Deep Reinforcement Learning for User Welfare Optimization on Recommendation Systems with Competing Content Creators

Tanush Talati[1] Julia Isaac[1]
[1]Department of Computer Science, Stanford University

June 9, 2026

Abstract

Content creators on modern recommendation platforms are known to be *strategic agents*: they choose what content to produce based on the rewards they receive, with such reward signals being mediated by the platform's matching and ranking decisions. Without platform intervention, prior work shows that creators rationally concentrate near the largest user clusters—where competition has the highest expected payoff—leaving niche user groups with little relevant content. While individual creators may see increased welfare in the short term, the platform degrades over time since the allocation is not globally optimal. This structural failure mode, known as *popularity bias*, depresses mean user welfare and erodes platform retention among underserved groups, and is one of the central problems platform owners must address.

Yao et al. (KDD 2024) formalize this interaction as the C^3_{ext} game between three players: a *platform* that commits a per-group importance weight vector $\mathbf{w} \in \mathbb{R}^L_{\geq 0}$ which acts as the platform's knob to indirectly shape creators' content, n *strategic creators* who reposition each epoch under Local Better Response (LBR) dynamics to maximize their $\mathbf{w}$-weighted utility, and *users* matched to creators via a softmax over a tent-kernel relevance score. The platform observes only the per-group average user utility $\bar{\boldsymbol{\pi}}^{(e)}$ each epoch and chooses $\mathbf{w}^{(e+1)}$ for the next epoch. Yao et al. propose an analytical *User Importance Reweighting* (UIR) rule, $\mathbf{w}^{(e+1)} \propto \mathbf{w}^{(e)} \cdot \exp(-\alpha\, \bar{\boldsymbol{\pi}}^{(e)})$, which boosts under-served groups (low $\bar{\pi}_l$) and is provably welfare-improving under three strong idealizations: orthogonal cluster centers, single-cluster user membership, and noise-free LBR creator response. They deploy UIR in production and report a +1.13% lift in their core utility metric over no-intervention. Our work investigates whether deep learning could extract utility beyond a hand-derived first-order rule and involves implementing a custom simulator to test Yao's environment as well the non-ideal ablations to it.

Concretely, we extend Yao's framework along three axes that the analytical rule does not address. **(i) MDP formulation.** We cast the platform's intervention as a Markov Decision Process and train state-conditional controllers that consume the full observation $\mathbf{o}_e = (\mathbf{w}^{(e)}, \bar{\boldsymbol{\pi}}^{(e)}, \text{hist}_4^{(e)}, t/T, \boldsymbol{\mu}_{1:L})$ consisting of the current weights, the latest per-group utility, the past four epochs' utility vectors as a smoothing signal, the episode time fraction, and the cluster centroids that encode environment geometry. Note that none of hist_4, t/T, or $\boldsymbol{\mu}$ are consumed by Yao's rule. **(ii) Yao-residual action parameterization.** We map the policy action $\mathbf{a}_e \in \mathbb{R}^L$ to the next weight vector as $\mathbf{w}^{(e+1)} \propto \mathbf{w}^{(e)} \cdot \exp(-\alpha \bar{\boldsymbol{\pi}}^{(e)}) \cdot \exp(\mathbf{a}_e)$, which factors Yao's analytical update out as the base and lets the policy learn a small correction on top. Setting $\mathbf{a}_e = \mathbf{0}$ recovers exact Yao UIR, so the policy's initial behavior is the analytical solution; we therefore initialize the actor's mean head to produce near-zero output. Vanilla deep RL from random $\mathbf{w}$ without this warm-start underperforms no-intervention in our experiments. **(iii) Stress environments.** We introduce three perturbations of Yao's ideal setup, each ablating one idealization: *noisy LBR* (creators add Gaussian noise to each move), *multi-cluster* (each user has soft membership across clusters), and *sparse embedding* (cluster centers lie in a 2D sub-manifold of $\mathbb{R}^d$). These let us measure where Yao's analytical guarantee weakens and where state-conditioning recovers additional welfare.

