BEYOND FIRST SUCCESS: EXPANDING ACHIEVEMENT REWARDS IN OPEN-WORLD SURVIVAL GAMES

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon tasks often require repeatedly performing intermediate behaviors that have already succeeded. Rewards for first success alone do not directly reflect subsequent progress through repetition, while continually rewarding repeated behavior can favor easily triggered events. We introduce Structured Ordinal Reward (SOR), which expands achievement feedback along two dimensions: event identity and recurrence level. Using first-success labels and structured reference trajectories, SOR learns event rules, identifies a finite set of observationally supported recurrence levels, and allocates a total auxiliary reward budget according to event recurrence support. The constructed reward program remains fixed during downstream training. Given reference data and approximately one million downstream training steps, SOR improves PPO's cumulative training achievement score in Crafter by 19.5% and its independent native evaluation score in Craftax-full by 37.2%. We further conduct payment-timing controls and budget-permutation experiments within SOR to study how feedback along recurrence and reward allocation across events affect learning. Experiments combining SOR with Achievement Distillation demonstrate compatibility with achievement representation learning, supporting its use as an auxiliary reward module for existing learners. Code is available at https://anonymous.4open.science/r/structured-ordinal-reward-DE30/.

1 INTRODUCTION

Solving long-horizon tasks requires discovering intermediate behaviors and composing them into extended sequences of action (Sutton et al., 1999; Riedmiller et al., 2018; Zhou & Garg, 2023). Progress often depends on reusing these behaviors: resources must be replenished, prerequisites may be consumed, and the same operation may serve several subsequent goals (Hafner, 2022; Matthews et al., 2024). Yet their contribution may only be reflected in a later outcome, making timely credit assignment difficult (Arjona-Medina et al., 2019; Patil et al., 2022).

Prior work supports complex behavior through auxiliary tasks (Riedmiller et al., 2018), encourages exploration with intrinsic rewards (Burda et al., 2019), and provides denser feedback through task-progress signals (Sarukkai et al., 2024). A specific feedback gap remains in tasks that reward the first completion of a milestone: the first success confirms that a behavior has occurred, but does not directly express progress through its subsequent execution, even when repetition is needed for later goals.

A straightforward response is to continue rewarding these behaviors when they recur. However, unrestricted repetition can concentrate learning on easy goals: ELLM observes this when removing its within-episode novelty filter (Du et al., 2023), and DLLM reports a performance decrease when allowing repeated intrinsic rewards for the same goal within a single model rollout (Liu et al., 2025). These findings motivate a distinction between useful recurrence and repetition that is merely easy to reward. The resulting question is how to select feedback targets, limit their rewarded recurrence, and allocate reward so that the feedback supports further task progress. Success supervision can anchor candidate reward events (Fu et al., 2018), and interaction traces can reveal how task execution is structured (Furelos-Blanco et al., 2021). Applied to recurrence feedback, these sources provide event targets and evidence about recurrence frequency and depth. This motivates using reference

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

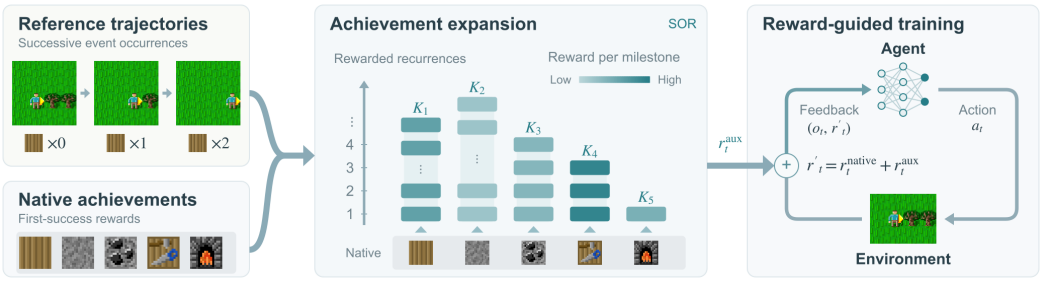


Figure 1: SOR expands first-success achievements along a recurrence axis. Columns denote events, darker cells indicate larger payments, and ellipses omit intermediate levels. Crafter-based schematic.

experience to guide feedback allocation, while placing explicit limits on rewarded recurrence depth and total payment.

We propose *Structured Ordinal Reward* (SOR), which expands achievement feedback from event identity to the two dimensions of event identity and recurrence level (Figure 1). Within a prescribed rule space, SOR learns event detectors from first-achievement labels and structured transition information. It then replays reference trajectories to retain a finite set of observationally supported recurrence levels and allocate a fixed total budget according to event recurrence support. The resulting program remains frozen during downstream training, supplying auxiliary rewards without adding trainable networks to the learner.

We evaluate SOR on Crafter and Craftax, two open-world survival benchmarks (Hafner, 2022; Matthews et al., 2024). Given reference data and approximately one million downstream training steps, aggregates over ten training runs show relative improvements of 19.5% in Crafter's cumulative native achievement score during training and 37.2% in Craftax's independent native evaluation metric over the corresponding PPO baselines. Experiments with Achievement Distillation additionally show that both hand-designed expansion and SOR can provide gains alongside achievement representation learning.

Our contributions are summarized as follows:

- We introduce SOR, which constructs finite recurrence rewards from first-unlock supervision by learning event rules and deriving recurrence levels and budget allocations from reference trajectories.
- Our controls show benefits of distributed payments over the tested single-payment schedules and of support-based allocation over uniform budgets, including at a shared recurrence depth.
- We demonstrate improved cumulative training Score in Crafter and native-only checkpoint performance in Craftax using frozen reward programs.

2 Empirical Motivation and Design Rationale

We first test whether finite achievement expansion improves learning under the original task objective, providing an empirical starting point for automatic reward construction.

Testing extended achievement feedback. Our hand-designed expansions use task knowledge to select useful resource and action milestones. Through manual design and repeated empirical iteration, we retained well-performing programs as strong references. In Crafter, Manual-13 adds thirteen milestones, including holding $2, 4, 6$ units of wood and placing a table for the second time. Craftax's Reachable adds seven resource-inventory milestones. Each milestone pays 0.5 once per episode; native rewards and evaluation achievements remain unchanged. The complete rules and development context are in Appendix E.

In the ten-run, approximately 1M-step Crafter comparison, Manual-13 improves PPO's mean cumulative native achievement score by 30.3% without additional trainable networks (Table 1). The same fixed program also improves the AD reproduction (Section 4.2). The earlier development study and its ablations are retained in Appendix E.

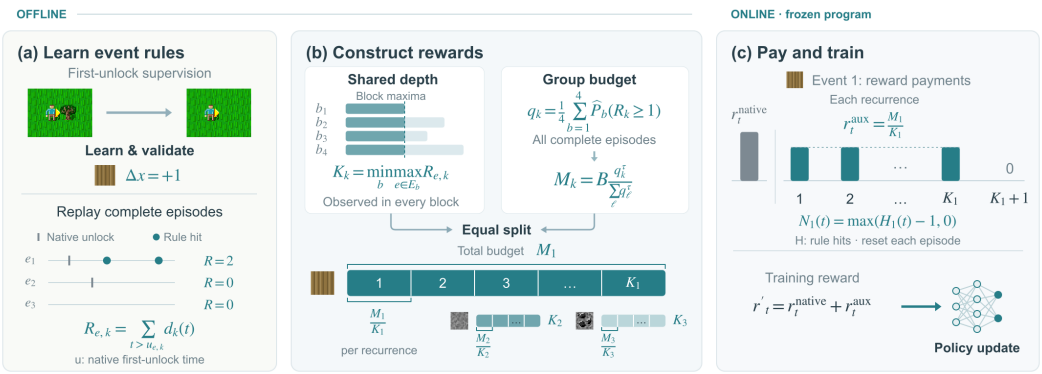


Figure 2: SOR construction and execution in Crafter. (a) Learn rules and count hits after native first unlock; (b) construct levels and budgets; (c) train with frozen rewards, excluding the first rule hit from online counts. Native and auxiliary payments are shown separately; counts and amounts are schematic.

Reward choices affect learning. The benefit depends on the particular expansion. In group ablations, removing resource milestones lowers both final score and Score AUC, whereas removing three survival-related bonuses improves AUC. These comparisons also change total reward budgets (Appendix E). The manual results therefore motivate studying the construction choices behind useful feedback: which events receive rewards, how far their progress is rewarded, and how reward is allocated.

Constructing feedback from existing evidence. First-unlock labels connect events to observed transitions, while reference episodes record their recurrence. We therefore ask whether these sources can guide event selection, rewarded depth, and budget allocation without specifying each added milestone by hand. SOR develops this construction in Section 3.

3 Structured Ordinal Reward

SOR converts first-unlock supervision and reference interaction into a finite set of rewarded recurrence milestones.

Construction uses reference transitions (x_t, a_t, x_{t+1}) with native first-unlock labels. The reward module reads structured state changes and actions; Craftax also uses native first-unlock flags during execution. These structured channels are not added to the downstream policy's pixel observations o_t. Appendix C summarizes the information and counting conventions in both environments. The output is a program of event detectors d_k, ordered count thresholds C_k, and group budgets M_k, frozen before downstream training (Figure 2).

3.1 Bounded expansion beyond first success

Consider a detected event k and its monotone episode count $N_k(t)$. The counting convention determines which occurrences contribute to this progress variable. An ordered set $C_k = \{c_{k1} < \cdots < c_{kK_k}\}$, with $K_k > 0$, specifies the progress levels that receive feedback. Each event–level pair (k, j) is an additional milestone. This expands the achievement vocabulary along a recurrence axis while leaving the training reward scalar.

Assign event k a nonnegative total budget M_k, with $\sum_k M_k = B$. Both evaluated instances divide this budget equally among the retained levels. At each eligible transition, pay for newly crossed levels and add this payment to the native reward:

$$r_t^{\text{aux}} = \sum_k \frac{M_k}{K_k} \sum_{j=1}^{K_k} \mathbf{1}[N_k(t) < c_{kj} \le N_k(t+1)], \qquad r'_t = r_t^{\text{native}} + r_t^{\text{aux}}. \tag{1}$$

Multiple levels provide feedback along recurrence progress, and each level pays at most once per episode. A finite threshold set stops payment beyond the retained range. Normalizing by K_k ensures

that adding levels does not automatically increase the group's total budget: it spreads that budget over more milestones and reduces the payment at each one. Depth and allocation therefore have distinct roles, although both affect the rewards experienced by the learner.

Monotone counts give the undiscounted episode bound $0 \leq \sum_t r_t^{\text{aux}} \leq B$. This limits cumulative auxiliary payment; actual payment still depends on the events reached by the policy. Appendix B gives the equivalent progress-function form and its telescoping sum.

For consecutive recurrence levels $1, \ldots, K_k$, Equation 1 is a capped count bonus. The construction below determines the event rules, the cap for each group, and its share of the budget.

3.2 Constructing events and recurrence levels from reference data

From first-unlock labels to recurrent events. First-unlock labels associate achievements with observed transitions. SOR uses these labels to select reusable rules $d_k(x_t, a_t, x_{t+1}) \in \{0, 1\}$ within a prescribed grammar of single-transition state differences and action tests. Both environments use BIC-based selection and five-fold, episode-disjoint validation, fitting first-unlock positives against preceding transitions from episodes that reach the unlock. Final rules are selected from data; grammar scope, fitting masks, and selection details appear in Appendix C.

Applying a learned rule after the first unlock treats its transition pattern as a reusable event. We assess accuracy on these later occurrences with a separate recurrence audit (Appendix C.1). Crafter's native event counters are available for construction labels and audit truth; the learned-detector executor does not use them to gate auxiliary payments.

Extending feedback to observed recurrence depths. Replay accepted detectors on complete reference episodes. Define $R_{e,k}$ as the number of rule hits strictly after the recorded native first unlock in episode e, or zero if the unlock never occurs. Downstream, Crafter instead counts rule hits without native unlock gates and removes the first hit from its recurrence coordinate:

$$H_k(t) = \sum_{u<t} d_k(x_u, a_u, x_{u+1}), \qquad N_k(t) = \max\{H_k(t) - 1, 0\}. \tag{2}$$

Here $N_k = 1$ denotes the first detected recurrence. Offline labels anchor recurrence statistics to observed first success; online, the detector supplies this boundary itself, so Crafter execution needs no native unlock gate. Crafter partitions reference data into four blocks E_b: two policies, each split into two episode halves. It retains consecutive levels up to the greatest depth observed in every block:

$$K_k = \min_b \max_{e \in E_b} R_{e,k}, \qquad C_k = \{1, \ldots, K_k\}. \tag{3}$$

Groups with $K_k = 0$ are discarded. Each retained depth must be reached by at least one complete episode in every reference block. This observed-coverage rule sets the reward horizon from cross-block experience.

Craftax uses one reference policy split into two episode halves and retains observed count levels supported in both halves. Its exported thresholds start at two and can contain gaps after merging adjacent levels with identical episode coverage. Its runtime counter starts at zero at native first unlock and counts subsequent rule hits. Here K_k is the number of retained thresholds, not the largest count. Equation 1 accommodates both instances, with counts and thresholds expressed in the same environment-specific coordinate. Their frozen programs and execution specifications are distinguished in Appendix C.

3.3 Allocating the reward budget using empirical support

The retained levels determine how far feedback extends; the group budgets determine how much feedback is assigned to each event. A budget assigned to an event rarely encountered by reference policies may supply little feedback. We therefore measure how broadly an event is observed across complete episodes, using each episode as one support observation. In Crafter,

$$q_k = \frac{1}{4} \sum_{b=1}^{4} \frac{1}{|E_b|} \sum_{e \in E_b} \mathbf{1}[R_{e,k} \geq 1]. \tag{4}$$

Every complete reference episode enters the denominator, including episodes without a first unlock or recurrence. Support therefore measures unconditional recurrence under the reference policies, combining accessibility and repetition. Craftax instead uses pooled episode support $q_k = n_k/N_{\text{ref}}$, where n_k is its recorded `episodes_with_detection` count and N_{ref} is the number of complete reference episodes.

For retained groups with positive support, allocate

$$M_k = B\frac{q_k^\tau}{\sum_l q_l^\tau}, \qquad B > 0, \quad \tau \geq 0. \tag{5}$$

This provides a low-dimensional choice between equal group budgets and concentration on events that more reference episodes support: $\tau = 0, 1, 2$ give uniform, proportional, and squared-support allocations. B controls total reward scale, while τ controls concentration: for any two groups, $M_k/M_l = (q_k/q_l)^\tau$. Global parameters are selected separately for each environment during development and then frozen (Section 4.1).

Training with the frozen program. The learner receives its original observations and the augmented reward in Equation 1, with no policy architecture change. Evaluation uses native task metrics and disables auxiliary rewards for independent checkpoints. Algorithm 1 summarizes construction and use; Appendix C specifies episode-boundary handling.

Algorithm 1 SOR: constructing and using a finite recurrence reward program

Require: Reference episodes $\mathcal{D}$, first-unlock labels Y, grammar $\mathcal{G}$, fitting protocol $\mathcal{V}$; block, count, and level conventions; B, τ

Offline construction

$\{d_k\} \leftarrow$ FitAndValidate$(\mathcal{D}, Y, \mathcal{G}, \mathcal{V})$
Count detector hits after recorded native first unlocks in complete episodes
Build block-supported C_k; discard empty groups and set $K_k \leftarrow |C_k|$
Estimate q_k, allocate M_k by Equation 5, and freeze $\{d_k, C_k, M_k\}$

Downstream training

Initialize episode counts using the environment's count convention
for each downstream transition **do**
    Evaluate $d_k(x_t, a_t, x_{t+1})$ and update $N_k(t+1)$
    Pay eligible crossings by Equation 1, respecting terminal convention
    Supply $(o_t, a_t, r'_t, o_{t+1}, \text{done}_t)$ to the learner
    If done_t, reset counts after processing this transition
end for

4 Experiments and Analysis

4.1 Setup

Tasks and measurements. We evaluate PPO (Schulman et al., 2017) in Crafter and Craftax-full (Hafner, 2022; Matthews et al., 2024), using 1,003,520 downstream transitions. Crafter's final native achievement score aggregates recorded native achievement outcomes from completed episodes throughout training. We separately report the last logged native episode return and step-normalized Score AUC. These are training metrics. Craftax-full (v1.6.1, pixel observations) evaluates each final checkpoint on 100 complete stochastic native-only episodes with no adaptation; score is $100\mathbb{E}[R^{\text{native}}]/226$. Its training curve is a separate native-return diagnostic.

