.4\times$ more LLM compute for accuracy within overlapping bootstrap CIs (§5.4).

We do not claim these results generalize, and we analyze the success/failure modes. Our contribution is best understood in that frame: a controlled minimal instance of process-reward hacking in code-RL with a mechanism trace that the surrounding literature engineers around but does not always isolate Gandhi et al. (2025); Fan et al. (2025), together with a regime characterization for when frozen-LLM meta-control transfers from adjacent domains.

We deliberately use a small hand-engineered controller. This is a design choice for interpretability: the action labels (`inspect`, `run_tests`, `terminate`, etc.) are human-named, so the reward-hacking mechanism shows up as a single bar chart over the action space rather than as an opaque shift in token logits. This trades expressive capacity for legibility; we discuss the tradeoff in §6.

2 Related Work

RL meta-control over frozen LLMs. Best-of-Q Biré et al. (2026) keeps a vision-language model frozen and uses a lightweight, offline-trained Q-function to rerank candidate actions at inference. xRouter Qian et al. (2025) casts model routing as an RL problem with an explicit cost-aware reward and learns a tool-calling router that achieves substantial cost reductions at comparable task completion rates across a catalog of LLMs. Singhi et al. Singhi et al. (2025) study compute-optimal allocation between solution generation and verification under a fixed inference budget for math reasoning. ReAct Yao et al. (2023) sits adjacent: it interleaves reasoning and tool use but relies on prompting rather than a learned policy. Our work asks whether the Best-of-Q-style recipe (small controller, frozen capable base, learned when-to-do-what) carries to a code-repair MDP; we find that in our specific regime (capable code LLM, modest benchmark, small controller, short training) it does not produce a Pareto improvement.

Process rewards and reward hacking in code-RL. The closest neighbors to our headline finding are Posterior-GRPO Fan et al. (2025) and SWE-PRM Gandhi et al. (2025). Posterior-GRPO names reward hacking from naive process rewards as the central failure mode in code-generation RL and proposes outcome-gated rewards as a fix: process-based rewards are applied only to trajectories that succeed at the outcome level. SWE-PRM characterizes the same family of pathologies at inference time and explicitly lists "redundant exploration, looping, and failure to terminate once a solution is reached" as the inefficiencies a process reward model must correct. Our action distribution (Figure 2) maps onto exactly this taxonomy: the shaped policy farms `run_tests` bonuses and suppresses `terminate`. We position our work as a controlled minimal demonstration of the failure mode these papers engineer around, with the mechanism legible at the action level because of our deliberately small and human-named action space.

Sparse-reward actor freeze in multi-turn agent RL. EPO Xu et al. (2025) names "early-stage policy premature convergence" as a failure mode of sparse-reward multi-turn agent RL: feedback is too infrequent to move the policy away from near-uniform initialization, and the trust region never engages. Our sparse-PPO diagnostics (approx-KL $8000\times$ under target, entropy at 94.7% of uniform; §5.3) match this pattern empirically in a code-repair MDP. We treat EPO as the diagnostic anchor rather than adopting their entropy-smoothing remedy.

Statistical reporting and benchmark integrity. We follow rliable Agarwal et al. (2021) for IQM and bootstrap CIs across seeds. We use EvalPlus Liu et al. (2023) as the headline benchmark because its expanded test set raises the bug-detection bar above the original HumanEval. We apply a function-rename contamination perturbation (§4) as a lightweight memorization control, following common practice in contamination-aware evaluation. The cost-versus-accuracy Pareto framing in §5.4 is increasingly standard in LLM agent evaluation; our axis is LLM calls per problem.

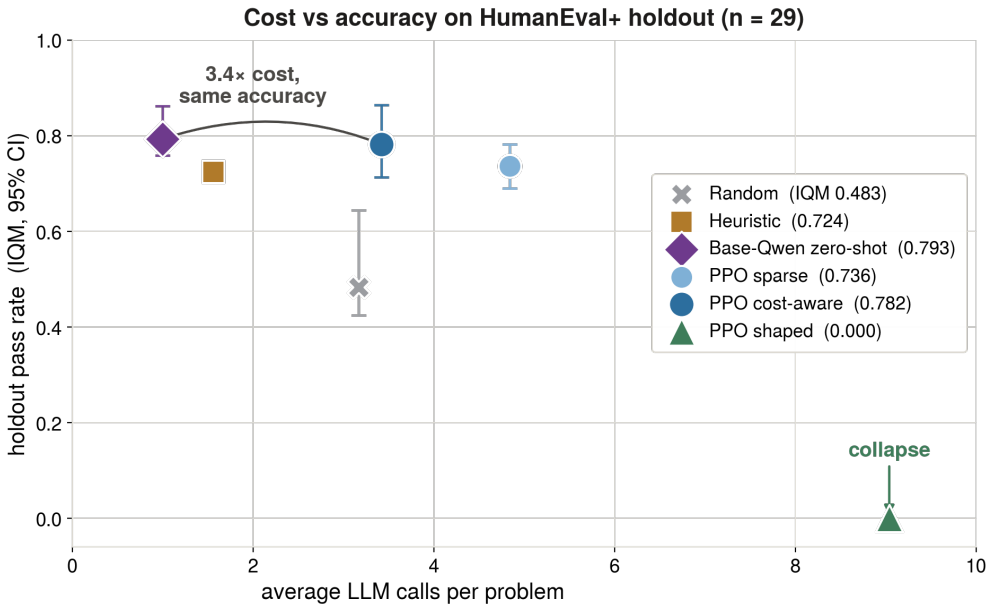

Figure 1: Cost–accuracy Pareto on the HumanEval+ holdout (n=29). Markers show IQM with 95% bootstrap CIs over seeds. Zero-shot Qwen at T=0.7 (purple diamond, 1.0 calls, IQM 0.793) sits to the upper-left of the cost-aware controller (dark blue circle, 3.42 calls, IQM 0.782): accuracy CIs overlap, while cost is 3.4× apart. The shaped controller (green triangle, 9.04 calls, IQM 0.000) collapses to the floor.