We train four controllers spanning two algorithmic design axes, on-policy vs. off-policy and stochastic vs. deterministic actor plus a gradient free baseline. **PPO** (on-policy, stochastic) optimizes a clipped surrogate with Generalized Advantage Estimation, $K = 10$ inner epochs over rollouts of length $T_{\text{roll}} = 128$ across $n_{\text{env}} = 8$ vectorized environments, with KL early-stop aborting the inner loop on $\hat{D}_{\text{KL}} > 0.015$. **SAC** (off-policy, stochastic) maximizes an entropy-regularized return using the reparameterization trick on a tanh-squashed Gaussian policy with twin critics $\min(Q_1, Q_2)$ to combat overestimation bias and an auto-tuned entropy coefficient. **TD3** (off-policy, deterministic) uses a deterministic actor with twin critics, target-policy smoothing, and delayed policy updates every $d = 2$ critic steps. **CMA-ES** performs gradient-free evolutionary search over a *constant* $\mathbf{w}$: it has no policy network and no temporal structure, so it isolates how much welfare is recoverable by tuning the best fixed weight vector which we treat as the open-loop ceiling.

Across the four environments and 3–10 random seeds per configuration, learned controllers **match Yao UIR on the ideal environment** (SAC and TD3 tie within noise; PPO trails by −0.2%) and **exceed Yao on both stress environments**, by +0.1 to +0.3% on multi-cluster and +0.4 to +0.6% on sparse embedding. CMA-ES tracks Yao to within 0.2% across all environments, establishing that the best static weight vector is essentially Yao's UIR fixed point and consequently every welfare gain deep RL extracts above this line comes from *the policy reacting to per-epoch state*, not from finding a better fixed target. The gains scale with how severely the environment violates Yao's analytical assumptions: near-zero on the ideal environment (no useful signal beyond $\bar{\boldsymbol{\pi}}$), small on multi-cluster, and largest on sparse embedding where there is less noise in the dynamics, allowing RL to be more powerful. No single algorithm dominates: PPO and TD3 tie on both stress environments, while SAC matches Yao on the ideal environment. The small per-method differences reflect each algorithm's distinct strengths, TD3's low-variance deterministic gradient, SAC's entropy-driven exploration, and PPO's conservative trust-region updates, which interact differently with each environment's structure. We discuss this in detail in the main paper. Action parameterization (Yao-residual vs. multiplicative) and state representation (presence of cluster centroids in $\mathbf{o}_e$) are nonetheless the more decisive design axes.

The +0.6% deep-RL lift on top of Yao's reported +1.13% deployment lift over no-intervention yields a cumulative $\sim$1.6% welfare gain over a passive platform, a fraction of a percent that maps to enormous absolute welfare at platform scale. Our results point to two open directions: richer state encoders and softer action parameterizations, either of which we have seen is likely to yield larger absolute gains than swapping in a different RL algorithm.

1 Abstract Abridged

Content creators on modern recommendation platforms act strategically, optimizing for monetary reward and audience engagement signals. Without targeted platform intervention, creators naturally herd toward majoritarian user segments to maximize expected payoffs, creating popularity bias that leaves niche audiences under-served. Recent game-theoretic approaches address this via a first-order User Importance Reweighting (UIR) feedback rule that dynamically alters ranking weights; however, its welfare guarantees rely on assumptions like orthogonal user clusters, hard demographic memberships, and noise-free creator local adjustments. This paper investigates whether deep reinforcement learning (RL) can optimize platform-level user welfare in environments that selectively violate these idealizations. We model the platform's intervention as an MDP, introducing an informational state representation that tracks cluster geometry and engagement histories, alongside a novel *Yao-residual action parameterization* that warm-starts policies near the known analytical baseline. Across simulated stress environments ablated for geometric sparsity and multi-modal user interest mixture, our state-conditional controllers (PPO, SAC, and TD3) effectively match first-order updates under clean conditions while extracting an additional $+0.3\%$ to $+0.6\%$ platform welfare lift when analytical assumptions break.