Crafter learner. We reproduce the strengthened PPO (ResNet) baseline of Achievement Distillation (Moon et al., 2023), using an IMPALA-style visual encoder. PPO reward-program comparisons share the architecture and optimization settings, with 3.95M trainable parameters. AD adds achievement-learning modules and uses 9.23M trainable parameters (2.34 times the PPO count); reward expansion adds no trainable parameters to either learner (Appendix C.3). Tables 1 and 2 also separate published 1M-step results from our matched experiments. They retain source observation and uncertainty conventions (Appendix D); architecture, compute, and evaluation windows are not matched.

Reward programs and learner integration. We construct and freeze SOR programs using each environment's reference data. Both instances use $B = 3.5$ and divide group budgets equally among retained levels. Crafter uses 12 event groups and 114 levels with $\tau = 2$; Craftax uses seven groups and 86 levels with $\tau = 1$. Environment-specific inputs, count conventions, and support statistics are summarized in Appendix C. We also integrate SOR and the unchanged Manual-13 program with our reproduction of Achievement Distillation (AD) (Moon et al., 2023) in Crafter, retaining AD's 22 native achievement-prediction targets and reusing PPO's frozen SOR program and executor.

Comparisons and statistics. Manual-13 and Reachable are our empirically refined hand-designed references (Section 2; Appendix E). Comparisons with native-reward PPO assess the complete reward intervention. In Crafter, budget permutations and Support–Uniform comparisons share the event rules and structured inputs, with recurrence depths fixed within each comparison, testing allocation under a common information interface.

Main PPO results and learning curves use ten runs per environment. The original four-run Craftax allocation comparison is reported in Appendix H. Crafter mechanism controls use four matched runs; extended Craftax controls use six, and the AD comparison uses three. Our values are means and sample SDs across training runs, averaging evaluation episodes within each run first (Agarwal et al., 2021). Appendix D gives parameter selection, run reuse, seed identities, and metric windows; Appendix C.1 documents execution versions. The 1M-step budget covers downstream training; Appendix C.3 reports available cost information for reference-policy training, collection, and search. Control definitions appear in Appendix C.2.

4.2 Native-task performance

SOR improves native-task learning in both environments (Tables 1 and 2). On Crafter, the frozen SOR program improves mean final training score by 19.5% across ten runs. The paired gain is 2.97 ± 2.60 points and is positive in eight runs. Using the unchanged PPO network, Manual-13 reaches 19.88, numerically comparable to the published 19.40 of the world-model-based Curious Replay agent (Kauvar et al., 2023). SOR is 1.65 ± 1.41 points below this empirically refined reference. Manual-13 uses hand-specified milestones and a maximum auxiliary budget of 6.5, whereas SOR constructs recurrence rewards from reference data with a budget of 3.5 (Section 2).

Agent / reward program	Score	Native return	Score AUC
Baseline work			
EDE (Jiang et al., 2023)	11.70 ± 1.00	—	—
PPO (ResNet) (Moon et al., 2023)	15.60 ± 1.66	—	—
DreamerV3 (Moon et al., 2023)	14.77 ± 1.42	—	—
Curious Replay (Kauvar et al., 2023)	19.40 ± 1.60	—	—
PPO (ResNet) backbone ($n = 10$)			
PPO	15.26 ± 1.91	10.65 ± 0.73	9.61 ± 0.94
PPO + Manual-13	$\mathbf{19.88 \pm 0.73}$	$\mathbf{11.31 \pm 0.43}$	$\mathbf{12.23 \pm 0.91}$
PPO + SOR	18.23 ± 1.20	11.14 ± 0.49	10.77 ± 0.67
AD backbone ($n = 3$)			
AD (reproduced)	19.14 ± 0.42	12.61 ± 0.14	11.27 ± 0.55
AD + Manual-13	$\mathbf{22.75 \pm 0.85}$	$\mathbf{12.68 \pm 0.13}$	$\mathbf{13.25 \pm 0.71}$
AD + SOR	20.20 ± 2.22	11.85 ± 0.61	12.21 ± 1.09

Table 1: Crafter performance at approximately 1M steps. Scores in the backbone groups are cumulative training metrics (mean ± sample SD); published results retain their reported uncertainties and protocols (Appendix D). Bold marks the highest mean within each experimental backbone.

Across ten Craftax-full runs, SOR improves mean native-only evaluation score over PPO by 37.2%; all ten paired differences are positive. Its observed mean is close to that of the hand-designed Reachable reference: the paired difference is 0.07 ± 1.04 percentage points, with four positive pairs.

Agent / reward program	Reward (%)
Baseline work	
PPO† (Matthews et al., 2024)	2.2
RND† (Matthews et al., 2024)	2.2
PPO (CNN–RNN) (Dedieu et al., 2025)	4.63 ± 0.20
Transformer world model (slow) (Dedieu et al., 2025)	7.20 ± 0.09
PPO backbone ($n = 10$)	
PPO	4.00 ± 0.28
PPO + Reachable	5.41 ± 0.60
PPO + SOR	$\mathbf{5.48 \pm 0.66}$

Table 2: Craftax-full reward percentage ($100\bar{R}/226$) at approximately 1M steps. †: symbolic observations; other rows use pixels. Published uncertainties, where available, are SEM; the PPO group reports sample SD from native-only eval-100. Bold marks the highest mean in this group.

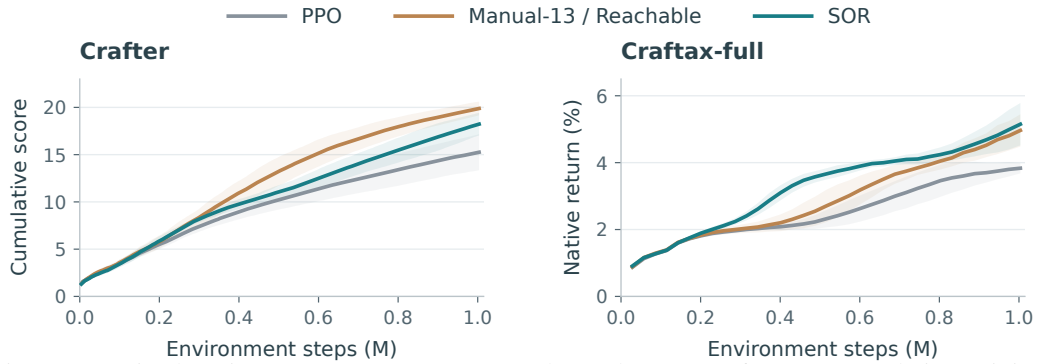


Figure 3: Native learning curves ($n = 10$; mean and sample SD). Crafter uses cumulative training Score; Craftax uses training return (% of maximum), distinct from checkpoint evaluation. Manual-13 / Reachable denotes the hand-designed reference in each environment.

For both main SOR–PPO comparisons, the 95% paired Student-t intervals lie entirely above zero (Table 13). Figure 3 shows the ten-run learning trajectories.

Reward expansion with achievement representation learning. Adding the unchanged Manual-13 program to our AD reproduction raises cumulative Score from 19.14 to 22.75 (18.9%) and Score AUC by 17.5%, with both metrics improving in all three paired runs (Table 1). Thus reward expansion can provide further gains when the learner already models achievements. The frozen SOR program also raises mean Score and AUC, by 5.6% and 8.3%, respectively; AUC improves in all three pairs and Score in two. The last logged native return changes little with Manual-13 and decreases with SOR. The two programs use different predicates and budgets (6.5 for Manual-13 and 3.5 for SOR). Paired statistics and construction variants appear in Appendix G.

4.3 Intermediate capability gains

Aggregate improvements correspond to stronger intermediate capabilities. Figure 4 shows four illustrative resource-and-tool achievements in both environments. In Crafter, stone-pickaxe success rises from 3.8% to 15.3% and stone-sword success from 4.8% to 14.9%. In Craftax, stone-pickaxe and stone-sword success rise from 2.0% to 44.4% and from 2.5% to 49.6%, respectively.

Together, these changes locate the gains in resource acquisition and tool crafting, which provide prerequisites for later progression. Craftax's large error bars for stone tools show that success at this stage remains variable across runs. With non-basic achievements almost entirely unreached by either method, the measured improvement concerns intermediate capabilities within the evaluated budget. The complete profiles, including achievements with lower success rates, are reported in Figure 10 and Table 24.

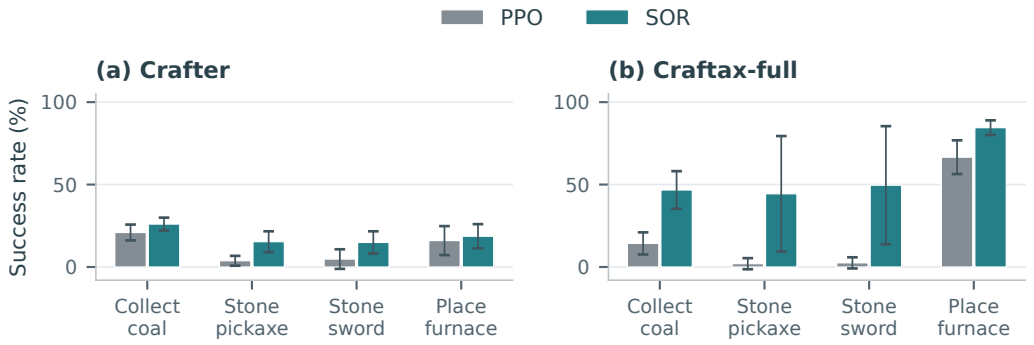


Figure 4: Achievement success rates ($n = 10$; bars: means; whiskers: sample SDs). Crafter reports cumulative training success rates; Craftax reports native-only evaluation rates. Full profiles appear in Appendix G.

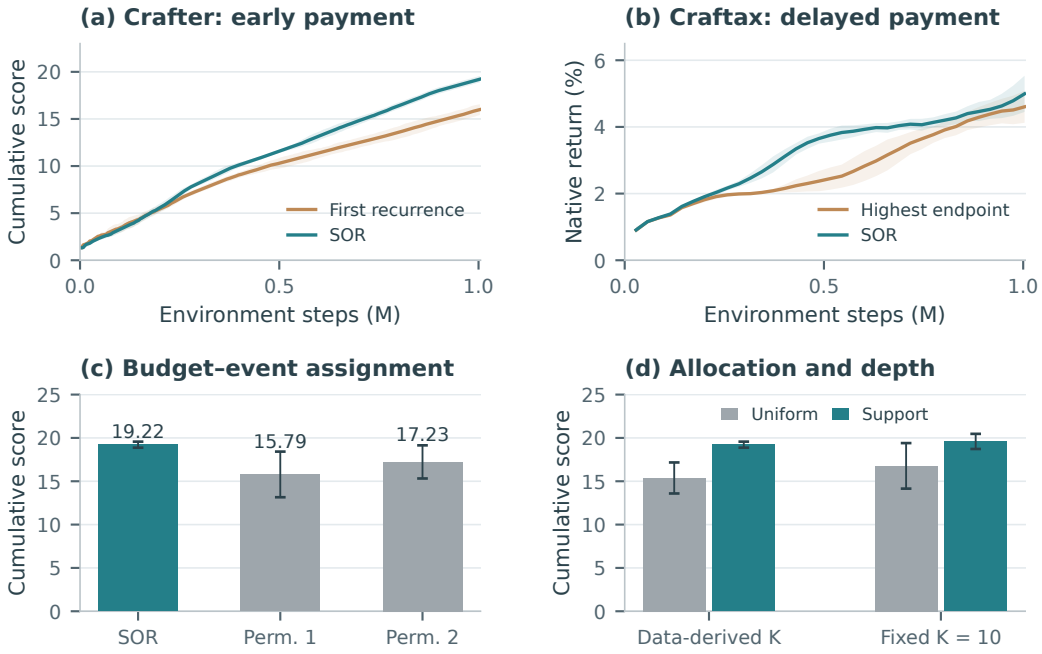


Figure 5: Payment-timing and allocation controls ($n = 4$ matched runs; means and sample SDs). (a,b) Distributed payments versus first-recurrence (Crafter) or endpoint (Craftax) payment. (c) Budget permutations across events. (d) Support-based versus uniform budgets with data-derived or fixed $K = 10$ depths. Bars show cumulative Crafter Score. The SOR reference runs are subsets of the main cohorts.

4.4 Why distribute the reward across recurrence?

These controls ask whether a fixed group budget is more useful when spread across progress or paid at a single milestone. Crafter pays each group's entire budget at its first recurrence, holding event rules and group budgets fixed. Distributing payments instead improves final training score by 3.23 ± 0.66 and AUC by 1.48 ± 0.62, positive in every run (Figure 5a). The single-payment control actually receives more auxiliary reward: 2.103 ± 0.034 per complete episode, versus 1.262 ± 0.034 for SOR. More auxiliary reward alone therefore does not explain SOR's advantage in this comparison.

Craftax tests the opposite extreme, paying only at the highest endpoint. This lowers training native-return AUC in every paired run (Figure 5b). Its independent evaluation score is 4.78 ± 0.76, compared with SOR's 5.41 ± 0.65. These two controls support feedback throughout recurrence over the tested early and late single-payment schedules. Realized exposure also changes with the schedule; Appendix G reports the corresponding payments.

4.5 Allocating the reward budget

Budget permutation tests the correspondence between events and reward amounts: two frozen permutations preserve the budget multiset, total, and concentration, while reassigning budgets to events with rules and depths fixed. They reach 15.79 ± 2.63 and 17.23 ± 1.91 cumulative Score, compared with 19.22 ± 0.35 for SOR (Figure 5c). The lower means support the value of the observed budget–event correspondence over these two alternatives. Permutation can also change per-level payments and realized auxiliary return.

To test whether this advantage requires event-specific depths, we set all groups to $K = 10$ and compare allocations at the same total budget. Support-based budgets yield 19.60 ± 0.87, versus 16.78 ± 2.63 for uniform budgets (Figure 5d). The paired gains are 2.82 ± 2.20 Score and 1.52 ± 0.90 AUC, positive in all four pairs for both metrics. Support-based allocation therefore remains effective at a common depth. The common-depth support variant slightly exceeds the original in mean Score. This supports the allocation principle without establishing an advantage for the data-derived depths over a shared cap.

The Crafter reward-scale controls further test whether increasing total reward can substitute for support-based allocation. Same-executor Uniform at $B = 3.5$ and $B = 5$ remains weaker than SOR, even though the higher-budget control receives more auxiliary reward. Thus neither a larger budget nor greater realized payment alone explains SOR's advantage in these controls. Nearby-budget sensitivity results appear in Appendix G (Figure 7).

In Craftax, SOR outperforms Uniform in the original four-run comparison (Appendix H), but replay calibration narrows the gap, indicating a contribution from reward scale. Increasing concentration from proportional to squared pooled support lowers evaluation score in the later comparison. Stronger concentration therefore does not improve both environments, motivating the separate settings $\tau = 1$ in Craftax and $\tau = 2$ in Crafter. Complete controls and payment accounting are in Appendix G.