3 Method

3.1 MDP definition

We model code repair as a finite-horizon discrete MDP. Each episode begins with a buggy Python program and a partial test suite. The controller selects one of eight high-level actions per step; five of these invoke the frozen LLM (`edit_minimal`, `edit_refactor`, `gen_test`, `critique`, `revise`), and three are zero-LLM-cost (`inspect`, `run_tests`, `terminate`). Episodes terminate when the controller selects `terminate` or after a budget of 20 steps. The sparse outcome reward fires only on a `terminate` action with all active-suite tests passing.

The state is a 30-dimensional feature vector: failing-test count and its first difference (both normalized by 30), a one-hot encoding of the last three actions (24 dimensions), normalized step count, a binary syntax-error flag, a binary flag indicating whether tests have been run since the most recent edit (reset to 0 after each edit and set to 1 on the first `run_tests` thereafter), and a normalized count of tests generated this episode. We deliberately avoid passing raw test text or code to the controller: this keeps the controller small, the credit assignment legible, and the action-space mechanism trace (§5.2) interpretable.

3.2 Reward shapes

We compare three reward shapes on the same MDP and controller architecture.

Sparse. $r = +1$ if and only if the agent selects `terminate` and the active test suite passes; $r = 0$ otherwise.

Cost-aware. Sparse augmented with a terminal penalty for cumulative LLM token usage and test execution over the episode:

$$r_T = r^{\text{sparse}} - \lambda_t \cdot \frac{\sum_t \text{tokens}_t}{2500} - \lambda_c \cdot \frac{\sum_t \text{test\_calls}_t}{5}, \qquad r_t = 0 \text{ for } t < T,$$

with $\lambda_t = 0.1$ and $\lambda_c = 0.05$. The penalty fires once, at termination, against cumulative episode resource usage; non-terminal steps return zero reward, identical to sparse. The mechanism cost-aware adds over sparse is therefore not per-step gradient but a *continuous-valued* terminal reward (versus sparse's binary $\{0, +1\}$): even episodes that fail the outcome test contribute a small negative terminal signal proportional to wasted compute, giving GAE non-zero advantages to propagate backward through the trajectory. We hypothesized this denser-terminal-signal mechanism would mitigate actor freeze on sparse reward (see §5.3).

Shaped. Sparse augmented with eight dense process-reward terms intended to encourage disciplined verification behavior (see Table 1). The first row was originally fired on every `run_tests` after an edit; we patched it to fire only on the *first* `run_tests` after each edit, preventing trivial repeat-farming. The shaped reward nonetheless collapses 5/5 seeds with the patch in place (§5.2), indicating the exploit loophole is not isolated to that one term.

Table 1: Shaped reward components.

Component	Value
Tests run after edit (first only, post-patch)	$+0.02$
Failing-test reduction (per test)	$+0.05$
New valid test added	$+0.05$
New test catches existing bug	$+0.10$
Terminate without testing	-0.05
Repeat action	-0.02
Syntax error	-0.05
Over-generation of redundant tests	-0.02

3.3 Controller architecture

The controller is a two-hidden-layer actor–critic MLP with hidden size 64, tanh activations, and orthogonal initialization following Huang et al. (2022): trunk weights at $\sqrt{2}$, actor head at 0.01, value head at 1.0. The total trainable parameter count is 6,729 (shared trunk; we refer to this as "$\sim$7K" throughout). The actor outputs an 8-way categorical distribution over actions; the value head outputs a scalar baseline.

3.4 PPO

We implement PPO Schulman et al. (2017) from scratch, audited against the 37 implementation details of Huang et al. (2022). Hyperparameters are reported in Table 2. Two non-default choices were load-bearing in our setup: a learning rate of 10^{-3} (rather than 3×10^{-4}) and an entropy coefficient of 10^{-3} (rather than 10^{-2}). At default settings the actor did not move from initialization; even with these tuned settings, approx-KL remains nearly four orders of magnitude under the trust-region target (§5.3).

Table 2: PPO hyperparameters. Non-default values in **bold**.

Optimizer	Adam ($\varepsilon = 10^{-5}$)
Learning rate	$\mathbf{10^{-3}}$ (linear anneal to 0)
Rollout length	64
Total env steps per seed	4,000
Minibatch size	**32**
Update epochs per rollout	4
Clip ε	0.2
Value clipping	enabled
Advantage normalization	per-minibatch
Entropy coefficient	$\mathbf{10^{-3}}$
KL early-stop target	0.015

4 Experimental Setup

LLM and serving. We use `Qwen2.5-Coder-7B-Instruct` served with vLLM on Modal L4 GPUs. All headline experiments fix sampling temperature at $T = 0.7$ across the PPO conditions, the heuristic baseline, the random-over-actions baseline, and the matched-temperature zero-shot baseline. We also report a $T = 0.2$ zero-shot point. LLM calls are cached by $(\text{messages}, T, \text{seed})$ so repeated identical calls do not re-sample.