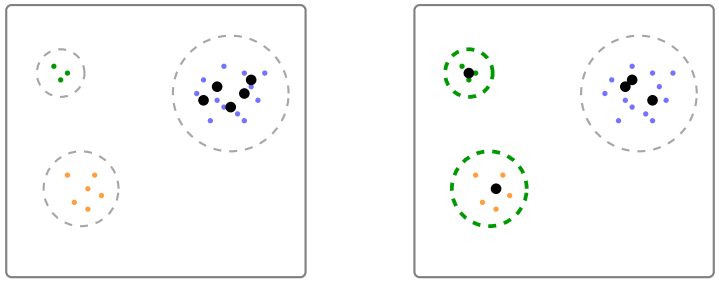
2 Introduction and Motivation

Modern content platforms like YouTube, TikTok, Spotify, and Instagram generate revenue primarily through user engagement, monetized through advertising, subscriptions, and creator-monetization programs that share a fraction of platform revenue back to creators based on the engagement their content receives. This creates a tightly coupled incentive structure: the platform's revenue depends on users staying engaged, users stay engaged when they see content that matches their interests, and content is produced by creators whose own income depends on the rewards (views, watch-time, ad-revenue share) that the platform's matching and ranking decisions surface to them. Crucially, creators are not passive content sources but *strategic agents*: they continuously adjust what they produce based on what the platform's algorithm appears to reward. A creator who notices that a particular trending format, for instance a short-form dance with a viral soundtrack, a recipe filmed in a recognizable style, a particular genre of commentary, reaches a broad audience will pivot to producing more of it, because trending content offers the highest expected short-term payout. A creator targeting a niche audience will see slower growth and, in the limit, either pivot to chasing trends or stop producing altogether.

When this incentive structure operates without platform intervention, the resulting equilibrium is well understood. Creators rationally concentrate on whatever maximizes expected reward, which is typically content that targets the largest user clusters since those clusters offer the highest expected engagement and therefore the highest expected payout per piece of content produced. The result is a positive feedback loop: trending topics draw more creators, the trending content saturates user feeds, and users in less-represented interest groups (niche music genres, regional cultures, specialized hobbies, under-represented languages) increasingly fail to find content that matches them. This is the *popularity bias* phenomenon. Individual creators may see short-term welfare gains from chasing trends, but aggregate platform welfare which is the mean quality of the user-content match across the full user population will degrade, because the global content allocation is not welfare-optimal. It also prevents discovery and exploration of new trends and niches. The platform sees this in retention metrics for under-served user groups; users see it as recommendations that grow increasingly homogeneous; creators see it as winner-take-all dynamics in their own ecosystem.

A worked example. To make popularity bias concrete, consider a small setting with three user clusters of sizes 60, 30, and 10 (out of 100 users) and five creators who freely choose which cluster to target. A creator who targets a cluster is matched to a roughly proportional fraction of its users; multiple creators targeting the same cluster split that audience. Without platform intervention, every creator's best response is to target the largest cluster, because the expected payout per creator is proportional to cluster size divided by the number of creators already there. The equilibrium has all five creators on the largest cluster: users in cluster 1 see saturated, mediocre matches (more creators than distinct interests), and users in clusters 2 and 3 see no relevant content at all. In this sample example, the mean welfare across the population is roughly $0.6 \cdot 0.5 + 0.3 \cdot 0.0 + 0.1 \cdot 0.0 = 0.30$ in our normalized welfare scale. With a platform-induced redistribution that moves two creators away from cluster 1 (one each to clusters 2 and 3), users in the niche clusters receive high-quality matches and the crowded cluster sees better match quality per remaining creator. Mean welfare rises to roughly $0.6 \cdot 0.7 + 0.3 \cdot 0.85 + 0.1 \cdot 0.85 = 0.76$, a 0.46-point gain in welfare over no-intervention. Figure 1 visualizes the two end-states. This is the intuition Yao et al. formalize at scale, and the welfare gap is what their UIR rule, and our learned controllers, attempt to close.

Yao et al. formalize this dynamic as the C^3_{ext} (Content Creator Competition, extended) game between three players and this is visualized in Figure 2. A *platform* commits a per-group importance weight vector $\mathbf{w} \in \mathbb{R}^L_{\geq 0}$, the only lever it has in the game, which rescales how much each user group's utility counts toward a creator's reward. In practice, this rescaling could be implemented through creator monetization (creators producing for a particular user segment receive a higher ad-revenue share or appear in promoted slots), differential ranking (recommendations algorithmically boost content matched to under-served clusters), or explicit creator-fund programs that pay out by category. Any of these constitutes a soft indication of the



(a) No intervention. *Mean welfare* ≈ 0.30. *(b) Platform redistributes.* *Mean welfare* ≈ 0.76.