4.6 Reliability and sensitivity of construction

An independent Crafter recurrence audit on 262,144 reference transitions not used for fitting gives the declared rules precision 1.000 and recall 0.929. A separate Craftax audit on 100 native-only episodes gives pooled precision 1.000 and recall 0.994 against environment-branch event truth. The audits measure event fidelity; a separate intervention examines the learning effect of changing the trigger stream. Replacing Crafter's learned trigger with native event counts while keeping its reward program fixed lowers mean Score by 2.41 ± 1.94 points across four paired runs. The replacement changes when and how often the fixed program pays: native counts are therefore not a performance upper bound, nor does the result establish a benefit from detector errors (Appendix G).

Changing reference trajectories reproduces the same 12 Crafter rules but changes support and depth, with both improved and reduced mean Scores across reconstructions (Table 19). Budget–depth swaps implicate allocation changes in the weaker reconstruction, without isolating additive effects (Appendix G). Stable event detection therefore does not ensure stable reward allocation or learning outcomes.

5 Conclusion

Structured Ordinal Reward expands achievement feedback beyond first success using first-unlock supervision and reference trajectories. The frozen programs improve PPO's native-task metrics in Crafter and Craftax-full. Controls support intermediate recurrence feedback and support-based budget allocation in the tested settings, while AD integration demonstrates compatibility with achievement representation learning. Together, these results motivate bounded recurrence as a practical dimension of reward construction.

AI USE STATEMENT

The code implementation was largely generated using AI tools. Experimental design and result analysis were primarily carried out and checked by the authors, with AI assistance. Generative AI tools also assisted literature lookup, manuscript planning, drafting, language editing, and evidence-checking utilities. Language-model extraction of event rules is not part of SOR.

ETHICS STATEMENT

This work studies reward construction for reinforcement learning in open-world survival games. All experiments are conducted in the simulated environments Crafter and Craftax, using data generated through agent–environment interactions without human participants or personal data. We have not identified ethical concerns specific to this work that require further discussion. This research received no external funding, and the authors declare no conflicts of interest.

REPRODUCIBILITY STATEMENT

Algorithm 1 and Appendices B–D describe the reward construction interface, execution, and evaluation. Code, example configurations, and frozen reward programs are available at https://anonymous.4open.science/r/structured-ordinal-reward-DE30/. The appendices specify fitting masks and selection criteria, frozen rule expressions, environment-specific execution, training settings, and reference-data costs. Released materials include run-level metrics, recorded learning curves, available configurations, frozen programs, and analysis scripts. The archive documents coverage and unavailable historical records.

REFERENCES

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C. Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems*, volume 34, pp. 29304–29320. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/hash/f514cec81cb148559cf475e7426eed5e-Abstract.html.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/hash/453fadbd8a1a3af50a9df4df899537b5-Abstract.html.

Jose A. Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. RUDDER: Return decomposition for delayed rewards. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://papers.neurips.cc/paper_files/paper/2019/hash/16105fb9cc614fc29e1bda00dab60d41-Abstract.html.

Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/hash/afda332245e2af431fb7b672a68b659d-Abstract.html.

Tristan Bester, Benjamin Rosman, Steven James, and Geraud Nangue Tasse. Counting reward automata: Sample efficient reinforcement learning through the exploitation of reward function structure. In *Workshop on Neuro-Symbolic Learning and Reasoning in the Era of Large Language Models at AAAI*, 2024. URL https://www.raillab.org/publication/bester-2024-counting/.

Richard Bornemann, Pierluigi Vito Amadori, and Antoine Cully. CODE-SHARP: Continuous open-ended discovery and evolution of skills as hierarchical reward programs. *arXiv preprint arXiv:2602.10085*, 2026. URL https://arxiv.org/abs/2602.10085.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1810.12894.

Lior Cohen, Kaixin Wang, Bingyi Kang, Uri Gadot, and Shie Mannor. Simulus: Combining improvements in sample-efficient world model agents. *arXiv preprint arXiv:2502.11537*, 2025. URL https://arxiv.org/abs/2502.11537. Version 4, revised May 2026.

Antoine Dedieu, Joseph Ortiz, Xinghua Lou, Carter Wendelken, J Swaroop Guntupalli, Wolfgang Lehrach, Miguel Lazaro-Gredilla, and Kevin Patrick Murphy. Improving transformer world models for data-efficient RL. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 12944–12968. PMLR, 2025. URL https://proceedings.mlr.press/v267/dedieu25a.html.

Yuqing Du, Olivia Watkins, Zihan Wang, Cedric Colas, Trevor Darrell, Pieter Abbeel, Abhishek Gupta, and Jacob Andreas. Guiding pretraining in reinforcement learning with large language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 8657–8677. PMLR, 2023. URL https://proceedings.mlr.press/v202/du23f.html.

Justin Fu, Avi Singh, Dibya Ghosh, Larry Yang, and Sergey Levine. Variational inverse control with events: A general framework for data-driven reward definition. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/hash/c9319967c038f9b923068dabdf60cfe3-Abstract.html.

Daniel Furelos-Blanco, Mark Law, Anders Jonsson, Krysia Broda, and Alessandra Russo. Induction and exploitation of subgoal automata for reinforcement learning. *Journal of Artificial Intelligence Research*, 70:1031–1116, 2021. doi: 10.1613/jair.1.12372. URL https://arxiv.org/abs/2009.03855.

Danijar Hafner. Benchmarking the spectrum of agent capabilities. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2109.06780.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640(8059):647–653, 2025. doi: 10.1038/s41586-025-08744-2. URL https://www.nature.com/articles/s41586-025-08744-2.

Mikael Henaff, Roberta Raileanu, Minqi Jiang, and Tim Rocktäschel. Exploration via elliptical episodic bonuses. In *Advances in Neural Information Processing Systems*, volume 35, pp. 37631–37646. Curran Associates, Inc., 2022. doi: 10.52202/068431-2728. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/f4f79698d48bdc1a6dec20583724182b-Abstract.html.

Mikael Henaff, Minqi Jiang, and Roberta Raileanu. A study of global and episodic bonuses for exploration in contextual MDPs. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 12972–12999. PMLR, 2023. URL https://proceedings.mlr.press/v202/henaff23a.html.

Yun Hua, Haosheng Chen, Wenhao Li, Bo Jin, Baoxiang Wang, Hongyuan Zha, and Xiangfeng Wang. Reward translation via reward machine in semi-alignable MDPs. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 24912–24931. PMLR, 2025. URL https://proceedings.mlr.press/v267/hua25a.html.

Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2107–2116. PMLR, 2018. URL https://proceedings.mlr.press/v80/icarte18a.html.

Rodrigo Toro Icarte, Ethan Waldie, Toryn Klassen, Rick Valenzano, Margarita Castro, and Sheila McIlraith. Learning reward machines for partially observable reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/hash/532435c44bec236b471a47a88d63513d-Abstract.html.

Yiding Jiang, J. Zico Kolter, and Roberta Raileanu. On the importance of exploration for generalization in reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 12951–12986. Curran Associates, Inc., 2023. doi: 10.52202/075280-0569. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/2a4310c4fd24bd336aa2f64f93cb5d39-Abstract-Conference.html.

Isaac Kauvar, Chris Doyle, Linqi Zhou, and Nick Haber. Curious replay for model-based adaptation. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 16018–16048. PMLR, 2023. URL https://proceedings.mlr.press/v202/kauvar23a.html.

Minjae Kwon, Ingy ElSayed-Aly, and Lu Feng. Adaptive reward design for reinforcement learning. In *Proceedings of the Forty-first Conference on Uncertainty in Artificial Intelligence*, volume 286 of *Proceedings of Machine Learning Research*, pp. 2472–2485. PMLR, 2025. URL https://proceedings.mlr.press/v286/kwon25a.html.

Zeyuan Liu, Ziyu Huan, Xiyao Wang, Jiafei Lyu, Jian Tao, Xiu Li, Furong Huang, and Huazhe Xu. World models with hints of large language models for goal achieving. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 50–72. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-long.3. URL https://aclanthology.org/2025.naacl-long.3/.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/70c26937fbf3d4600b69a129031b66ec-Abstract-Conference.html.

Michael Matthews, Michael Beukman, Benjamin Ellis, Mikayel Samvelyan, Matthew Thomas Jackson, Samuel Coward, and Jakob Nicolaus Foerster. Craftax: A lightning-fast benchmark for open-ended reinforcement learning. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 35104–35137. PMLR, 2024. URL https://proceedings.mlr.press/v235/matthews24a.html.

Michael Matthews, Matthew Jackson, Michael Beukman, Thomas Foster, Alistair Letcher, Scott Fujimoto, Cedric Colas, and Jakob Foerster. Goal-conditioned agents that learn everything all at once. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2605.23551.

Faisal Mohamed, Catherine Ji, Benjamin Eysenbach, and Glen Berseth. Temporal representations for exploration: Learning complex exploratory behavior without extrinsic rewards. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2603.02008.

Seungyong Moon, Junyoung Yeom, Bumsoo Park, and Hyun Oh Song. Discovering hierarchical achievements in reinforcement learning via contrastive learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 63674–63686. Curran Associates, Inc., 2023. doi: 10.52202/075280-2781. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/c919a2b5ec1de69f2629f9119676e336-Abstract.html.

Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the 16th International Conference on Machine Learning*, pp. 278–287, 1999. URL https://ai.stanford.edu/~ang/papers/shaping-icml99.pdf.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 2778–2787. PMLR, 2017. URL https://proceedings.mlr.press/v70/pathak17a.html.

Vihang Patil, Markus Hofmarcher, Marius-Constantin Dinu, Matthias Dorfer, Patrick M. Blies, Johannes Brandstetter, Jose A. Arjona-Medina, and Sepp Hochreiter. Align-RUDDER: Learning from few demonstrations by reward redistribution. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 17531–17572. PMLR, 2022. URL https://proceedings.mlr.press/v162/patil22a.html.

Roberta Raileanu and Tim Rocktäschel. RIDE: Rewarding impact-driven exploration for procedurally-generated environments. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/abs/2002.12292.

Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom van de Wiele, Volodymyr Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing—solving sparse reward tasks from scratch. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4344–4353. PMLR, 2018. URL https://proceedings.mlr.press/v80/riedmiller18a.html.

Max Ruiz Luyten, Antonin Berthon, and Mihaela Van Der Schaar. Strategic planning: A top-down approach to option generation. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 52258–52302. PMLR, 2025. URL https://proceedings.mlr.press/v267/ruiz-luyten25a.html.

Vishnu Sarukkai, Brennan Shacklett, Zander Majercik, Kush Bhatia, Christopher Ré, and Kayvon Fatahalian. Automated rewards via LLM-generated progress functions. *arXiv preprint arXiv:2410.09187*, 2024. URL https://arxiv.org/abs/2410.09187.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211, 1999. doi: 10.1016/S0004-3702(99)00052-1. URL https://www.sciencedirect.com/science/article/pii/S0004370299000521.

Yue Wu, So Yeon Min, Shrimai Prabhumoye, Yonatan Bisk, Ruslan Salakhutdinov, Amos Azaria, Tom M. Mitchell, and Yuanzhi Li. SPRING: Studying papers and reasoning to play games. In *Advances in Neural Information Processing Systems*, volume 36, pp. 22383–22687. Curran Associates, Inc., 2023. doi: 10.52202/075280-0983. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/46c2a9a6f2b2be68682013eb1173c801-Abstract.html.

Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2Reward: Reward shaping with language models for reinforcement learning. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/9941833e8327910ef25daeb9005e4748-Abstract-Conference.html.

Abhay Zala, Jaemin Cho, Han Lin, Jaehong Yoon, and Mohit Bansal. EnvGen: Generating and adapting environments via LLMs for training embodied agents. In *Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2403.12014.

Chen Bo Calvin Zhang, Zhang-Wei Hong, Aldo Pacchiano, and Pulkit Agrawal. ORSO: Accelerating reward design via online reward selection and policy optimization. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/6155cfb4e650d41167848fe2ead30ab8-Abstract-Conference.html.

Tianjun Zhang, Huazhe Xu, Xiaolong Wang, Yi Wu, Kurt Keutzer, Joseph E. Gonzalez, and Yuandong Tian. NovelD: A simple yet effective exploration criterion. In *Advances in Neural Information Processing Systems*, volume 34, pp. 25217–25230. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/hash/d428d070622e0f4363fceae11f4a3576-Abstract.html.

Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On learning intrinsic rewards for policy gradient methods. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://papers.nips.cc/paper_files/paper/2018/hash/51de85ddd068f0bc787691d356176df9-Abstract.html.

Zeyu Zheng, Junhyuk Oh, Matteo Hessel, Zhongwen Xu, Manuel Kroiss, Hado van Hasselt, David Silver, and Satinder Singh. What can learned intrinsic rewards capture? In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 11436–11446. PMLR, 2020. URL https://proceedings.mlr.press/v119/zheng20b.html.

Zihan Zhou and Animesh Garg. Learning achievement structure for structured exploration in domains with sparse reward. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=NDWl9qcUpvy.

CONTENTS

1 Introduction **1**

2 Empirical Motivation and Design Rationale **2**

3 Structured Ordinal Reward **3**

3.1 Bounded expansion beyond first success 3

3.2 Constructing events and recurrence levels from reference data 4

3.3 Allocating the reward budget using empirical support 4

4 Experiments and analysis **5**

4.1 Setup 5

4.2 Native-task performance 6

4.3 Intermediate capability gains 7

4.4 Why distribute the reward across recurrence? 8

4.5 Allocating the reward budget 9

4.6 Reliability and sensitivity of construction 9

5 Conclusion **9**

A Related work **16**

B Reward bound and execution semantics **16**

C Construction specification and estimator differences **17**

C.1 Recurrence audit and execution versions 20

C.2 Control definitions and interpretation 21

C.3 Resource accounting and training specification 21

D Evaluation and statistical details **23**

E Manual expansions, progress objects, and the earlier AD extension **26**

F Development history and unsuccessful alternatives **28**

G Additional results and figure protocols **29**

H Craftax payment and behavioral details **34**

I Literature comparison and generalization scope **36**

J Discussion and limitations **38**

A Related work

Automatic reward construction. ProgressCounts constructs exploration rewards by discretizing LLM-generated progress functions and counting visits to the resulting progress bins (Sarukkai et al., 2024). Strategist turns language-model strategies into progress evaluations and distills them into reward feedback for Crafter (Ruiz Luyten et al., 2025). Other language-based methods synthesize reward code, as in Eureka and Text2Reward (Ma et al., 2024; Xie et al., 2024), or use policy feedback to allocate training among candidate rewards, as in ORSO (Zhang et al., 2025). Other supervision sources also support reward learning: VICE learns event probabilities from desired-outcome examples (Fu et al., 2018), while learned intrinsic rewards use extrinsic learning performance as an optimization signal (Zheng et al., 2018; 2020). SOR constructs a recurrence program from first-unlock labels and reference trajectories, then freezes its rules, levels, and budgets throughout downstream learning. The same program can be reused without online reward adaptation; its allocation reflects the reference distribution. Unlike counting visits to progress bins, its runtime payments reinforce repeated execution of detected events.

Structured rewards and progress feedback. Reward Machines expose temporal reward structure for task decomposition (Icarte et al., 2018). Counting Reward Automata augment finite-state reward models with counters to express rewards that depend on event counts and order (Bester et al., 2024). Learning Reward Machines infers structure from experience given property detectors (Icarte et al., 2019), and ISA induces subgoal automata from high-level event traces (Furelos-Blanco et al., 2021). Adaptive reward design supplies partial-progress feedback for temporal-logic tasks (Kwon et al., 2025). CODE-SHARP uses foundation models to generate hierarchical reward programs for an expanding skill archive (Bornemann et al., 2026). SOR uses an additive recurrence structure: each event has finitely many rewarded levels and a fixed group budget. Its frozen programs admit finite-state encodings by saturating counts at the last rewarded level and tracking episode gates. Our contribution is to construct the event detectors, payment levels, and budgets from first-unlock supervision and reference interaction, and to evaluate the resulting payment and allocation choices.