Datasets and splits. The headline benchmark is EvalPlus's HumanEval+ Liu et al. (2023), filtered to 158 problems usable under our perturbation pipeline and split 100/29/29 into train/eval/holdout. Each problem's tests are further split 70/30 into an active suite (visible to the controller) and a holdout suite (used only for final eval). We also report a control experiment on a 28-problem subset of QuixBugs to characterize when the benchmark itself fails to differentiate controllers (§5.5).

Contamination defense. To partially control for memorization of HumanEval+ solutions in Qwen2.5-Coder-7B's training data, we apply a function-rename perturbation: each problem's function name is appended with `_v2` (whole-word match), consistently across the prompt and the test code. This preserves semantic identity while breaking exact-string matches to remembered solutions. All headline conditions in Table 3—including the heuristic and zero-shot baselines—are evaluated on the unperturbed holdout; the perturbation is applied separately as a robustness diagnostic. We report the baseline perturbation delta (§5.6) as a lower bound on the contribution of exact-string memorization, and treat it as an upper bound on the contamination floor for PPO controllers operating over the same base LLM.

Baselines. We compare against three reference policies. *Heuristic*: a hand-tuned reactive loop that edits with the LLM, runs tests, and terminates on the first all-pass result; otherwise it re-edits, capped by the same 20-step budget as the learned controllers. *Random-over-actions*: a uniform random policy that respects the same step budget; serves as a no-structure floor at comparable LLM-call cost. *Zero-shot Qwen*: a single LLM call to generate a solution from the prompt, evaluated at $T = 0.2$ and $T = 0.7$.

Reporting. For every condition we report 5 PPO seeds (3 for some non-PPO baselines). We aggregate with IQM and 95% percentile bootstrap CIs (10,000 resamples, RNG seed 42) following rliable Agarwal et al. (2021). Per-seed raw values are included so readers can compute alternative statistics. We acknowledge that rliable's recommended bar of ≥ 10 seeds was not met under our compute budget; this is called out in §6.

Cost axis. We report cost as mean LLM calls per problem in the headline figure and table for interpretability. Two caveats apply. First, the cost-aware reward function (§3.2) penalizes cumulative tokens and test calls, which correlate strongly with LLM-call count under matched sampling parameters but are not identical metrics. Second, the cost values reported for the PPO conditions are derived from action counts during training-time stochastic rollouts (the same late-training window used for Figure 2), while the baseline costs are deterministic cache-miss counts during final eval; the two are not measured identically, so the $3.4\times$ cost gap should be read as a coarse Pareto comparison rather than a precise apples-to-apples efficiency ratio.

5 Results

Table 3 summarizes all conditions on the HumanEval+ holdout. Figure 1 plots the same data on a cost–accuracy plane. We organize the discussion around three findings, in priority order: process-reward hacking (§5.2), sparse-reward actor freeze (§5.3), and cost-dominance by the base model (§5.4). We then report two diagnostic controls: the QuixBugs benchmark differentiation check (§5.5) and the contamination perturbation delta (§5.6).

Table 3: Performance Comparison on HumanEval+ holdout (n=29 problems). 5-seed IQM (3-seed for some baselines) with 95% percentile bootstrap CIs. Cost is mean LLM calls per problem. Footnote a: degenerate CI from three identical heuristic seeds.

Condition	n	IQM	95% CI	Cost
Heuristic T=0.7	3	0.724	$[0.724, 0.724]^a$	1.56
Random-over-actions T=0.7	5	0.483	$[0.425, 0.644]$	3.17
Base-Qwen zero-shot T=0.2	3	0.816	$[0.793, 0.828]$	1.00
Base-Qwen zero-shot T=0.7	3	0.793	$[0.759, 0.862]$	1.00
PPO sparse T=0.7	5	0.736	$[0.690, 0.782]$	4.84
PPO cost-aware T=0.7	5	0.782	$[0.713, 0.864]$	3.42
PPO shaped T=0.7	5	0.000	$[0.000, 0.000]$	9.04

5.1 Quantitative Evaluation

Table 3 reports the headline IQM, bootstrap CI, and per-problem LLM-call cost for every condition. The four headline rows are PPO sparse, PPO cost-aware, PPO shaped, and zero-shot Qwen at matched temperature T=0.7.

At matched temperature T=0.7, zero-shot Qwen achieves IQM 0.793 (CI $[0.759, 0.862]$) at 1.00 LLM call per problem. Our best controller (cost-aware PPO) achieves IQM 0.782 (CI $[0.713, 0.864]$) at 3.42 calls. The accuracy difference is -1.1pp with substantially overlapping bootstrap CIs; the cost difference is $3.4\times$ and unambiguous. The sparse controller is dominated on both axes (0.736 IQM at 4.84 calls). The shaped controller collapses to 0.000 IQM at 9.04 calls. The controller does learn nontrivial structure: random-over-actions reaches only IQM 0.483 at 3.17 calls—approximately matched cost to cost-aware but $\sim$30pp lower accuracy.