Figure 1: End-state of a toy 3-cluster, 5-creator system under (a) no intervention and (b) platform-induced redistribution. Dashed circles indicate cluster boundaries (green = served, gray = abandoned), colored dots are users, black dots are creators. Without intervention, every creator's best response is to target the largest cluster, depriving niche users of relevant content. A redistribution moves two creators to the smaller clusters, raising mean welfare from 0.30 to 0.76 the welfare gap Yao's UIR rule and our learned controllers attempt to close.

platform's preference for what kind of content the creator should pursue.

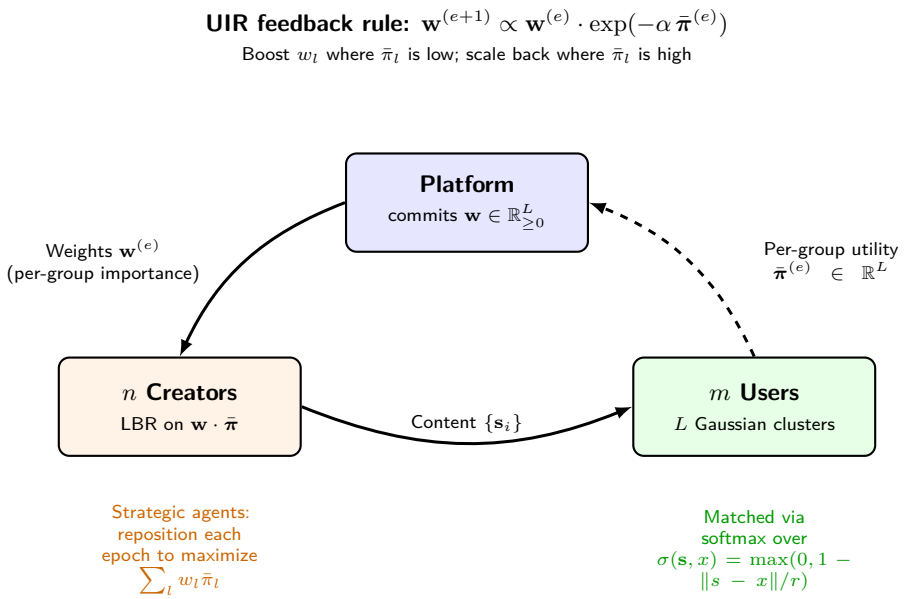


Figure 2: The C^3_{ext} three-player game. The platform commits a per-group weight vector $\mathbf{w} \in \mathbb{R}^L_{\geq 0}$ each epoch. This weight rescales how much each user group's utility counts toward a creator's reward. The n creators respond by performing Local Better Response (LBR), where each creator samples a small neighborhood of candidate next positions (just 1 in our main experiments) and adopts the one that maximizes its $\mathbf{w}$-weighted expected utility. The m users are matched to creators via a softmax over a tent-kernel relevance score, producing per-user match qualities that the platform aggregates into a per-group utility vector $\bar{\boldsymbol{\pi}}^{(e)}$. The platform sees only $\bar{\boldsymbol{\pi}}^{(e)}$ (not individual creator positions or user identities) and uses it to update $\mathbf{w}^{(e+1)}$ for the next epoch.

A population of n *strategic creators* with positions $\mathbf{s}_i \in \mathbb{R}^d$ in a content-embedding space reposition each epoch under Local Better Response (LBR) dynamics, a bounded-rationality local-search rule that each creator independently applies to maximize their $\mathbf{w}$-weighted utility. The position $\mathbf{s}_i$ represents what *kind* of content the creator produces encompassing their style, niche, format, or topic mix as a point in a learned content-embedding space where nearby points correspond to similar content. "Repositioning" then represents the creator's incremental choice of what to make next: a tech YouTuber might add more humor to their explainers, or shift from long-form tutorials to short-form demos. LBR models the fact that creators do not re-engineer their entire brand each epoch but probe a small neighborhood of nearby content options and pick the one with the best expected payoff. The bounded-rationality assumption captures real-world constraints: creators cannot survey the full content landscape, so they iterate locally based on what is working in their vicinity.