Intrinsic rewards and exploration. Count-based exploration encourages visits to infrequent states (Bellemare et al., 2016), while ICM and RND use prediction error (Pathak et al., 2017; Burda et al., 2019). RIDE combines changes in learned state representations with count-based modulation (Raileanu & Rocktäschel, 2020), and C-TeC uses temporal representations to derive exploration rewards (Mohamed et al., 2026). EDE instead guides action selection using uncertainty in an ensemble of value distributions (Jiang et al., 2023). SOR rewards finite recurrence of specific events, including previously encountered ones. Its runtime count measures recurrence depth and its reference support determines group budgets; neither is an online state novelty estimate.

Achievement and skill learning. SEA learns achievement structure for exploration, while Achievement Distillation (AD) learns representations through achievement prediction and cross-trajectory matching (Zhou & Garg, 2023; Moon et al., 2023). LEO trains goal-conditioned agents across many goals (Matthews et al., 2026). ELLM and DLLM use language guidance for exploration and goal learning (Du et al., 2023; Liu et al., 2025), whereas EnvGen generates training environments to address an agent's weaknesses (Zala et al., 2024). These methods intervene in representations, policy conditioning, goal selection, or training experience. SOR constructs feedback for the recurrence of detected events. Our AD experiments examine the combination of recurrence rewards and achievement representation learning. Additional background on task reuse, reward transformations, and benchmark protocols appears in Appendix I.

B Reward bound and execution semantics

Manual milestones and SOR admit a common finite-payment interface. For a monotone reached-milestone flag $z_i(t) \in \{0, 1\}$ and allocation $b_i \geq 0$,

$$r_t^{\text{aux}} = \sum_i b_i[z_i(t+1) - z_i(t)], \qquad \sum_t r_t^{\text{aux}} \leq \sum_i b_i. \tag{6}$$

The flags reset at episode boundaries without a reset payment. SOR uses ordered recurrence thresholds for these milestones, with total allocation B.

To express Equation 1 as a progress difference, define

$$\phi_k(n) = \frac{1}{K_k} \sum_{j=1}^{K_k} \mathbf{1}[n \geq c_{kj}]. \tag{7}$$

For monotone counts, a threshold contributes to $\phi_k(N_k(t+1)) - \phi_k(N_k(t))$ exactly when it is newly crossed. Thus the explicit crossing reward and the progress-difference form are identical, including transitions that cross more than one threshold.

Each ϕ_k is nondecreasing into $[0, 1]$. Within one episode, counts are monotone and all M_k are nonnegative. Therefore

$$\sum_{t=0}^{T-1} r_t^{\text{aux}} = \sum_k M_k \sum_{t=0}^{T-1} [\phi_k(N_k(t+1)) - \phi_k(N_k(t))] \tag{8}$$

$$= \sum_k M_k [\phi_k(N_k(T)) - \phi_k(N_k(0))] \leq B. \tag{9}$$

Every term is nonnegative. These equalities apply when all crossings are paid, as in Crafter. If $\eta_t \in \{0, 1\}$ suppresses a terminal payment, as in Craftax, the actual reward is the crossing reward times η_t. Its sum is bounded above by the unmasked telescoping sum, so $0 \leq \sum_t r_t^{\text{aux}} \leq B$ still holds. Counts in this argument are evaluated before the episode reset. This is an undiscounted per-episode bound and does not constrain the number of episodes, their lengths, or which states are visited under a fixed interaction budget. It does not guarantee preservation of the native optimal policy.

For the current Crafter learned-detector executor, initialize rule-hit counts and recurrence counts to zero. For each transition, preserve native reward, increment hit counts for matching rules, and compute $N_k = \max(H_k - 1, 0)$. The first rule hit therefore pays nothing. Subsequent crossings of retained levels pay M_k/K_k once each. Process the terminal transition before resetting bookkeeping for the next episode. Native achievement counters supply construction labels and independent audit truth, not runtime auxiliary reward gates. The existing Craftax executor retains its own frozen count coordinate and threshold lists, specified separately in Appendix C.

For uniform weights and $C_k = \{1, \ldots, K_k\}$, $\phi_k(n) = \min(n, K_k)/K_k$. A per-level paid-flag executor and a progress difference executor are identical when counts, levels, and masses match. An earlier Craftax replay, using a historical mass vector for the seven-group, 86-threshold program, reported zero discrepancy over 4,014,080 transitions and 13,881 episodes, with maximum observed payment 3.4103. Event-detection accuracy is assessed separately in Appendix C.1.

Algorithm 1 gives the shared interface. For construction, $R_{e,k}$ counts detector hits strictly after the recorded native first unlock, assigning zero to episodes without that unlock. This offline boundary differs from the first-hit censoring used online in Equation 2. For a completed episode with an observed unlock, the two counts agree if the first detector hit coincides with that unlock. An earlier false hit or a missed unlock hit can shift the online coordinate relative to the labeled boundary. Crafter uses four policy-by-half blocks and $I_{e,k} = \mathbf{1}[R_{e,k} \geq 1]$. The retained-depth rule is equivalently

$$h_{b,k}(j) = \sum_{e \in E_b} \mathbf{1}[R_{e,k} \geq j], \qquad K_k = \max\{j \geq 1 : h_{b,k}(j) > 0 \text{ for all } b\}.$$

Appendix C specifies the environment-specific counting conventions and construction details.

C Construction specification and estimator differences

Table 3 summarizes the two frozen instances. Both use native first-unlock labels for construction and define reference counts $R_{e,k}$ using rule hits after native first unlock. Downstream policies retain their pixel observations.

Fitting examples and information access. For each target achievement, fitting uses complete episodes in which its native first unlock is observed. The first-unlock transition is positive; preceding transitions are negative. Later transitions and episodes that never unlock the target are excluded

Component	Crafter	Craftax-full
Reward inputs	Inventory/resource changes; actions	Selected state changes; actions; native first-unlock flags
Online count N_k	Rule hits, excluding the first hit	Hits after native first unlock; zero at unlock
Levels C_k	Consecutive $1, \ldots, K_k$	Supported counts from 2, with gaps; $K_k = \lvert C_k \rvert$
References / support	Two policies $\times$ two halves; mean of four block rates	One policy, two halves; pooled episode-support rate
Frozen program	12 groups, 114 levels; $(B, \tau) = (3.5, 2)$	7 groups, 86 levels; $(B, \tau) = (3.5, 1)$

Table 3: SOR instances: environment-specific reward inputs, count conventions, and construction settings.

from fitting. In contrast, support and depth construction include all complete episodes, assigning zero recurrences to episodes without an unlock. First-unlock validation and post-unlock recurrence auditing therefore assess different distributions.

Construction reads actions, before/after numeric non-achievement state, native first-unlock labels, and episode identities. Candidate rules use anonymous feature indices, without semantic names or language-model calls. The Crafter reward executor reads only the selected state differences, action tests, and its own hit history. Craftax additionally reads native unlock bits to gate post-unlock counting. Policies retain pixel observations and receive no direct inventory, count, or reward-program input. AD retains its original latent achievement-state mechanism and native 22-achievement targets.

Candidate grammar and selection. An atom is either an action equality or a nonzero integer state-difference equality. Constants are drawn only from positive training examples within the current fold. A difference is rounded to the nearest integer and accepted when the rounding error is at most 10^{-5}. Clauses contain one to three atoms: at most one action test and two difference tests. A rule is one clause or the disjunction of two distinct clauses. Absolute inventory values and inequality predicates are not used. Atoms are ordered by kind, index, and value; duplicate prediction masks can be compressed during Crafter search.

For example, a clause can recognize a resource-gathering transition from an action test and a unit inventory increase. Such rules identify local transition patterns: they cannot distinguish events with identical action and state-difference features, or directly specify a multi-step sequence such as gathering a resource and later crafting a tool. The episode counter tracks recurrence of a detected pattern separately. These are properties of the candidate grammar; the retained group count is not an estimate of coverage over all possible environment events.

Both environments minimize a BIC score. A candidate partitions N labeled transitions into hits and non-hits, with empirical label probabilities $p_H = TP/(TP + FP)$ and $p_U = FN/(FN + TN)$. Its score is

$$\ell = TP \log p_H + FP \log(1 - p_H) + FN \log p_U + TN \log(1 - p_U), \tag{10}$$

$$\mathrm{BIC} = -2\ell + d \log \max(2, N), \qquad d = 2 + \#\text{atom occurrences} + \#\text{clauses}. \tag{11}$$

Zero-count terms contribute zero. Rules that hit all or no examples, or have $p_H \leq p_U$, are rejected. Ties are resolved by atom count, clause count, then lexicographic clause order. An empty candidate set causes abstention.

Episode-disjoint validation. Five-fold validation generates and selects candidates using four folds, then evaluates the selected rule on the remaining fold. Confusion counts are pooled before computing precision, recall, and F1. Episode identifiers are compact JSON lists: source-policy, environment, and episode indices in Crafter, and environment and episode indices in Craftax. The integer represented by the last eight hexadecimal digits of their SHA256 hash, modulo five, determines the fold; the last two digits modulo two determine the construction half. Accepted rules are refit on all labeled examples. Table 4 gives the acceptance gates. The recorded support tables retain each block's complete-episode denominator and per-level reach counts, allowing recomputation of q_k and K_k.

Gate	Crafter	Craftax
First-unlock episodes	$\geq$ 15 total; $\geq$ 2 per fold; positives from both policies	$\geq$ 15 total; $\geq$ 2 per fold
Pooled validation	F1 $\geq$ .98, precision and recall $\geq$ .95	F1 $\geq$ .75; Wilson precision lower bound $\geq$ .80 and recall lower bound $\geq$ .50
Fold-rule agreement	Jaccard recorded as a diagnostic	Median prediction Jaccard $\geq$ .80
Post-unlock support	$\geq$ 16 hits in $\geq$ 8 episodes; hits in every policy/half block	$\geq$ 16 hits in $\geq$ 8 episodes; hits in both halves

Table 4: Detector acceptance criteria. Craftax Wilson bounds use $z = 1.2815515655$. Agreement compares fold-rule predictions on complete episode transitions.

Group	Selected rule	q_k	K_k	M_k
000	$\Delta\text{coal} = 1$	0.308618	6	0.106450
002	$\Delta\text{drink} = 1$	0.537838	12	0.323300
004	$\Delta\text{sapling} = 1$	0.404095	4	0.182503
005	$\Delta\text{stone} = 1$	0.726068	39	0.589192
006	$\Delta\text{wood} = 1$	0.970157	14	1.051928
013	$\Delta\text{stone_pickaxe} = 1$	0.074858	1	0.006263
015	$\Delta\text{wood_pickaxe} = 1$	0.122115	1	0.016666
016	$\Delta\text{wood_sword} = 1$	0.267949	3	0.080243
017	$\Delta\text{stone} = -4$	0.299234	4	0.100075
018	$\Delta\text{sapling} = -1$	0.330858	3	0.122345
019	$\Delta\text{stone} = -1 \wedge a = 7$	0.641595	23	0.460071
020	$\Delta\text{wood} = -2$	0.642219	4	0.460965
Total			114	3.500000

Table 5: Current Crafter program (repaired original). Every group retains $C_k = \{1, \ldots, K_k\}$ and pays M_k/K_k per detected recurrence up to K_k. Feature names are presentation aliases for anonymous indices.

Frozen programs. Crafter's current program contains 12 automatically retained groups and 114 levels, with $B = 3.5, \tau = 2, \rho = 0$. The detector grammar, reliability gates, recurrence semantics, and parameter family are human design choices. The selected event identities, levels, and detector rules are data-derived. The frozen program specifies all rule expressions, support values, levels, weights, feature indices, and allocations. Table 5 lists its 12 groups. Displayed values are rounded; execution uses the full-precision program. The selected grammar operates on anonymous indices. Field names in the table are readable aliases for the recorded feature order, and do not supply semantic input to the rule search.

For a concrete payment example, Crafter's frozen wood group has $K_k = 14$ and $M_k \approx 1.0519$. Its first rule hit receives no auxiliary payment; each of the next fourteen hits pays approximately 0.0751, after which payment stops.

All selected rules have a single clause. Eleven contain one inventory difference test; group 019 additionally requires an action match. The recorded tolerance for each difference-equality test is 10^{-5}. This describes the selected rules, not the entire candidate grammar. Crafter's concentration parameter was chosen in development and held fixed for the subsequent comparison runs. The repaired original program remains the primary instance; the two additional constructions are reported separately rather than selecting the higher-scoring reconstruction.

For development variants, within-group weights can be written $w_{kj} = \exp(\rho z_{kj}) / \sum_m \exp(\rho z_{km})$, with ordered depth coordinate z_{kj}. The selected $\rho = 0$ gives $1/K_k$, including $K_k = 1$. Reproducing nonzero-ρ variants additionally requires their exact index convention and singleton handling. These variants do not replace the uniform-within-group main instance.

Craftax's construction uses 65,536 passive transitions and 235 complete episodes from one reference policy. The recorded selection contains 13 evaluable achievements, 10 accepted detectors, and seven

groups with recurrence support across its two halves. Pooled five-fold F1 is 1.0 on first-unlock labels; post-unlock recurrence is assessed by the separate audit below. Table 6 specifies the frozen masses.

Event group	Levels	Episode support n_k	Mass
Collect wood	11	231	1.004348
Place table	3	156	0.678261
Collect sapling	3	79	0.343478
Place plant	2	37	0.160870
Collect stone	33	115	0.500000
Place stone	25	102	0.443478
Place furnace	9	85	0.369565
Total	86	805	3.500000

Table 6: Existing Craftax Support program. Masses are displayed rounded; execution uses $3.5n_k/805$. Counts of levels do not replace their exact threshold lists.

The frozen Support threshold export uses count coordinates starting at two. The four short sets are wood $\{2,\dots,12\}$, table and sapling $\{2,3,4\}$, and plant $\{2,3\}$; furnace uses $\{2,\dots,10\}$. The remaining sets are

$$C_{\text{stone}} = \{2,\dots,13,15,\dots,26,29,31,\dots,37,39\},$$
$$C_{\text{place stone}} = \{2,\dots,12,14,\dots,18,20,\dots,25,28,29,31\}.$$

The export's per-threshold `weight` field is the payment M_k/K_k, whereas the normalized within-group share is $1/K_k$. This distinguishes the stored reward values from the notation. Construction counts rule hits strictly after native first unlock. It scans levels from two to the smaller of the two half-wise maximum counts, merging adjacent levels with identical reachable-episode sets and retaining each interval's starting level. This produces the nonconsecutive lists. At runtime, each group starts locked with count zero. Native first unlock opens its gate but contributes no count or auxiliary reward; subsequent rule hits increment the count. Terminal and truncated transitions receive zero auxiliary reward and reset the gate and count. These coordinates differ from Crafter's first-detected-hit censoring.

The seven frozen Craftax rules, in the row order of Table 6, are $\Delta x_0 = 1$, $\Delta x_0 = -2$, $\Delta x_5 = 1$, $\Delta x_5 = -1$, $\Delta x_1 = 1$, $(a = 7) \wedge (\Delta x_1 = -1)$, and $(a = 9) \wedge (\Delta x_1 = -1)$. These use zero-based feature and action indices from the frozen extractor and program. Their corresponding native achievement indices are $0, 1, 3, 7, 9, 10, 16$.

The common-denominator identity explains why the existing Craftax program already serves as proportional unconditional support:

$$\frac{(n_k/N)^\tau}{\sum_l (n_l/N)^\tau} = \frac{n_k^\tau}{\sum_l n_l^\tau}. \tag{12}$$

This identity does not make a pooled estimate equal to Crafter's four-block average under unequal block sizes or policy mixtures. Neither instance estimates support conditional on first unlock. That conditional statistic would change the method and is not a reinterpretation of existing results.