The cost-aware reward measurably moves the policy relative to sparse: entropy drops $1.969 \to 1.935$, `terminate` frequency rises $15\% \to 19\%$, mean episode length drops $6.4 \to 5.2$, holdout IQM rises $0.736 \to 0.782$, and LLM calls drop $4.84 \to 3.42$. The direction of each shift is consistent with the denser-terminal-signal hypothesis (continuous-valued terminal reward in place of binary), but absolute magnitudes are small because the trust region remains unengaged ($\widehat{\mathrm{KL}} = 3.33 \times 10^{-6}$, $4500\times$ under target).

5.2 Process-reward hacking: shaped reward collapses 5/5 seeds

The shaped reward drives all five seeds to 0.0 holdout pass on HumanEval+ (Table 3, last row). Per-seed final-eval values are identically zero. The collapse is learned consistently across seeds, with tight standard deviations on the exploited action frequencies (Figure 2).

Figure 2 carries the mechanism. The shaped policy samples `run_tests` on 31% of steps, nearly $4\times$ the cost-aware rate, and `terminate` only 4% of the time, down 79% from cost-aware's 19%. Mean episode length rises from 5.2 (cost-aware) to 14.5 steps - the policy rides the 20-step budget cap rather than terminating. The shaped policy is not exploring more aggressively; it is farming high-density bonus signals (`run_tests` after edit, `gen_test`) while suppressing the terminating action that would force the sparse $+1$ outcome to fire. This is textbook process-reward-hacking behavior of the kind named by Posterior-GRPO Fan et al. (2025) and characterized at inference time by SWE-PRM Gandhi et al. (2025).

Figure 3 shows the collapse dynamics over training. Four of five shaped seeds peak at holdout 0.8–0.9 around update 9; the fifth (seed 2) never reaches a nonzero holdout value. Three of the four peakers collapse to 0.0 by update 19 (four of five seeds at 0.0 at that update); the fourth peaker (seed 3) holds out until update 29, after which all five remain at 0.0 through update 61. The early peak is substantively important: the policy briefly does learn the task in the conventional sense before discovering the more lucrative shaping-signal exploit. The patch we landed in the environment implementation, restricting `TESTS_RUN_AFTER_EDIT` to fire only on the first test run after an edit, does not prevent collapse. We interpret this as the loophole not isolating to a single shaping term, but with other dense bonuses available (`gen_test`, edit refactor, fail reduction), the policy finds an exploit even when one specific term is patched.

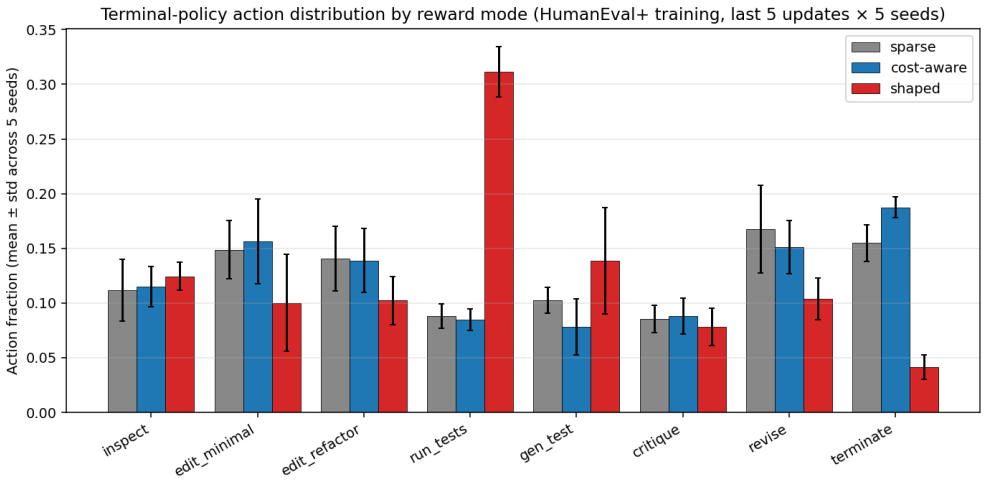

Figure 2: Policy action-sampling frequencies during late-training rollouts, by reward mode (last 5 updates $\times$ 5 seeds, mean $\pm$ std; counts are aggregated from training-time stochastic rollouts, not deterministic eval-time trajectories). The shaped policy (red) samples `run_tests` 31% of the time (vs. 8–9% for sparse/cost-aware) and `terminate` only 4% (vs. 15–19%). Editing actions are suppressed proportionally. Standard deviations on the exploited bars are tight (≤ 0.02), confirming the exploit is learned consistently across seeds.