A population of m *users* with query points $x \in \mathbb{R}^d$ drawn from L Gaussian interest clusters are matched to creators via a softmax over a tent-kernel relevance score. Each query point x represents a user's interest at a moment in time, basically what they are implicitly searching for when the platform serves a recommendation. The L clusters represent the natural

groupings of similar interests in the population (e.g., a "sports fans" cluster, a "cooking enthusiasts" cluster, a "k-pop fans" cluster); users within a cluster share broad interests but differ in fine-grained taste, which the Gaussian spread within each cluster captures. The tent-kernel relevance score $\sigma(\mathbf{s}, x) = \max(0, 1 - \|\mathbf{s} - x\|/r)$ encodes the intuition that recommendation quality decays with distance between user intent and content position, going to zero past a radius r. The softmax matching $P(\mathbf{s}_i \mid x) \propto \exp(\sigma(\mathbf{s}_i, x)/\beta)$ mirrors how real recommender systems route attention: they do not deterministically serve the single closest match but distribute exposure probabilistically among the top-relevance candidates, with the temperature β controlling how sharply attention is concentrated on the best matches.

The platform observes only the per-group average user utility $\bar{\boldsymbol{\pi}}^{(e)} \in \mathbb{R}^L$ at the end of each epoch, notably it never sees individual creator positions or user identities and adjusts $\mathbf{w}$ for the next epoch. This partial-observation structure mirrors the real-world constraint that platforms cannot directly inspect every creator's strategic decisions or every user's internal preferences. Instead, they observe aggregate engagement signals: average watch time per content category, retention rates per demographic segment, click-through rates per content cluster. The aggregate signal is the only feedback channel the platform has, and any intervention must be derived from it. This is also consistent with practical privacy considerations, even if a platform could observe finer-grained signals, regulatory and trust constraints often limit it to aggregate per-segment metrics.

Yao et al. prove that under three idealized conditions, orthogonal cluster centers, single-cluster user membership, and noise-free LBR creator response, an analytical *User Importance Reweighting* (UIR) rule,

$$\mathbf{w}^{(e+1)} \propto \mathbf{w}^{(e)} \cdot \exp\big(-\alpha\, \bar{\boldsymbol{\pi}}^{(e)}\big), \tag{1}$$

monotonically improves welfare by boosting the weight of any group whose average utility $\bar{\pi}_l$ is currently low, making content for that under-served group more rewarding to produce and de-emphasizing groups already being well served. Intuitively, the rule is a feedback controller: when a user segment is being neglected (low $\bar{\pi}_l$), raise the incentive for creators to target that segment; once it is well served, scale the incentive back down. Deployed in production on a real recommender platform, UIR delivered a +1.13% lift in the platform's core utility metric over no-intervention, with secondary gains of +0.76% cold-start impressions and +0.71% creator income.

The three idealized conditions Yao's analysis requires are precisely the kind of structural assumptions that real recommender environments routinely violate. Cluster centers in a learned content-embedding space need not be orthogonal; user interests are typically soft mixtures across multiple clusters rather than hard assignment to one; creator behavior involves genuine exploration and noise rather than pure local-better-response. We hypothesize that in regimes where these idealizations break, a *state-conditional* learned controller could extract additional welfare beyond what Yao's hand-derived first-order rule achieves by conditioning its corrections on signals (the geometry of the cluster centroids, the recent history of $\bar{\boldsymbol{\pi}}$, the episode time fraction) that Yao's rule cannot consume by construction.

This paper presents the deep reinforcement learning system we built to test that hypothesis. Our contributions are:

- **An MDP formulation** of the platform's intervention whose observation includes cluster centroids and a 4-epoch utility history, which are signals Yao's rule discards.
- **A Yao-residual action parameterization** that warm-starts the policy at Yao's analytical solution and lets it learn a small correction on top. Empirically, this is what makes deep RL tractable in this regime; vanilla deep RL from random $\mathbf{w}$ underperforms no-intervention in our experiments.
- **A set of three stress environments** *noisy LBR*, *multi-cluster*, and *sparse embedding* each of which selectively ablates one of Yao's analytical idealizations. These let us measure where state-conditional learning helps and by how much.
- **A comparison across four learned controllers** PPO, SAC, TD3, and a gradient-free CMA-ES baseline that span the on-policy/off-policy and stochastic/deterministic actor design axes, with CMA-ES establishing the open-loop welfare ceiling.
- **A comprehensive simulator** of the game, which also models the various user, cluster, creator, and weight update behaviors we modeled beyond Yao's assumptions.