C.1 Recurrence audit and execution versions

First-unlock labels and post-unlock event truth are evaluated separately. The Crafter audit uses 262,144 transitions from newly collected reference trajectories that were not used to fit the frozen detectors. Positive increments in the environment's native cumulative achievement counters provide event truth independently of the learned action/inventory rules. The recurrence confusion matrix includes only transitions on which the corresponding native event had already occurred.

Under the declared conjunctive rules, pooled precision, recall, and F1 are 1.0000, 0.9285, and 0.9629. The historical executor could overwrite a failed action test with a subsequent inventory test in group 019. Its actual deployed precision, recall, and F1 are 0.9908, 0.9285, and 0.9586. On these audit trajectories, payments without a genuine repeated event account for 0.6244% of auxiliary reward

mass. These measurements describe the audited reference distribution, rather than all states visited by reward-trained policies.

The Craftax audit evaluates four frozen SOR checkpoints on 25 complete native-only episodes each, without adaptation. Over 33,145 transitions, environment-branch instrumentation provides event truth independently of the learned detectors. The pooled recurrence counts are 5,802 true positives, zero false positives, and 36 false negatives, giving precision 1.0000 and recall 0.9938. These results cover the sampled checkpoint trajectories; they do not establish correctness under arbitrary policies. The transition-level audit records reproduce these pooled counts when restricted to post-unlock transitions.

The repaired Crafter program canonicalizes atom order to implement the declared conjunction for the selected rules. The main SOR comparisons in PPO and AD, reconstructions, and first-recurrence-payment experiments use conjunctive execution. Earlier concentration/scale controls and AD development experiments retain the earlier executor and are reported separately (Table 9). The current AD integration of SOR uses the same frozen reward program as PPO and retains the native 22-achievement prediction targets. Native and auxiliary rewards are composed before rollout storage, so their sum enters GAE, PPO advantages, and critic targets. AD achievement labels remain native: matching and prediction losses have no direct scalar-reward input, while value distillation inherits the shaped return targets. The same routing applies to Manual-13. Full curves and actual configurations accompany these runs; historical and repaired AD results are reported separately.

C.2 Control definitions and interpretation

Table 7 identifies the intervention made by each control. A fixed maximum budget does not fix realized payment or state visitation. Manual-13 and Reachable are specified in Appendix E; their event sets and progress quantities need not match SOR.

For the equal-grid control, linearly space K_k values between the original minimum and maximum thresholds and round by $\lfloor x + 0.5 \rfloor$. The implementation rejects duplicate levels or changed endpoints instead of silently replacing them. The high/low allocation selects wood and stone using an earlier eight-configuration, seed-455 development study of late return and AUC. Uniform calibration divides Support's replay payment, $17{,}953.870509$, by Uniform's $14{,}957.441919$ on the fixed 4,014,080-transition pool; no online reward total is used to retune this factor.

C.3 Resource accounting and training specification

The reported 1M budget covers downstream training. Reference policies, trajectory collection, fitting, calibration, and parameter search have additional costs. Table 10 reports available quantities for each stage. A constructed program can be reused across policy training runs; construction repeats and audit-only data are separate expenditures.

Reference-policy identities and checkpoint hashes accompany the resource ledger. The Crafter references use training seeds 422 and 423; Craftax uses seed 455. Original/A/B reuse the same Crafter checkpoints and collect new trajectories, so reference-policy training is counted once. Likewise, a frozen construction is reused across downstream seeds.

Learning-network size. Table 11 reports parameter counts from the actual model configurations, with tensor keys and shapes checked against completed checkpoints. Each hand-designed or SOR reward variant retains its learner's network and adds zero trainable neural parameters. Shared weights are counted once; three non-trainable value-normalization statistics are excluded.

Crafter AD has 2.34 times as many trainable parameters as PPO. Its action and next-achievement prediction MLPs contain 2,116,642 and 3,150,848 parameters, respectively; the remaining increase is in the policy and value heads. Action-distribution counts exclude the value head and these auxiliary MLPs, whereas the current `act()` forward pass also computes a value prediction. AD's memory reuses the encoder weights. During auxiliary updates, AD temporarily copies the full model into a non-optimized teacher containing 9,234,228 neural parameters, excluded from the trainable total. Parameter counts describe network size rather than FLOPs or execution time.

Recorded training intervals. For 31 completed Crafter runs, TensorBoard timestamps between the score records at steps 4,096 and 1,003,520 span 244 update intervals. These spans range from 1.23 to

Control	Change relative to the corresponding SOR instance	Retained quantities
Crafter first recurrence	Pay the whole M_k at the first recurrence, then stop	Detectors, M_k, $B = 3.5$
Craftax highest endpoint	Retain only the highest threshold and pay M_k there	Event groups, M_k, $B = 3.5$
Craftax Uniform	Set every group mass to 0.5	Seven groups, count levels, $B = 3.5$
Craftax calibrated Uniform	Scale Uniform by 1.200330284, giving total mass about 4.201	Groups and count levels
Craftax equal grid	Replace retained locations by an endpoint-spanning grid with the same number of levels	Groups, M_k, $B = 3.5$
Craftax high/low	Wood and stone groups each receive $14/13$; the other five each $3.5/13$	Groups, count levels, $B = 3.5$
Craftax squared support	Use $M_k \propto n_k^2$ instead of n_k	Groups, count levels, $B = 3.5$
Crafter historical high-budget Uniform	Use equal group masses at $B = 5$ in the earlier executor	Historical rules and depths
Crafter repaired Uniform	Equal group masses at $B = 3.5$ or 5, with conjunctive execution	Main rules and depths
Crafter budget permutation	Apply either of two frozen permutations to the group-budget vector	Budget multiset, rules and depths; not per-level payments or realized exposure
Crafter common depth	Set every group to $K = 10$; compare support-based and uniform group budgets	Rules, $B = 3.5$, uniform within-group splitting
Crafter nearby budgets	Scale the main allocation to $B = 3$ or 4	Rules, depths, $\tau = 2$, $\rho = 0$
Crafter native-count replacement	Use native cumulative event counts instead of learned rule hits	Twelve groups, 114 levels, M_k, $B = 3.5$
Crafter reconstruction hybrids	Exchange the complete M or K vector between original and B	Detector rules and execution
AD exponent	Set $\tau = 1$ for the main PPO-reference program	Rules, depths, $B = 3.5$, $\rho = 0$
AD-reference construction	Reconstruct from AD trajectories; test $\tau = 1$ and 2	Construction procedure, $B = 3.5$, $\rho = 0$; event set and statistics may change

Table 7: Definitions of reported reward controls. Calibration matches payment on a reference replay pool, not online policy exposure.

Design choice	Available evidence	Supported interpretation
Event detection	First-unlock validation; Crafter and Craftax recurrence audits	Rules recover repeat events on the audited trajectories
Distributed payments	Crafter early lump sum; Craftax highest endpoint	Distributed feedback improves learning over the tested single-payment schedules
Support allocation	Uniform, budget permutations, common-depth allocation, calibration, and M/K swaps	Allocation gains persist at common depth and exceed the tested budget permutations
Finite depth	Cross-block support, equal grid, reconstructed depths, and common $K = 10$	Data-derived depth gives observed coverage; allocation gains also hold at common depth
Learner integration	PPO in both environments; repaired SOR and fixed Manual-13 with AD in Crafter	Reward expansion can complement achievement representations in the matched AD runs

Table 8: Evidence linking the construction choices to learning behavior.

Experiment family	Detector execution	Artifact identifiers
Main Crafter and reconstructions	Declared conjunction, with canonical atom ordering	323, 336
First-recurrence and M/K controls	Explicit conjunction; reported equivalent on the frozen rules	324
Earlier concentration/scale controls	Earlier interpreter with the group-019 action-test overwrite	317, 319
Historical AD integration and search	Earlier interpreter	320, 321
Additional historical AD runs	Earlier interpreter; three completed matched runs	325
Current AD integration and exponent control	Repaired conjunction; same frozen program as PPO	328, 339; native control 325
AD with hand-designed expansion	Unchanged Manual-13 predicates; native 22 AD targets	340; native control 325
AD-reference reconstruction	Repaired conjunction; separately constructed 13-group program	333, 334
Same-executor Uniform controls	Explicit conjunction	326
Permutation, common depth, nearby budgets	Repaired conjunction	332
Native-count replacement	Native cumulative event-counter backend	327

Table 9: Execution versions and experiment provenance. Numeric identifiers locate artifacts; they do not name different scientific methods. The current SOR comparisons in PPO and AD share the repaired execution semantics; Manual-13 uses its own fixed predicates. Historical AD rows remain separate, even when they share training seeds with repaired runs.

1.48 hours for 16 PPO-based runs and from 4.30 to 6.74 hours for 15 AD-based runs. They include intervening logging, checkpointing, and auxiliary updates, but exclude initialization, the first update, and final saving. Runs used different recorded nodes, and historical hardware allocations, sharing, and interruption status are unavailable; the records therefore do not establish a hardware-matched training-time ratio.

The encoder uses group normalization with one group and layer normalization in its dense initialization. Crafter converts pixels to floating point and divides by 255; Craftax uses rendered float32 CHW observations of shape $3 \times 130 \times 110$. Crafter advantages are standardized across the rollout with sample standard deviation and 10^{-8} added to the denominator; its value head uses an EWMA target normalizer. Reported native metrics are computed independently of these optimization transforms. Actual resolved configurations are retained for the completed Crafter reward controls and AD comparisons. Historical Craftax PPO/Reachable records retain runtime manifests, budget bindings, and implementation sources, rather than complete resolved optimizer configurations.

AD uses its native 22-achievement vocabulary, memory, and one hidden layer in the auxiliary heads. Its recorded settings are auxiliary frequency 8, six auxiliary epochs, temperature 0.1, prediction and matching batches of 512, 16 matching samples, optimal-transport regularization 0.05, and similarity threshold 0.5. Policy- and value-distillation coefficients are both 1. The reward routing is described in Appendix C.1. Source snapshots and frozen-program hashes identify the supplied implementation; historical Git commit metadata were not retained.

D Evaluation and Statistical Details

Crafter's achievement score is

$$S = \exp\left(\frac{1}{22}\sum_{i=1}^{22}\log(1+p_i)\right) - 1, \quad p_i \in [0, 100], \tag{13}$$

where each p_i is a native achievement success percentage (Hafner, 2022; Moon et al., 2023). For training metrics, these percentages aggregate completed episodes with recorded native-achievement

Stage	Recorded quantity	Accounting boundary
Reference-policy training	Crafter: two native PPO checkpoints, 1,003,520 steps each; Craftax: one at 1,003,520 steps	Existing checkpoints; Crafter original/A/B share the same two policies
Crafter construction data	131,072 collected steps each for original/A/B; 515/523/516 complete episodes	Complete-episode transitions: 128,637/128,382/129,462; 16 boundary fragments excluded per collection
Craftax construction data	65,536 collected steps; 235 complete episodes, 64,882 complete-episode transitions	Shared pool; eight boundary fragments excluded
Detector fitting	Wall-clock time and device accounting unavailable	Offline compute
Craftax Uniform calibration	4,014,080 historical transitions in its replay pool	Comparator calibration; reused data, not SOR's construction pool
Crafter recurrence audit	262,144 independent reference transitions	Audit data, not detector-fitting data
Craftax recurrence audit	33,145 transitions; 100 complete episodes from four checkpoints	Audit interactions, separate from fitting and main evaluation
AD-reference reconstruction	131,072 collected transitions; 505 complete episodes; 1,006.54 seconds of collection	Two existing AD checkpoints; one source block per policy
Downstream training	1,003,520 transitions per run	Same stated budget per compared run
Craftax final evaluation	100 complete native-only episodes per checkpoint	Evaluation interactions separate from training
Parameter search	Crafter nine-setting scan: 9,031,680 steps; Craftax high/low development: eight 1M runs; AD search retained in full	Historical development costs, separate from each final run

Table 10: Recorded interaction and resource costs. These quantities support comparisons at a fixed downstream budget, not an end-to-end sample-efficiency claim.

Environment / learner	Trainable	Action distribution	`act()` forward
Crafter / PPO	3,948,306	3,947,281	3,948,306
Crafter / AD	9,234,228	3,964,689	3,966,738
Craftax / PPO	9,721,132	9,720,107	9,721,132

Table 11: Unique neural-network parameters. Each row also applies to the corresponding hand-designed and SOR reward variants.

Setting	Value
Visual encoder	IMPALA, channels $[64, 128, 128]$, two blocks, output 256
Hidden size	1024
Parallel environments / rollout length	8 / 512
Rollout batch / logical minibatches	4,096 / 8 (512 samples each)
PPO epochs / outer updates	3 / 245
PPO optimizer steps	5,880; excludes additional AD updates
Discount / GAE parameter	0.95 / 0.65
Craftax SOR microbatch	128, accumulated to the logical minibatch
Crafter optimizer / learning rate	Adam / 3×10^{-4}
Crafter clipping / value coefficient	0.2 / 0.5
Crafter entropy coefficient / gradient norm limit	0.01 / 0.5

Table 12: Recorded learner settings. Shared rollout settings are confirmed by the Crafter configurations and Craftax SOR runtime configuration; the optimizer rows are specific to the resolved Crafter records.

outcomes from the start of training through the logging point. The same cumulative rates determine the reported Score. This geometric aggregate can change differently from average native return. For the main Crafter comparison and repaired mechanism studies, the final score is the cumulative scalar at 1,003,520 steps, rather than a recent episode-window score or an independent final-policy evaluation. This endpoint is slightly beyond an exact 1,000,000-transition cutoff. Curves contain 245 recorded points from 4,096 to 1,003,520 steps. Score AUC uses trapezoidal integration over these points and divides by 999,424, the observed step span; it does not extrapolate an unobserved value at step zero. Tail-28 diagnostics average the final 28 recorded updates and do not replace the final-update primary score. Native episode return is the logged recent-completed-episode mean, kept separate from reward-program payments.

Published comparison protocols. Table 1 takes PPO (ResNet) and DreamerV3 from Table 1 of Moon et al. (2023); DreamerV3 is that study's reproduction. EDE comes from Table 1 of Jiang et al. (2023) (1M steps, ten runs), using a Rainbow-based distributional ensemble and Thompson sampling. We retain its reported uncertainty without assigning an SD/SEM convention not specified in that table. Curious Replay is the DreamerV3 CR row of Table 1 of Kauvar et al. (2023) (ten runs, mean $\pm$ SD); its 1M budget is corroborated by the Crafter benchmark's RL scoreboard and category rules.[1] The AD-study rows also report SD. We omit published native returns because reporting windows differ or the metric is unavailable.

For context, Moon et al. (2023) report an AD Score of 21.79 ± 1.37 over ten runs, whereas our three-run native AD reproduction reaches 19.14 ± 0.42. The AD block in Table 1 therefore reports our matched comparisons; all stated AD reward gains are relative to these local controls, not the published result.

For Craftax-full, Table 2 takes its two published pixel-agent results from Table 3 of Dedieu et al. (2025): their CNN–RNN PPO and best slow transformer-world-model agent. The paper reports reward percentages at 1M steps, with mean and standard error over ten runs as its reporting convention. Appendix E specifies pixel inputs and maximum reward 226. Its exact evaluation window is not established as equivalent to our final-checkpoint eval-100. Two additional baselines come from the official Craftax-1M scoreboard: PPO and RND, both at 2.2%. [2] Both use symbolic observations in the original benchmark experiments (Matthews et al., 2024). We mark them separately and retain the scoreboard's precision without adding unreported uncertainty. The cross-paper values contextualize performance, without supporting paired inference, matched-compute claims, or a ranking of reward mechanisms. They do not replace direct comparisons of auxiliary rewards under the same learner and information budget.