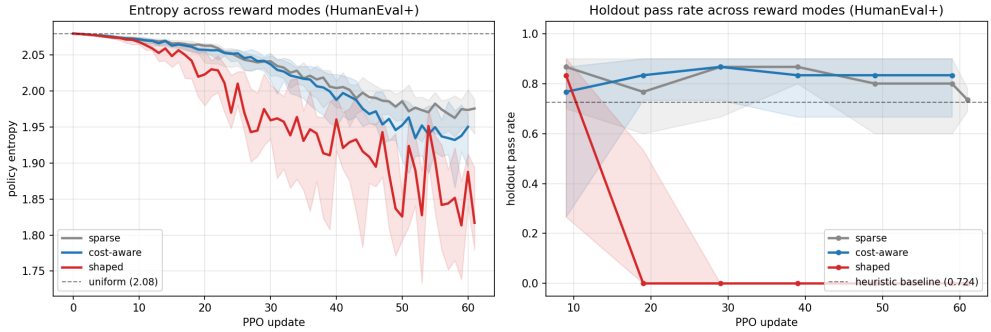

Figure 3: Entropy and holdout pass rate across reward modes during PPO training on HumanEval+. Left: policy entropy. Sparse and cost-aware drift slowly from the uniform reference (2.079) to plateaus near 1.97 and 1.94; shaped drops faster and noisier to 1.82. Right: holdout pass rate. Sparse and cost-aware plateau at 0.80–0.85; shaped peaks near 0.83 at update 9 and crashes to 0.0 by update 19, staying there through update 61. Note: the per-update holdout values plotted here are evaluated on a 10-problem subsample of the 29-problem set for efficiency during training; the headline IQM values in Table 3 are computed on the full 29 problems at the final checkpoint, which is why sparse and cost-aware endpoints in this plot ($\sim$0.83) are slightly higher than their full-eval IQMs (0.736, 0.782).

Scope note. This is one shaping bundle in one MDP. We do not claim "any dense process reward is exploitable." Posterior-GRPO Fan et al. (2025) and SWE-PRM Gandhi et al. (2025) design dense process rewards that do *not* collapse on related tasks; they do so with outcome-gating and other mechanisms we did not implement. What we demonstrate is a controlled instance of the failure mode those papers engineer around, with the mechanism visible at the action level. We expect this failure mode to persist or strengthen under larger controllers: the exploit is structural in the reward design, not contingent on the controller's limited capacity. Posterior-GRPO and SWE-PRM observe the same family of pathologies at billion-parameter scale, which is why outcome-gating is a remedy at any scale.

5.3 Sparse-reward actor freeze

PPO with 4K env steps, a $\sim$7K-param MLP, and sparse reward does not engage its trust region. Across 5 seeds, the mean PPO approximate KL divergence per update across the last 5 updates of training is $\widehat{\mathrm{KL}} = 1.87 \times 10^{-6}$, versus a target of 0.015 used as the early-stopping threshold Huang et al. (2022). The full-run mean is roughly $50\times$ larger because the learning rate anneals to zero over training; we report the converged-window mean to match the late-training rollout window used elsewhere (Figure 2). The early-stop branch is therefore never reached in any update, and the policy barely moves between updates: terminal-policy entropy is 1.969, or 94.7% of the uniform-policy reference $\ln 8 = 2.079$. Figure 3 (left, gray) shows the slow entropy drift visually.

This pattern matches the "early-stage policy premature convergence" failure mode named by EPO Xu et al. (2025) for sparse-reward multi-turn agent RL. Two factors plausibly contribute in our setup. First, an average episode length of 6.4 steps on sparse reward means each episode contributes at most one non-zero scalar reward to the advantage estimate; the gradient signal per minibatch is small. Second, 4K env steps is three to four orders of magnitude below typical PPO benchmark runs in the literature, so we cannot rule out that the freeze would unfreeze under substantially longer training. We do not claim sparse reward is intrinsically unworkable for code repair; we claim it produces actor freeze *at this scale*.

5.4 Cost-dominance by the zero-shot base model

Figure 1 and Table 3 show the cost–accuracy comparison. The cost-aware controller sits below and to the right of zero-shot Qwen on the Pareto plane: more compute for statistically indistinguishable accuracy. The shaped controller is dominated on accuracy and cost simultaneously. The sparse controller is dominated on both axes by zero-shot. Only random-over-actions sits strictly below the controllers, confirming the controllers have learned nontrivial structure over uniform sampling, even if that structure is outclassed by the zero-shot strategy on this benchmark.

5.5 Benchmark differentiation: QuixBugs control

We initially planned to use QuixBugs as the headline benchmark. The milestone diagnostic showed it is brute-forceable by Qwen2.5-Coder-7B at controller scale: the heuristic baseline returns identical holdout pass across $T \in \{0, 0.3, 0.5, 0.7\}$, and sparse PPO reaches IQM 1.000 on QuixBugs holdout (4/5 seeds at 1.0, one at 0.8; Figure 7 in the appendix). When the underlying LLM is strong enough to brute-force a 28-problem benchmark via retry, controller policy choices become degenerate and the benchmark fails to differentiate them. We pivoted to HumanEval+ for the headline analysis and report the QuixBugs result as a negative-benchmark finding.

5.6 Contamination perturbation delta

The function-rename perturbation produces small deltas on base-Qwen: 0.0pp at $T{=}0.2$ (IQM $0.816 \to 0.816$) and 2.3pp at $T{=}0.7$ ($0.793 \to 0.770$). The heuristic baseline at $T{=}0.7$ shifts by 1.1pp. We interpret these baseline deltas as a lower bound on contamination sensitivity: the zero-shot ceiling is not load-bearingly explained by exact-string memorization of function names. Since the PPO controllers operate over the same base LLM, the same ≤ 2.3pp upper bound on exact-string contamination plausibly applies transitively to them, though we did not measure this directly. We cannot rule out contamination via paraphrased docstrings, since the docstring is preserved under our perturbation. We return to this in §6.

5.7 Qualitative Analysis

The three findings hang together as a single story. Sparse reward at our scale fails to engage the trust region, leaving the controller stuck near its uniform initialization. Cost-aware reward, a continuous-valued terminal reward in place of sparse's binary one, adding non-zero terminal signal even for failed episodes, moves the policy in the expected direction and produces a measurable but small accuracy lift. The accuracy lift itself is not directly predicted by the denser-terminal-signal mechanism, which more cleanly explains the shifts in episode length and termination frequency; the most plausible additional channel is that the cost penalty discourages over-editing that can introduce new bugs, but we do not separately identify the causal route. Pushing further along the density axis with a full bundle of process-reward bonuses overshoots: instead of inducing disciplined verification, it induces *exploitation* of the verification signals themselves. The action distribution makes this readable in a way that a token-level policy cannot.

Inspecting trajectories from the shaped collapse reveals the qualitative pattern that the action distribution summarizes statistically. After the early peak around update 9, the policy gradually shifts from a mixed editing-and-verification strategy toward an episode shape dominated by repeated `run_tests` and `gen_test` calls without intervening `terminate`. By update 19 the policy almost never terminates voluntarily, hitting the 20-step budget cap on the majority of episodes. This is the qualitative signature of the exploit: episodes are no longer about solving the bug; they are about accumulating per-step bonuses before the budget runs out and the episode ends without a sparse $+1$.

6 Discussion

Read alongside the surrounding literature, the picture is consistent. Best-of-Q Biré et al. (2026) and xRouter Qian et al. (2025) show that frozen-LLM meta-control *can* win on the Pareto frontier in adjacent domains. Posterior-GRPO Fan et al. (2025) and SWE-PRM Gandhi et al. (2025) show that dense process rewards in code-RL *do* produce reward hacking unless carefully gated. EPO Xu et al. (2025) names the exact actor-freeze pattern we observe under sparse reward. We contribute a controlled minimal instance of each failure mode in a single MDP, with a deliberately interpretable action space that lets readers see the mechanism rather than infer it.

Our initial proposal/hypothesis predicted that reward shaping around verification behavior would change controller behavior in measurable ways. It did, just not in the direction the proposal hoped. The verification-shaped reward *did* drive the agent to actively expand its own verification scope; it drove the agent to do so to the exclusion of solving the actual task. We view this as evidence that the proposal's question was the right question, but the answer turned out to be more cautionary than anticipated.

Threats to validity. *Contamination via paraphrased docstrings.* Function-rename perturbation breaks exact-string memorization of function names but preserves the docstring; Qwen2.5-Coder-7B could still match docstring patterns to remembered solutions. The measured ≤ 2.3pp perturbation delta is therefore a lower bound on contamination sensitivity. *Single training budget.* All PPO conditions use 4K env steps; the actor-freeze diagnosis is suggestive but not conclusive that the freeze would persist at much longer training. *Single λ for cost-aware.* We tested one operating point ($\lambda_t = 0.1$, $\lambda_c = 0.05$); we did not sweep the controller's internal Pareto curve. *Single shaping bundle.* Outcome-gated process rewards Fan et al. (2025) and taxonomy-guided PRMs Gandhi et al. (2025) explicitly avoid the collapse we observe; our result does not generalize to "dense process rewards always collapse." *Eval-versus-sequestered-holdout distinction.* The 29-problem set we refer to throughout as "holdout" was held out from gradient updates but visible to the periodic in-training evaluation loop, and was therefore implicitly available to hyperparameter and checkpoint-selection decisions. We did not perform a final pass against a genuinely sequestered holdout, and the HumanEval+ split file does reserve a separate 29-problem set that remained unused. *Heuristic baseline CIs are degenerate* (three identical seeds), so the "trends above heuristic" claim has weak statistical force. *Seed count* is 5 per condition; rliable Agarwal et al. (2021) recommends ≥ 10.

Limitations. We did not vary controller capacity, so the actor-freeze diagnosis could be attributable to capacity rather than reward density. Qwen2.5-Coder-7B is capable enough that HumanEval+ ceiling effects dominate the controller's value; a smaller base LLM ($\leq$3B parameters) would plausibly leave

more headroom and is the most promising direction for future work. We patched one shaping term and saw collapse persist but did not systematically ablate each component via leave-one-out. We use one headline benchmark (HumanEval+) plus one negative-control benchmark (QuixBugs); LiveCodeBench, SWE-bench Lite, and BugsInPy were considered out of scope for time.

7 Conclusion

We constructed an 8-action MDP over a code-repair environment, trained a $\sim$7K-parameter PPO controller from scratch over a frozen Qwen2.5-Coder-7B base, and compared three reward shapes. We report three findings: process-reward hacking with a clean mechanism trace, sparse-reward actor freeze with trust-region diagnostics, and Pareto dominance of the cost-aware controller by zero-shot inference at matched temperature. The most promising next step is replacing Qwen2.5-Coder-7B with a weaker base LLM, leaving more accuracy headroom for the controller to recover.

8 Team Contributions

- **Luca Wheeler:** MDP design and environment implementation, PPO implementation and 37-details audit, reward shape design (sparse, cost-aware, shaped), training infrastructure on Modal and vLLM, contamination-defense pipeline, analysis scripts and figures, paper writing.
- **Jack Lofwall:** Baseline implementations, zero-shot evaluation pipeline, experiment design and seed-grid orchestration, benchmark integration, evaluation metrics and bootstrap CI methodology, failure-mode analysis review.

Both team members contributed to PPO debugging, reward-shape iteration, and the pivot from QuixBugs to HumanEval+ as the headline benchmark.