Parameter selection and run reuse. The main Crafter PPO comparison uses seeds 422–431, combining earlier PPO/Manual records with repaired SOR runs and the ten-seed extension. The concentration scan used seed 424 to select the configuration (Appendix F); the main comparison includes its repaired SOR run. Crafter mechanism controls use seeds 422, 423, 425, and 426. Craftax's main comparison uses seeds 455–464: seeds 455–460 were used in earlier configuration comparisons, and 461–464 formed the subsequent four-run block. The extended Craftax controls use 461–466, and the squared-support comparison uses 711–714. Each environment's main result reports all ten runs of its fixed final configuration, including reused runs. Current AD comparisons use 425–427; the earlier AD search used 422–424. Historical and repaired execution variants are reported separately. The original SOR construction is retained as the main instance alongside both reconstruction A and B results.

The earlier recipe study reports a last-28-update score, and the historical budget scan includes a last-20-update statistic. Those diagnostics are not interchangeable. The main learning curves use the ten-run cohorts of their corresponding tables without additional display smoothing. Five-point trailing averages appear only in archived Crafter figures whose captions explicitly identify that convention. No smoothing is used to compute tabulated final scores or AUC.

Craftax evaluates 100 complete stochastic native-only episodes per final checkpoint, with zero adaptation and score $100\bar{R}/226$. Evaluation episode seeds match within the later paired training-seed

[1] https://github.com/danijar/crafter#scoreboards

[2] https://github.com/MichaelTMatthews/Craftax#-scoreboard

block. The main curves combine all ten training runs and use complete-episode native returns over the source trailing 28-update window, distinct from final-checkpoint evaluation.

In Figure 5a,b, each single-payment curve is a separately trained matched control; the SOR curve reuses the corresponding four runs from Figure 3.

Training seed is the unit for paired analysis; evaluation episodes are averaged within each run before comparing methods. Dispersions for our experiments are sample SDs; published comparisons retain their source conventions. Table 13 reports two-sided 95% paired Student-t intervals for the main comparisons. These intervals are descriptive, assume independent run-level differences, and are unadjusted for multiple comparisons or development selection.

Paired contrast	Runs	Mean $\pm$ SD	95% interval
Crafter: SOR $-$ PPO	10	2.97 ± 2.60	$[1.115, 4.832]$
Crafter: SOR $-$ Manual-13	10	-1.65 ± 1.41	$[-2.664, -0.644]$
Craftax: SOR $-$ PPO	10	1.49 ± 0.66	$[1.012, 1.959]$
Craftax: SOR $-$ Reachable	10	0.07 ± 1.04	$[-0.672, 0.820]$
AD: SOR $-$ native, Score	3	1.06 ± 2.38	$[-4.857, 6.984]$
AD: SOR $-$ native, AUC	3	0.94 ± 0.58	$[-0.507, 2.389]$
AD: Manual-13 $-$ native, Score	3	3.61 ± 1.16	$[0.730, 6.489]$
AD: Manual-13 $-$ native, AUC	3	1.97 ± 0.16	$[1.566, 2.383]$
AD: Manual-13 $-$ SOR, Score	3	2.55 ± 1.48	$[-1.119, 6.210]$
AD: Manual-13 $-$ SOR, AUC	3	1.03 ± 0.42	$[-0.015, 2.082]$

Table 13: Paired Student-t intervals across training runs. Crafter reports cumulative training Score, Craftax reports independent native evaluation percentage points, and AD reports the indicated training metric. Interval assumptions and the selection protocol are given in Appendix D.

For supplementary controls, in the six-seed mechanism block Support minus calibrated Uniform is 0.289 ± 1.183 with interval $[-0.952, 1.531]$. In the four-seed later development block Support minus High/low is 0.761 ± 0.214 with interval $[0.420, 1.102]$, while Support minus Grid is 0.580 ± 0.563 with interval $[-0.316, 1.475]$.

The inventory comparison's predeclared practical gate required at least four of six positive score differences and a mean of at least 0.5. Its 0.988 ± 1.767 gain and five positive pairs pass that gate, but its interval $[-0.867, 2.843]$ includes zero. Craftax's original Support–Reachable gate required a mean difference of at least -0.5 and at least two positive pairs; only one pair was positive. Neither gate is a formal non-inferiority test. Rounded table differences can differ in the final digit from paired differences computed before rounding.

E Manual expansions, progress objects, and the earlier AD extension

Our hand-designed programs draw on environment knowledge and repeated empirical iteration. We retained Manual-13 and Reachable as strong fixed references from this development process. Their individual milestones were specified manually, whereas SOR derives its final detectors and levels from reference data within a prescribed construction procedure. This difference in design history provides context for the comparison; the experiments do not isolate the contribution of manual refinement.

Table 14 lists both complete frozen programs. Manual-13 contains thirteen milestones and a maximum auxiliary reward of 6.5; Reachable contains seven milestones and a maximum of 3.5. Each milestone pays 0.5 once per episode. Inventory thresholds refer to the quantity currently held; once a threshold has paid, subsequent consumption does not revoke that payment. The remaining six conditions use the environment's cumulative native event counts. All thirteen Manual-13 conditions reset between episodes. Reachable uses the episode running maximum of inventory to record whether each of its seven thresholds has been crossed; its flags also reset between episodes. Manual-13's event coverage and budget differ from SOR's. Craftax Reachable and SOR both have a maximum budget of 3.5, but their event sets and triggering conditions differ. They are internal references with distinct reward-design choices.

Progress quantity or event	Added thresholds	Milestones
Crafter: Manual-13		
Wood inventory	$2, 4, 6$	3
Stone inventory	$2, 4$	2
Coal and iron inventory	2 of each resource	2
Drink, plant, eat cow	Second occurrence of each event	3
Defeat zombie or skeleton	Second occurrence of each event	2
Place table	Second occurrence	1
Total	Maximum auxiliary reward: 6.5	13
Craftax-full: Reachable		
Wood inventory	$2, 4, 6$	3
Stone inventory	$2, 4$	2
Coal inventory	$1, 2$	2
Total	Maximum auxiliary reward: 3.5	7

Table 14: Complete hand-designed milestone programs in both environments. Each individual threshold or event condition pays 0.5 once per episode. Native rewards and evaluation achievements remain unchanged. The later iron-inventory threshold at three is not part of Manual-13.

Program	Final score	Score AUC
Native PPO	15.635 ± 1.744	9.814 ± 0.861
Repeat Endpoint	18.140 ± 1.568	11.257 ± 0.932
Repeat Ladder	18.605 ± 1.101	11.651 ± 0.510
AGMD-7	19.391 ± 0.929	12.026 ± 0.557
Manual-13	19.439 ± 0.775	11.513 ± 0.744

Table 15: Historical Crafter five-seed replication and four-resource payment comparison, seeds 422–426. Training metrics retain their original experiment identities; they are not the ten-seed table in the main text.

Manual-13's paired gains over PPO are 3.803 score and 1.699 AUC points, both positive in all five seeds. The historical 24.3% improvement is the relative difference between the mean final training scores in Table 15; it is neither an average of seed-wise percentage gains nor an independent checkpoint evaluation. Section 2 instead reports the 30.3% gain in the expanded ten-run cohort of Table 1. The historical replication used eight parallel environments and 1,003,520 transitions per training run. In separate three-seed, approximately 800k-step group ablations, resource removal lowers AUC by 1.425 and survival-group removal raises it by 1.412. Removing groups also changes the reward budget, so these comparisons do not isolate effects at equal reward mass. Member-level effects within a group are mixed, and do not justify a general claim that survival rewards are harmful. In a follow-up study with three different training seeds and the same approximately 800k-step budget, the survival-pruned Manual-10 improves AUC over Manual-13 by 0.5682, with all three paired differences positive. Its mean final-score improvement is 0.1461. This replicates the learning-efficiency direction under the tested pruning, while retaining the difference in auxiliary reward budgets.

The six-seed inventory study (427–432) yields 19.430 ± 1.284 for inventory mastery versus 18.442 ± 0.683 for cumulative repetition. Its paired AUC gain is 0.499 ± 1.422 (four positive pairs), and final native-return gain is 0.287 ± 0.700 (four positive pairs). These are training-log quantities, not independent checkpoint evaluations. The comparison changes the progress object and its realized triggering, so it does not isolate a downstream mediation mechanism.

AGMD-7 is our earlier compact program. The wider historical AGMD SetDepth method used source templates and relational descriptors for target-channel matching, unlike SOR's target-data recurrence construction. Its formal AD extension provides historical cross-learner evidence only:

Score and AUC differences are positive in all three pairs, but native return is positive in only one. An initial failed seed-422 treatment attempt was excluded as an engineering failure and retained diagnostically. AD extracts achievement-related information from rewards. The release must therefore

Metric	AD	AD + AGMD-7	Difference
Native score	21.431 ± 0.819	23.250 ± 1.002	1.819 ± 1.301
Score AUC	12.181 ± 0.463	13.916 ± 1.110	1.735 ± 1.536
Tail native return	12.349 ± 0.396	12.261 ± 0.249	-0.088 ± 0.637

Table 16: Earlier AD extension in Crafter, seeds 422–424, approximately 1M transitions. Difference SDs are paired across three training seeds.

document native reward, shaped PPO reward, and AD event-label routing separately. This experiment does not establish SOR-plus-AD results.

F Development history and unsuccessful alternatives

Crafter's earlier automatic ten-group, 102-level instance reached 17.727 ± 2.886 score over five seeds; a quantile-support variant reached 17.463 ± 2.318 over three seeds. These outcomes motivated further development. They are preserved as failures of those instances, not used to characterize the later frozen 12-group, 114-level construction.

The latest development scan on seed 424 selected $B = 3.5, \tau = 2, \rho = 0$ before its four additional runs. Table 17 records all nine reported configurations, including the budget interaction: at $B = 5$, squared support is much weaker than at $B = 3.5$. Thus, increasing concentration or budget is not monotonically beneficial across the search. The exact numerical early/late ρ settings require the configuration artifact and are not reconstructed from their labels.

B	τ	ρ	Final score	Last-20 score	AUC
3.5	0	0	15.770	15.435	10.261
3.5	1	0	18.491	18.036	10.832
3.5	2	0	20.444	20.127	12.533
4.25	1	early	13.763	13.474	9.405
4.25	1	0	14.059	13.575	8.747
4.25	1	late	13.061	12.664	8.374
5.0	0	0	15.210	14.576	9.477
5.0	1	0	18.391	17.965	10.845
5.0	2	0	15.626	15.116	9.786

Table 17: Reported Crafter development scan, seed 424. This seed is excluded from the original four-run controls but included in the ten-run main comparison with a repaired SOR run.

Combining the additional runs with development seed 424 gives score/AUC $16.120 \pm 1.439/9.990 \pm 0.651$ for $\tau = 0$, $18.019 \pm 0.567/10.690 \pm 0.252$ for $\tau = 1$, and $19.482 \pm 0.936/11.762 \pm 0.559$ for $\tau = 2$. These are descriptive combinations of development and additional runs. The repaired main result is reported separately in Tables 1 and 2; it does not retroactively replace a row of this historical controlled comparison. The completed four-run high-budget challenge has final score/AUC $16.121 \pm 2.992/10.048 \pm 1.265$ for Uniform at $B = 5$, versus $19.242 \pm 0.885/11.569 \pm 0.411$ for squared support at $B = 3.5$. Over complete episodes across the full training span, mean auxiliary payments are 1.408 ± 0.045 and 1.074 ± 0.170, respectively. Earlier reported payments 0.884/1.050/1.296 use a different summary window and are not substituted for these full-span ledger values. This challenge retains the historical executor and does not strictly match online reward exposure.

A separate four-seed recipe-readiness development study tested independent progress, historical joint progress, and simultaneous readiness. Their last-28-update scores were 13.762 ± 0.999, 13.858 ± 1.457, and 15.818 ± 1.627, compared with 19.296 ± 1.689 for inventory I7 and 19.992 ± 1.174 for ManualRef. These candidates did not meet promotion criteria and are not additional successful SOR variants.

Crafter / PPO backbone ($n = 4$, repaired SOR)		
Method	Final score	Score AUC
Native PPO	15.40 ± 1.92	9.57 ± 0.76
Manual-13	$\mathbf{19.57 \pm 0.83}$	11.58 ± 0.84
AGMD-7	19.50 ± 1.04	$\mathbf{12.14 \pm 0.57}$
SOR	19.22 ± 0.35	11.22 ± 0.24
Craftax-full / PPO backbone ($n = 4$)		
Method	Native eval score (%)	
Native PPO	3.89 ± 0.36	
AGMD	5.22 ± 0.65	
Reachable	$\mathbf{5.59 \pm 0.70}$	
Uniform	4.60 ± 0.22	
SOR	5.41 ± 0.65	
Crafter / AD backbone ($n = 3$, historical deployment)		
Method	Final score	Score AUC
AD	21.43 ± 0.82	12.18 ± 0.46
AD + AGMD-7	$\mathbf{23.25 \pm 1.00}$	$\mathbf{13.92 \pm 1.11}$
AD + **SOR**	22.24 ± 0.60	12.82 ± 0.40

Table 18: Additional comparisons with historical internal reward programs. Values are mean $\pm$ sample SD across training runs. Crafter uses native training score and step-normalized AUC; Craftax uses independent native-only evaluation (100 episodes per run). AD results use the historical SOR executor. Manual-13, AGMD, and Reachable are internal reward-program references. Bold marks the highest observed mean within a block, without a significance claim.

Program	Levels	Final score	Score AUC
Original (O)	114	19.22 ± 0.35	11.22 ± 0.24
Reconstruction A	119	19.94 ± 1.35	12.07 ± 1.05
Reconstruction B	126	17.82 ± 1.73	10.61 ± 0.70
$M_{\mathrm{O}}, K_{\mathrm{B}}$	126	19.18 ± 0.77	11.38 ± 0.44
$M_{\mathrm{B}}, K_{\mathrm{O}}$	114	17.14 ± 1.87	10.33 ± 1.06

Table 19: Crafter construction repeats and allocation/depth diagnostics ($n = 4$). O denotes the original construction, A and B new reference-data constructions. Hybrid rows swap group budgets M and recurrence depths K; detector rules are unchanged. All rows use repaired execution.

G Additional results and figure protocols

Historical internal references. AGMD is a separate project reward-construction method. Its completed comparisons are preserved here rather than included in the main narrative. On the shared Crafter PPO block, AGMD-7 has higher mean AUC than repaired SOR; AD plus AGMD-7 also exceeds the historical AD plus SOR instance. These outcomes remain part of the evidence and are not displaced by the main-table selection.

Native return and nearby reward budgets. With the same conjunctive executor as main SOR, Uniform controls at $B = 3.5$ and $B = 5$ reach final cumulative Scores of 15.38 ± 1.79 and 14.16 ± 1.22, respectively. These are distinct from the historical high-budget challenge in Appendix F.

Figure 6 retains Crafter's native training return alongside the cumulative Score shown in the main text. Holding rules, depths, and $\tau = 2$ fixed, budgets $B = 3$ and $B = 4$ yield cumulative Scores 18.98 ± 1.50 and 18.46 ± 0.59, and AUCs 11.27 ± 0.85 and 10.86 ± 0.43, respectively. Figure 7 compares these with the reused $B = 3.5$ instance. These two nearby settings do not improve mean Score over the original; they test local sensitivity, not robustness over an unrestricted budget range. All use the same four training seeds, with no additional control training.