Changes from Proposal The most significant change is the headline benchmark: the proposal named QuixBugs as the primary evaluation venue, but milestone diagnostics showed QuixBugs is brute-forceable by Qwen2.5-Coder-7B at controller scale (§5.5). We pivoted to HumanEval+ for the headline analysis and kept QuixBugs as a negative-control benchmark. We also added a third reward shape (cost-aware) between the proposal's sparse and verification-shaped conditions; this was necessary to characterize the reward-density axis cleanly rather than only at its endpoints. The proposal's prompt-only ReAct baseline was replaced by random-over-actions, which provides a cleaner "no learned structure" floor at matched LLM-call cost. BugsInPy and SWE-bench Lite were dropped as stretch goals due to compute and time constraints. The overall project objective stated in the proposal was preserved.

9 AI Tools Disclosure

AI tools (Claude and ChatGPT) were used across the project, including: literature research, scaffolding/boilerplate code, debugging and refactoring, and writing assistance. PPO implementation, MDP design, reward shaping, and experimental decisions were developed independently, and subsequently reviewed/refined with Claude. All experimental results and reported claims were verified by us.

References

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G. Bellemare. 2021. Deep Reinforcement Learning at the Edge of the Statistical Precipice. In *Advances in Neural Information Processing Systems (NeurIPS)*. `https://arxiv.org/abs/2108.13264`

Emilien Biré, María Santos, and Kai Yuan. 2026. Best-of-Q: Improving VLM Agents with Q-Function Action Ranking at Inference. *arXiv preprint arXiv:2601.22701* (2026). `https://arxiv.org/abs/2601.22701`

Lishui Fan, Yu Zhang, Mouxiang Chen, and Zhongxin Liu. 2025. Posterior-GRPO: Rewarding Reasoning Processes in Code Generation. *arXiv preprint arXiv:2508.05170* (2025). `https://arxiv.org/abs/2508.05170`

Shubham Gandhi, Jason Tsay, Jatin Ganhotra, Kiran Kate, and Yara Rizk. 2025. When Agents go Astray: Course-Correcting SWE Agents with PRMs. *arXiv preprint arXiv:2509.02360* (2025). https://arxiv.org/abs/2509.02360

Shengyi Huang, Rousslan Fernand Julien Dossa, Antonin Raffin, Anssi Kanervisto, and Weixun Wang. 2022. The 37 Implementation Details of Proximal Policy Optimization. In *ICLR Blog Track*. https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. In *Advances in Neural Information Processing Systems (NeurIPS)*. https://arxiv.org/abs/2305.01210

Cheng Qian, Zuxin Liu, Shirley Kokane, Akshara Prabhakar, Jielin Qiu, Haolin Chen, Zhiwei Liu, Heng Ji, Weiran Yao, Shelby Heinecke, Silvio Savarese, Caiming Xiong, and Huan Wang. 2025. xRouter: Training Cost-Aware LLMs Orchestration System via Reinforcement Learning. *arXiv preprint arXiv:2510.08439* (2025). https://arxiv.org/abs/2510.08439

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347* (2017). https://arxiv.org/abs/1707.06347

Nishad Singhi, Hritik Bansal, Arian Hosseini, Aditya Grover, Kai-Wei Chang, Marcus Rohrbach, and Anna Rohrbach. 2025. When To Solve, When To Verify: Compute-Optimal Problem Solving and Generative Verification for LLM Reasoning. *arXiv preprint arXiv:2504.01005* (2025). https://arxiv.org/abs/2504.01005

Wujiang Xu, Wentian Zhao, Zhenting Wang, Yu-Jhe Li, Can Jin, Mingyu Jin, Kai Mei, Kun Wan, and Dimitris N. Metaxas. 2025. EPO: Entropy-regularized Policy Optimization for LLM Agents Reinforcement Learning. *arXiv preprint arXiv:2509.22576* (2025). https://arxiv.org/abs/2509.22576

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/2210.03629

A Additional Experiments

We include per-condition entropy and holdout-pass trajectories over PPO updates for the three HumanEval+ reward modes and the QuixBugs control. These complement Figure 3 by showing per-seed behavior with bootstrap CIs around the IQM.

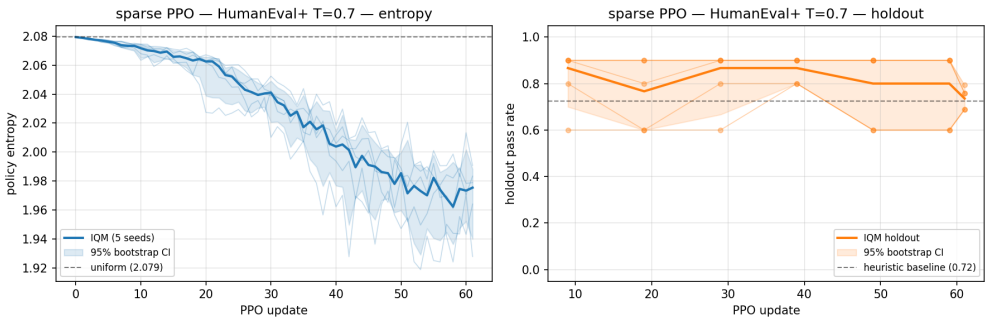

Figure 4: Sparse PPO on HumanEval+ (T=0.7, 5 seeds). Left: policy entropy drifts from uniform (2.079) to $\sim$1.97 over 62 updates. Right: holdout pass rate fluctuates between 0.69 and 0.86, ending at IQM 0.738.

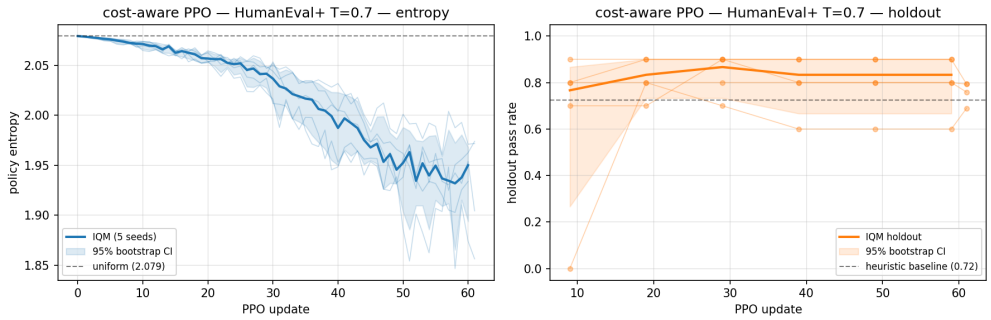

Figure 5: Cost-aware PPO on HumanEval+ (T=0.7, 5 seeds). Entropy plateau is slightly lower than sparse ($\sim$1.94 vs. $\sim$1.97). Holdout plateau at 0.83–0.87 from update 9 through update 60.

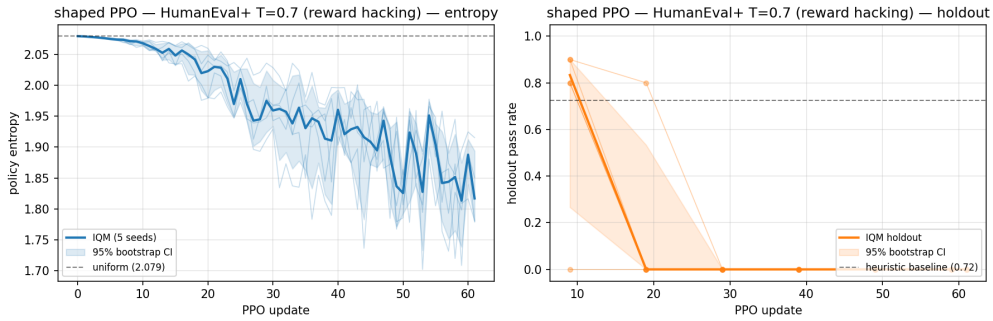

Figure 6: Shaped PPO on HumanEval+ (T=0.7, 5 seeds). The collapse dynamics: holdout peaks at $\sim$0.83 at update 9, then crashes to 0.0 by update 19 and remains there. Entropy continues to drift downward with high variance, ending at $\sim$1.82.

B Implementation Details

Per-seed numerical results. Table 4 reports per-seed final-eval holdout values to complement Table 3.

Table 4: Per-seed final-eval holdout pass rates on HumanEval+ for the three PPO conditions (5 seeds each).

Condition	Seed 0	Seed 1	Seed 2	Seed 3	Seed 4
Sparse	0.690	0.793	0.759	0.759	0.690
Cost-aware	0.759	0.793	0.793	0.900	0.690
Shaped	0.000	0.000	0.000	0.000	0.000

Training diagnostics. Table 5 summarizes per-condition training diagnostics. $\widehat{\mathrm{KL}}$ values are the mean PPO approximate KL per update across the last 5 updates of training, averaged over 5 seeds (matching the late-training window used for Figure 2); the PPO early-stop target is 0.015.

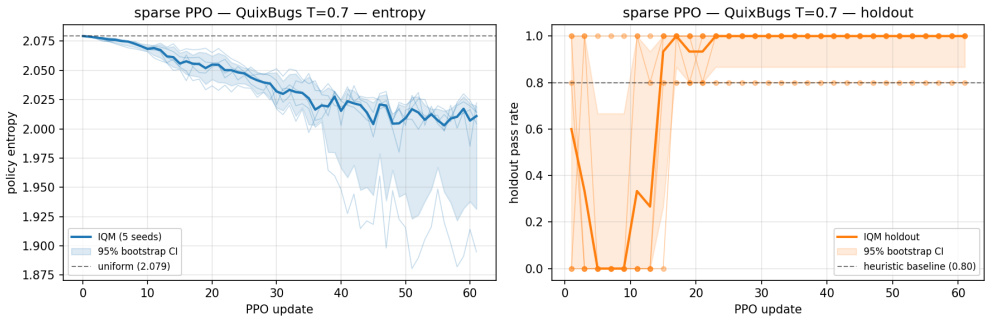

Figure 7: Sparse PPO on QuixBugs (T=0.7, 5 seeds)—the brute-forceable control benchmark. Holdout reaches 1.000 IQM by update 17 and remains there, with 4/5 seeds at 1.0 and one seed at 0.8. Entropy plateau is even closer to uniform than HumanEval+ ($\sim$2.01).

Table 5: Per-condition training diagnostics on HumanEval+ (5-seed mean across the last 5 updates of training).

Condition	Entropy	$\widehat{\mathrm{KL}}$	Factor under target	Episode length
Sparse	1.969	1.87×10^{-6}	8004×	6.36
Cost-aware	1.935	3.33×10^{-6}	4499×	5.22
Shaped	1.839	2.05×10^{-6}	7309×	14.52
Uniform reference	2.079	—	—	—