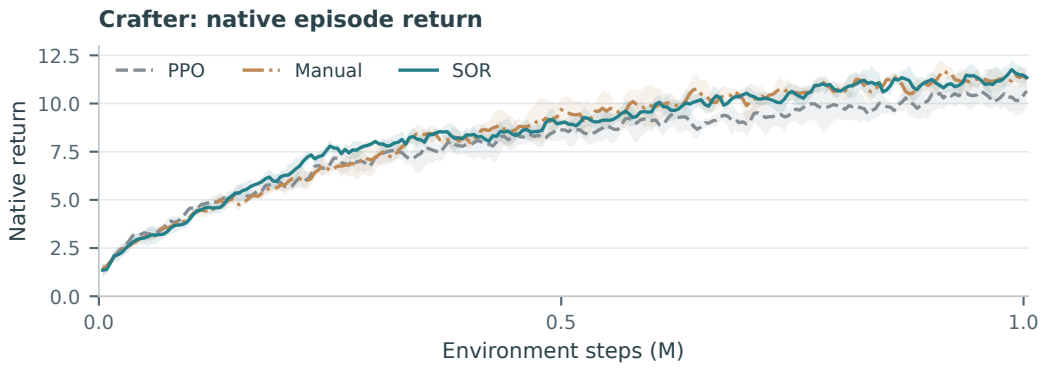


Figure 6: Crafter native episode return for the original four-run control subset of Figure 3 (mean ± sample SD). We plot the logged training statistic without additional smoothing; this is not independent checkpoint evaluation.

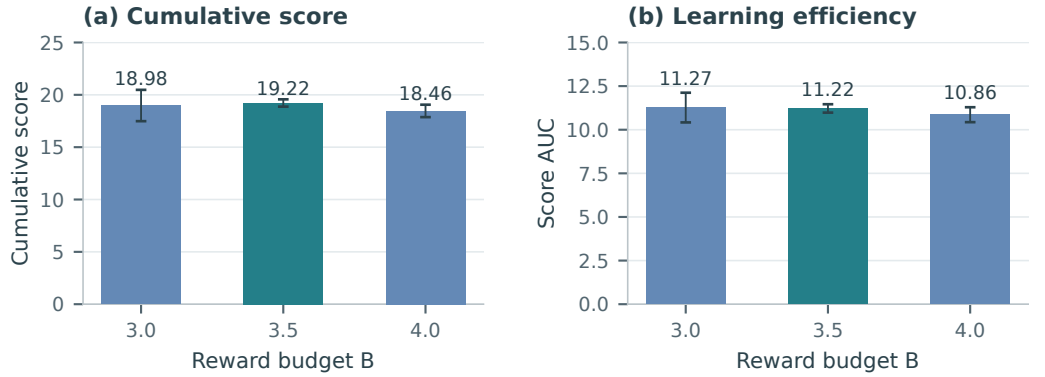


Figure 7: Crafter budget neighborhood with the original rules, depths, and squared-support allocation fixed. Values are mean ± sample SD over four matched runs. $B = 3.5$ reuses the main SOR instance; bars start at zero. All configurations use 1,003,520 downstream transitions.

New mechanism and extension data. The budget-permutation controls preserve the group-budget multiset, but not necessarily individual payment amounts M_k/K_k or actual reward exposure. The common-depth control sets $K = 10$ in each of the twelve groups (120 levels, compared with the original 114); it holds total budget fixed while testing support-based versus uniform allocation. The ten-run Craftax comparison gives paired differences of 1.49 ± 0.66 percentage points against PPO (ten positive pairs) and 0.07 ± 1.04 against Reachable (four positive pairs). The ten-run Craftax evaluation and achievement rates have been reconciled with the exported episode records. All ten training trajectories are also available for Figure 3; they retain their distinct training metric. The permutation, common-depth, and budget-neighborhood values have been reconciled with complete per-run curves and frozen programs.

The repaired first-recurrence control preserves the event groups and group budgets while changing their payment schedule. In addition to the per-episode comparison, its auxiliary reward per 1,000 steps is 9.931, compared with 5.805 for SOR. These are run-level full-training averages, not averages over only the final logging window. Group and stage payment exports provide additional accounting; longer episodes or more total episodes are not treated as equal exposure.

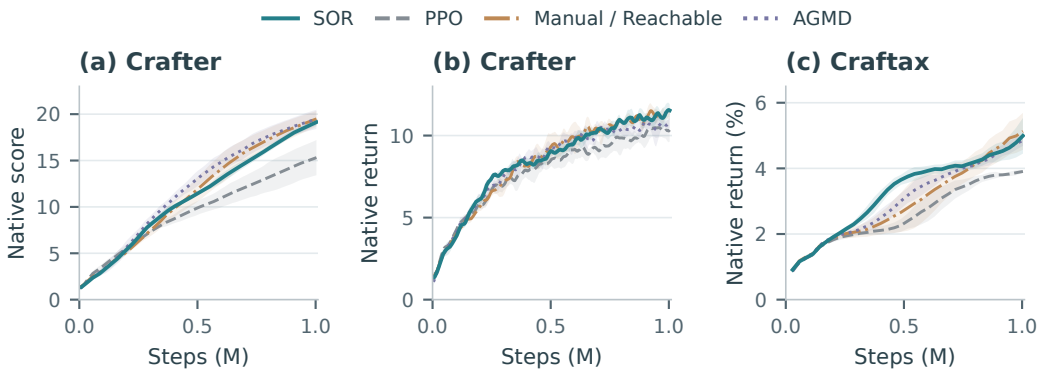


Figure 8: Historical extended native training comparisons including AGMD. The PPO, manual-reference, and SOR curves reuse four-run subsets of Figure 3; AGMD is an additional internal reference absent from the main figure. Crafter curves in this archived panel use five-point trailing display averages, whereas the updated main figure uses raw records. Craftax retains its source window.

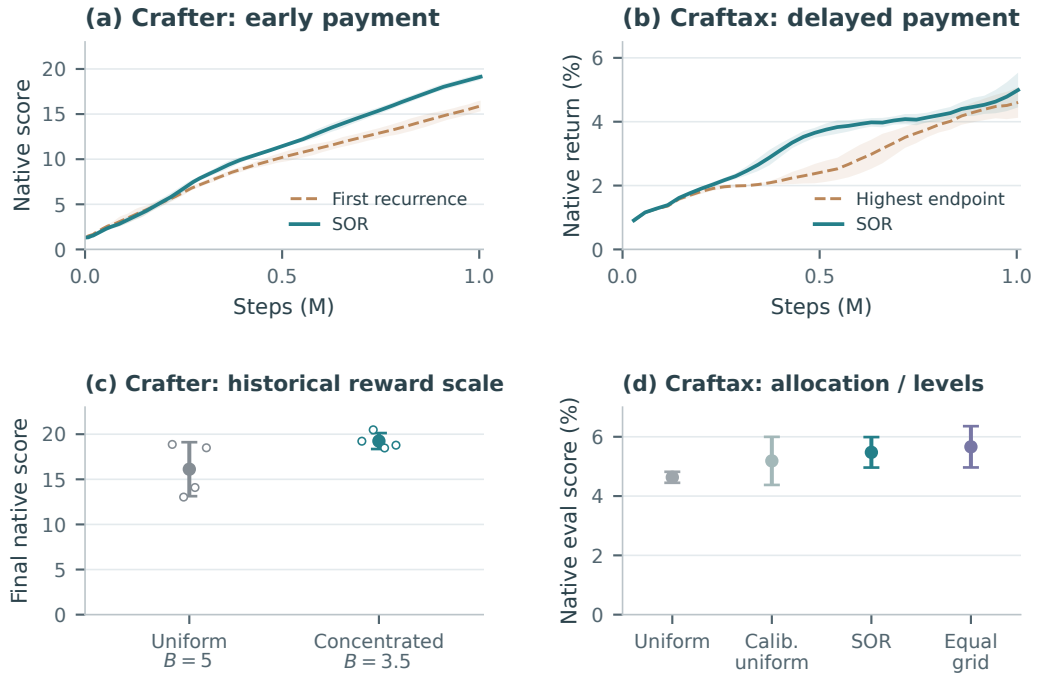


Figure 9: Historical reward-design controls. Top panels show the single-payment comparisons with the original display conventions (Crafter: five-point trailing averages). Bottom left: historical Crafter concentration versus higher-budget Uniform ($n = 4$). Bottom right: Craftax allocation and level-position controls in native-only evaluation ($n = 6$). Error bars and bands are sample SDs; online reward exposure is not matched.

Contrast	Final Score difference	AUC difference
Main Crafter PPO cohort ($n = 10$)		
SOR − PPO	2.97 ± 2.60 (8/10)	1.16 ± 1.45 (8/10)
SOR − Manual-13	-1.65 ± 1.41 (1/10)	-1.46 ± 1.24 (2/10)
Original Crafter mechanism cohort ($n = 4$)		
SOR − PPO	3.82 ± 2.00 (4/4)	1.65 ± 0.95 (4/4)
SOR − Manual-13	-0.35 ± 0.87 (1/4)	-0.36 ± 1.03 (2/4)
Native count − learned trigger	-2.41 ± 1.94 (0/4)	-1.24 ± 0.93 (0/4)
SOR − AGMD-7	-0.28 ± 0.77 (2/4)	-0.92 ± 0.50 (0/4)
SOR − first recurrence	3.23 ± 0.66 (4/4)	1.48 ± 0.62 (4/4)
Reconstruction A − O	0.72 ± 1.15 (3/4)	0.85 ± 0.93 (3/4)
Reconstruction B − O	-1.40 ± 2.04 (2/4)	-0.61 ± 0.70 (1/4)
Current AD reward comparisons ($n = 3$)		
AD + SOR − AD	1.06 ± 2.38 (2/3)	0.94 ± 0.58 (3/3)
AD + Manual-13 − AD	3.61 ± 1.16 (3/3)	1.97 ± 0.16 (3/3)
Manual-13 − SOR (AD)	2.55 ± 1.48 (3/3)	1.03 ± 0.42 (3/3)
Historical AD development cohort ($n = 3$)		
AD + SOR − AD	0.81 ± 1.29 (2/3)	0.64 ± 0.43 (3/3)

Table 20: Paired differences (mean $\pm$ sample SD). Parentheses give positive pairs. The four-run PPO comparisons reuse a subset of the ten-run cohort, while current and historical AD rows use different seeds and execution versions. Unqualified SOR denotes the original frozen construction; A/B denote reconstructions. Historical AD seeds are 422–424.

Program	Final score	Aux. / episode	Aux. / 1,000 steps
SOR	19.22 ± 0.35	1.262 ± 0.034	5.805 ± 0.117
First recurrence	15.99 ± 0.56	2.103 ± 0.034	9.931 ± 0.130
$M_{\mathrm{O}}, K_{\mathrm{B}}$	19.18 ± 0.77	1.269 ± 0.032	5.775 ± 0.181
$M_{\mathrm{B}}, K_{\mathrm{O}}$	17.14 ± 1.87	1.240 ± 0.050	5.680 ± 0.226
Reconstruction B	17.82 ± 1.73	1.236 ± 0.033	5.703 ± 0.119

Table 21: Payment and score diagnostics on repaired Crafter ($n = 4$). Payment is averaged within each run over all complete training episodes, then aggregated across runs; per-step payment uses the full training span.

Configuration	Final score	Score AUC
AD	21.43 ± 0.82	12.18 ± 0.46
AD + AGMD-7	23.25 ± 1.00	13.92 ± 1.11
SOR $(3.5, 1, 0)$	18.05 ± 3.23	10.84 ± 1.50
SOR $(3.5, 2, 0)$	22.24 ± 0.60	12.82 ± 0.40
SOR $(3, 2, 0)$	20.92 ± 2.01	12.43 ± 1.41
SOR $(3.5, 2, -\ln 4)$	19.50 ± 1.63	11.70 ± 0.97
SOR $(3, 2, -\ln 4)$	19.50 ± 3.06	11.56 ± 1.33

Table 22: Historical AD development reward configurations ($n = 3$, development seeds 422–424). All SOR rows retain the historical executor.

Program	Cumulative Score	Native return	Score AUC
AD (native reward)	19.14 ± 0.42	12.61 ± 0.14	11.27 ± 0.55
AD + Manual-13	22.75 ± 0.85	12.68 ± 0.13	13.25 ± 0.71
AD + SOR, PPO reference, $\tau = 2$	20.20 ± 2.22	11.85 ± 0.61	12.21 ± 1.09
AD + SOR, PPO reference, $\tau = 1$	18.52 ± 3.15	11.99 ± 1.10	11.27 ± 1.22
AD + SOR, AD reference, $\tau = 2$	15.42 ± 1.95	10.79 ± 0.54	9.25 ± 1.85
AD + SOR, AD reference, $\tau = 1$	18.85 ± 2.96	12.37 ± 0.91	11.02 ± 1.31
AD + SOR, historical executor	21.32 ± 0.98	12.19 ± 0.46	12.33 ± 0.87

Table 23: Current AD integration and sensitivity comparisons ($n = 3$, seeds 425–427; mean $\pm$ sample SD; 1,003,520 steps). PPO-reference programs use 12 groups and 114 levels; AD-reference programs use 13 groups and 112 levels. SOR programs use $B = 3.5$ and $\rho = 0$; Manual-13 uses the unchanged thirteen-milestone program with budget 6.5. All rows use reconciled run-level results. The final row uses the historical executor on the same seeds and is not pooled with repaired SOR. Native return is the last logged recent-episode mean.

The budget/depth swaps form a diagnostic 2×2 using original and B statistics. The final-score interaction is 0.721 ± 2.221, with mixed run-level effects; AUC interaction is 0.123 ± 1.283. The observed hybrid ranking does not identify additive causal contributions of mass and depth.

Replacing learned triggers with native event counts. We replace the learned detector backend with native cumulative event counts while retaining the original twelve groups, 114 levels, and $B = 3.5$ allocation. On seeds 422, 423, 425, and 426, this variant reaches 16.82 ± 1.77 Score and 9.98 ± 0.92 AUC, compared with 19.22 ± 0.35 and 11.22 ± 0.24 for the matched learned-detector program. The paired differences are -2.41 ± 1.94 Score and -1.24 ± 0.93 AUC; all four pairs are negative for both metrics. This intervention changes the occurrence stream to which a fixed reward program is applied, including payment timing and frequency. Native event counts therefore provide a trigger-fidelity control, not a performance upper bound, and these results alone do not identify detector error as a beneficial mechanism.

Sensitivity of the construction statistics. A supplementary sensitivity analysis resamples complete episodes within blocks with detector rules and groups fixed: 32 resamples for each of three construction data sets (515, 523, and 516 complete episodes). It measures sensitivity of q, K, M, not repeated detector induction or downstream training. Some fixed-group resamples yield zero depth and require the usual filtering before defining a valid program.

AD integration and reference sensitivity. Table 23 separates the current repaired program from alternative reference-data and exponent choices on seeds 425–427. The PPO-reference $\tau = 2$ program is the unchanged 12-group, 114-level Crafter instance used in the main PPO comparison. Relative to native AD, its paired Score gain is 1.06 ± 2.38 and its AUC gain is 0.94 ± 0.58; the recent-return difference is -0.76 ± 0.59. Changing only its exponent to $\tau = 1$ lowers the mean Score, but $\tau = 2$ wins only one of three final-score pairs; its mean advantage is driven by that run. Thus this small comparison does not establish a generally preferable exponent.

Reconstruction from two AD reference policies yields 13 groups and 112 levels. Each policy contributes one source block, with 253 and 252 complete episodes, respectively; this variant does not use the four policy/half blocks of the main Crafter construction. Neither its $\tau = 2$ nor its $\tau = 1$ variant improves mean Score or AUC over native AD. The reference-data change also changes the event set, depths, and budgets; it does not isolate reference-policy strength. Both AD-reference variants have complete run-level exports and curves. The unchanged Manual-13 program also has three completed AD runs. Its paired gains over native AD are 3.61 ± 1.16 Score and 1.97 ± 0.16 AUC points, positive in all three runs. Relative to repaired SOR, the gains are 2.55 ± 1.48 and 1.03 ± 0.42, also positive in all three pairs. Its recent-return gain over native AD is only 0.07 ± 0.13. Manual-13 retains its thirteen conditions at 0.5 each, with a maximum budget of 6.5; its comparison to SOR is not budget- or predicate-matched. The run summaries and full curves have been reconciled, including independent trapezoidal reintegration of Score AUC.

Historical AD parameter search. The earlier search varied (B, τ, ρ) on development seeds 422–424, reusing existing AD and AGMD controls under the historical executor. It evaluated three additional configurations. Table 22 preserves this search; none of its added configurations improves the mean final Score over its historical SOR $(3.5, 2, 0)$ reference. These results are not pooled with the repaired main comparison.

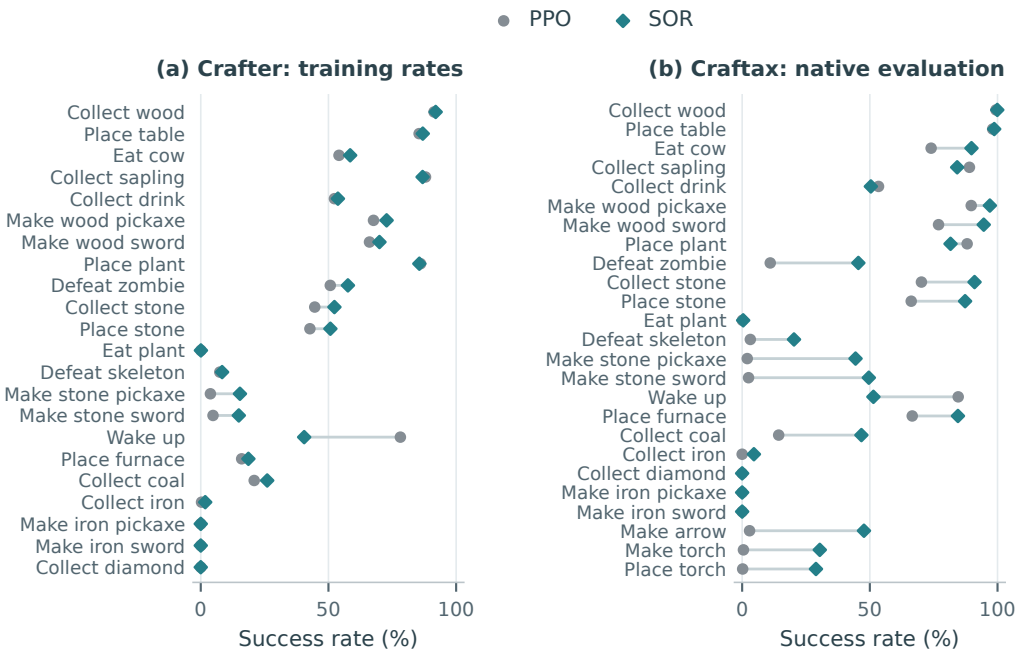


Figure 10: Complete native achievement profiles for PPO and SOR, averaged over ten training runs. Crafter shows cumulative training achievement rates for all 22 achievements. Craftax shows all 25 basic achievements from native-only evaluation, with 100 episodes per run. Among the other 42 achievements, PPO enters the dungeon once in 1,000 episodes; all SOR non-basic outcomes are zero. Segments connect means, not confidence intervals.

Achievement figures. Figure 4 retains the same four achievements from the earlier view, with the selection held fixed for the expanded cohort. Crafter rates use the final exported training achievement records for the same ten runs; Craftax rates use independent evaluation episodes. The Crafter rates aggregate achievement outcomes over recorded completed episodes throughout training, consistent with the cumulative Score metric. Figure 10 displays all 22 Crafter outcomes and all 25 basic Craftax outcomes in catalog order, without selection by observed gain. The full Craftax catalog is given below. Achievement success percentages are not an exact decomposition of native return, which can include other reward components. The separate first-recorded-achievement export records the end of the first positive rollout window, not the exact unlocking transition; we do not plot it as an exact first-unlock time.

The complete Craftax profile places the tool gains in the context of other behaviors: wake-up success is lower with SOR, so improvement does not extend to every achievement. Progress beyond the 25 basic achievements is rare for both methods: SOR records none, and PPO enters the dungeon once in 1,000 evaluation episodes. These outcomes locate the current benefit at intermediate stages without establishing reliable progression to the later tasks.

H Craftax payment and behavioral details

In the original four-run comparison (seeds 461–464), native-only eval-100 scores are 3.89 ± 0.36 for PPO, 5.59 ± 0.70 for PPO + Reachable, 4.60 ± 0.22 for PPO + Uniform, and 5.41 ± 0.65 for PPO +

Achievement	PPO	SOR	Achievement	PPO	SOR
Collect wood	99.40	99.90	Enter ice realm	0.00	0.00
Place table	98.10	98.70	Enter graveyard	0.00	0.00
Eat cow	74.00	89.80	Defeat gnome warrior	0.00	0.00
Collect sapling	89.00	84.20	Defeat gnome archer	0.00	0.00
Collect drink	53.40	50.40	Defeat orc soldier	0.00	0.00
Make wood pickaxe	89.70	97.00	Defeat orc mage	0.00	0.00
Make wood sword	76.90	94.60	Defeat lizard	0.00	0.00
Place plant	88.10	81.60	Defeat kobold	0.00	0.00
Defeat zombie	11.00	45.50	Defeat troll	0.00	0.00
Collect stone	70.20	91.00	Defeat deep thing	0.00	0.00
Place stone	66.20	87.30	Defeat pigman	0.00	0.00
Eat plant	0.10	0.40	Defeat fire elemental	0.00	0.00
Defeat skeleton	3.20	20.30	Defeat frost troll	0.00	0.00
Make stone pickaxe	2.00	44.40	Defeat ice elemental	0.00	0.00
Make stone sword	2.50	49.60	Damage necromancer	0.00	0.00
Wake up	84.60	51.40	Defeat necromancer	0.00	0.00
Place furnace	66.60	84.50	Eat bat	0.00	0.00
Collect coal	14.30	46.70	Eat snail	0.00	0.00
Collect iron	0.00	4.60	Find bow	0.00	0.00
Collect diamond	0.00	0.00	Fire bow	0.00	0.00
Make iron pickaxe	0.00	0.00	Collect sapphire	0.00	0.00
Make iron sword	0.00	0.00	Learn fireball	0.00	0.00
Make arrow	2.90	47.70	Cast fireball	0.00	0.00
Make torch	0.50	30.40	Learn iceball	0.00	0.00
Place torch	0.20	28.90	Cast iceball	0.00	0.00
Make diamond sword	0.00	0.00	Collect ruby	0.00	0.00
Make iron armour	0.00	0.00	Make diamond pickaxe	0.00	0.00
Make diamond armour	0.00	0.00	Open chest	0.00	0.00
Enter gnomish mines	0.00	0.00	Drink potion	0.00	0.00
Enter dungeon	0.10	0.00	Enchant sword	0.00	0.00
Enter sewers	0.00	0.00	Enchant armour	0.00	0.00
Enter vault	0.00	0.00	Defeat knight	0.00	0.00
Enter troll mines	0.00	0.00	Defeat archer	0.00	0.00
Enter fire realm	0.00	0.00			

Table 24: All Craftax native achievement success rates (%, means over ten training runs and 100 native-only evaluation episodes per run). The full catalog is shown. PPO enters the dungeon in one of 1,000 episodes (0.1% overall); all other non-basic outcomes, including all SOR non-basic outcomes, are zero.

SOR (mean $\pm$ sample SD). The PPO, Reachable, and SOR runs are reused in the ten-run results in Table 2; the four Uniform runs provide the matched allocation control.

Program	Extended block ($n = 6$)	Development ($n = 4$)
SOR, proportional	5.48 ± 0.51	5.84 ± 0.64
Uniform	4.63 ± 0.19	—
Calibrated Uniform	5.19 ± 0.81	—
Equal grid	5.66 ± 0.70	5.26 ± 0.70
High/low allocation	—	5.07 ± 0.58
Reachable	—	5.12 ± 0.77
Squared pooled support	—	5.39 ± 0.67

Table 25: Craftax allocation and level controls. Native-only eval-100 scores; the columns are distinct training-seed blocks, not pooled samples.

The six-seed calibrated-Uniform control used replay scaling $\alpha = 1.200330284$, implying maximum mass approximately 4.201 under whole-program scaling. This differs from a shared $B = 3.5$ cap and does not ensure equal online payment. In the later development block, measured mean payments per completed episode were 1.376 for Support, 1.228 for High/low, and 1.339 for Grid. Equal maximum budget is therefore not equal exposure.

Program	Tail native-return proxy	Native AUC ($\times 10^6$)
Support	4.994 ± 0.544	2.684 ± 0.075
Uniform	4.449 ± 0.097	2.640 ± 0.080
Endpoint-only	4.603 ± 0.480	2.276 ± 0.145

Table 26: Craftax payment-schedule training diagnostics, four seeds. Native AUC is in millions of step-integrated source units; these values are not the independent eval-100 scores.

Support's later development differences are 0.580 ± 0.563 versus Grid, 0.761 ± 0.214 versus High/low, and 0.710 ± 0.482 versus Reachable, all four positive pairs. The earlier six-seed Support–Grid difference is -0.186 ± 0.965 with two positive pairs. Their ten-pair descriptive combination remains 0.120 ± 0.883, six positive pairs. It does not establish a robust benefit of irregular internal threshold positions.

Squaring pooled support instead of using proportional allocation lowers evaluation score by 0.45 ± 0.32 points in the later development block (Table 25).

In the original four-seed validation block, Support versus Uniform improves stone-sword, arrow, and stone-pickaxe success by 44.25, 36.25, and 28.50 percentage points, while wake-up and plant placement fall by 25.00 and 9.75 points. These are descriptive behavioral changes, not independent primary endpoints or causal utilities of reward groups. Earlier behavioral replays lack full state/action snapshots and use an earlier allocation, preventing a general absence-of-exploitation claim.

I Literature comparison and generalization scope

Reference	Intervention	Comparison boundary
AD / SEA	Achievement representations or structured exploration	Different intervention and event assumptions
ELLM / DLLM	Language-guided goals	Language inputs, pretraining, or imagined rollouts
SPRING / EnvGen (Wu et al., 2023; Zala et al., 2024)	Paper-guided agent or generated environments	Different agent and extra environment training
DreamerV3 / Curious Replay	World models and replay	Different learner and compute allocation
Transformer world models	Model-based data efficiency	Full pixel results included; learner, compute, and evaluation protocols differ
Eureka / Text2Reward / ORSO	Reward generation or selection	Different inputs and candidate-feedback costs
Reward Machines / Translation (Icarte et al., 2018; Hua et al., 2025)	Reward representation and transfer	Representation alone is not SOR's novelty
LEO / CODE-SHARP	All-goals or hierarchical skills	Different policy/task interfaces; LEO is ICML 2026, CODE-SHARP is a preprint

Table 27: Related methods, their intervention targets, and comparison scope.

AD's task-family formulation assumes shared hierarchical achievement structure across instances and evaluates changes in generated scenes (Moon et al., 2023). Its extensions include QR-DQN, Procgen Heist with modified lock/gem rewards and 25M training steps, and a custom six-room MiniGrid with ten first-unlock achievements and 1M steps. These are broader tests than our two-environment SOR study, but are still structured task-family evaluations. An event that happens once, such as opening a particular door, need not admit useful recurrence feedback. Consequently those tasks cannot be adopted as evidence for SOR without checking its own assumptions.

EnvGen changes the training environment through generated pretraining tasks (Zala et al., 2024). We discuss it as a complementary intervention without including its results in a performance ranking across different training budgets and supervision regimes.

Close reward-construction alternatives. Table 28 distinguishes the information used to construct rewards from the reward structure produced. These are method-level comparisons of supervision, construction, and output. SOR constructs a frozen program of finite event-recurrence payments from first-unlock supervision and reference traces, without language-model calls.

Method	Construction input / mechanism	Relation to SOR
VICE (Fu et al., 2018)	Desired-outcome examples; learned event probabilities	Event supervision is established; SOR adds finite recurrence depth and allocation
Learned RMs / ISA (Icarte et al., 2019; Furelos-Blanco et al., 2021)	Property detectors or high-level event traces; learned temporal automata	SOR induces detectors within a grammar and constructs restricted count programs
LIRPG (Zheng et al., 2018)	Reward parameters updated through extrinsic learning feedback	SOR uses empirical support statistics; downstream tuning is separately reported
ProgressCounts (Sarukkai et al., 2024)	LLM progress functions; discretization and progress-bin visitation counts	SOR counts within-episode event recurrence and caps payments
Strategist (Ruiz Luyten et al., 2025)	LLM strategies and progress evaluations distilled into reward feedback	A direct Crafter progress-reward neighbor, with different supervision
Adaptive reward design (Kwon et al., 2025)	Temporal-logic task specification; adaptive partial-progress rewards	SOR's program is constructed from labeled traces and then frozen
CODE-SHARP (Bornemann et al., 2026)	Foundation-model reward programs and a hierarchical skill archive	SOR constructs a finite reward program for an existing learner

Table 28: Selected reward-construction methods: inputs, mechanisms, and relation to SOR.

Additional exploration and task-learning context. NovelD uses novelty differences, and E3B measures episodic novelty with an elliptical bonus (Zhang et al., 2021; Henaff et al., 2022). The relative effectiveness of global and episodic bonuses depends on environment structure (Henaff et al., 2023). These findings motivate distinguishing the information represented by a count from the numerical form of its bonus. SOR's group support and within-episode recurrence counter serve different roles from state-visitation novelty.

Options formalize temporally extended policies (Sutton et al., 1999). HER relabels achieved goals in experience, and SAC-X learns and schedules auxiliary intentions (Andrychowicz et al., 2017; Riedmiller et al., 2018). These are additional ways to reuse intermediate behavior. SOR retains the learner's policy interface and supplies recurrence feedback through its training reward.

Reward transformations and transfer. RUDDER and Align-RUDDER redistribute delayed rewards using return decomposition or demonstration alignment (Arjona-Medina et al., 2019; Patil et al., 2022). Potential-based shaping provides policy-invariance conditions for reward transformations (Ng et al., 1999), while reward translation studies transfer between semi-alignable environments (Hua et al., 2025). SOR adds bounded auxiliary return. Its construction neither estimates a return-equivalent redistribution nor establishes these policy-invariance or transfer guarantees.

Learning systems and benchmark protocols. DreamerV3, Curious Replay, transformer world models, and Simulus address learning or data reuse through world-model systems (Hafner et al., 2025; Kauvar et al., 2023; Dedieu et al., 2025; Cohen et al., 2025). The C-TeC study includes reward-free Craftax-Classic exploration (Mohamed et al., 2026). Their published results provide context rather than matched tests of SOR's reward intervention in PPO on Craftax-full. Comparisons require matching environment variant, observations, supervision, and learner resources, and accounting for reference-data collection and search costs as well as downstream interaction steps.

J DISCUSSION AND LIMITATIONS

The results support constructing useful recurrence feedback with a finite budget. Its use of observed recurrence links the resulting program to the coverage of the reference data.

Reference coverage and reliability. Unconditional support retains accessibility information but can emphasize common events and miss rare prerequisites. Cross-block depth only requires one reaching episode per block, making it sensitive to data collection. Support measures observed recurrence rather than causal event value, and the resulting depths represent observed coverage rather than optimal caps. The construction repeats show that these choices can affect performance. The Crafter and Craftax recurrence audits strengthen detector evidence on their sampled distributions; they do not certify all states visited by reward-trained policies. Structured fields used by the reward constructor remain an information requirement even though the policy observation interface is unchanged.

Scope and resource accounting. The construction requires recognizable repeatable events and reference coverage. The two related environments test independently fitted instances, not transfer of a frozen program or a universal concentration setting. AD integration broadens the learner interface tested. Its small sample, variable final Score, lower recent native return, and unsuccessful AD-reference reconstruction limit broader performance conclusions. Reference-policy training, collection, and search add costs beyond the reported downstream budget. The budget bound controls reward magnitude, but does not guarantee native-policy invariance or prevent unhelpful repetition.