The remainder of the paper is organized as follows. Section 3 situates our work against prior strategic-recommender and welfare-optimization literature. Section 5 describes the C^3_{ext} simulator we implemented, the three stress environments, and the 2D visualizations that motivate the empirical setup. Section 6 details our MDP formulation, the two action parameterizations, and the four learned controllers along with their training loops and hyperparameter choices. Section 7 reports welfare across all environments and analyzes which design axes (algorithm choice, action parameterization, state representation) explain the observed welfare ordering. Section 8 discusses limitations, open directions, and the follow-up experiments planned for the future version of this work.

3 Related Work

Our work sits at the intersection of three threads: strategic content provider models that formalize creator incentives in recommendation platforms, welfare optimization in recommender system design, and deep reinforcement learning for continuous control.

Strategic content providers. Ben-Porat and Tennenholtz [2018] introduce a game-theoretic model in which creators are no-regret learners adapting their content to maximize cumulative reward under a fixed platform matching rule, and characterize the resulting equilibrium content distribution. Jagadeesan et al. [2022] extend this analysis to study supply-side equilibria when multiple creators compete for user attention. Our work builds directly on the C^3_{ext} framework of Yao et al. [2024], which extends these models to a setting where the platform observes per-group user utility each epoch and updates its weights for the next epoch, and derives the analytical User Importance Reweighting (UIR) rule we use as our baseline. Yao et al. prove UIR is welfare-improving under three structural idealizations (orthogonal cluster centers, single-cluster user membership, noise-free LBR creator response). Our contribution is to ask whether a state-conditional learned controller can extract additional welfare when those idealizations break.

Welfare optimization in recommenders. Mladenov et al. [2020] formulate welfare optimization as an offline constrained matching problem solved via integer programming, producing a fixed matching rule that maximizes welfare subject to creator-side and user-side constraints. Their approach does not adapt online and does not model strategic creator response which are both contributions Yao et al.'s framework adds. We follow Yao et al.'s framing, treating welfare optimization as a problem of repeatedly adjusting platform-side weights based on observed per-group utility under a strategic creator population.

Deep reinforcement learning. We deploy four standard continuous-control methods that span the modern design space. PPO [Schulman et al., 2017] optimizes a clipped importance-ratio surrogate with Generalized Advantage Estimation [Schulman et al., 2016], enabling stable multi-epoch reuse of each rollout. SAC [Haarnoja et al., 2018] adds an entropy bonus to the standard return and uses a reparameterized stochastic actor with twin Q-networks to combat off-policy overestimation bias. TD3 [Fujimoto et al., 2018] addresses three failure modes of DDPG [Lillicrap et al., 2016]: target-policy smoothing, clipped double-Q targets, and delayed policy updates. CMA-ES [Hansen et al., 2003] is a gradient-free evolutionary search that we use to establish the open-loop welfare ceiling, or the best welfare achievable by a *constant* policy. The application of deep RL to recommendation is well-established, but typically treats the recommender itself as the policy outputting item recommendations. Our framing is s different: the policy outputs the platform's reweighting parameters that shape creator incentives, and the actual matching is performed by the stocastic mechanism the simulator implements.

Positioning. Our contributions relative to this prior work are: (i) we are the first to apply deep RL to the C^3_{ext} welfare-optimization problem; (ii) we introduce the Yao-residual action parameterization that makes the resulting learning problem tractable by warm-starting the policy at the analytical UIR solution; (iii) we introduce stress environments that selectively ablate Yao's structural assumptions, allowing us to measure where state-conditional learning helps and by how much. In this manner, we show potential for more robust algorithms, simulators, and framework for recommendations. Table 1 summarizes the major differences.

Method	Strategic creators	Online adapt.	Welfare objective	Learned policy
Mladenov et al. 2020	—	—	✓	—
Ben-Porat & Tennenholtz 2018	✓	—	—	—
Jagadeesan et al. 2022	✓	—	—	—
Yao et al. 2024 (UIR)	✓	✓	✓	—
This work	✓	✓	✓	✓

Table 1: Comparison of prior work along four axes. **Strategic creators**: does the model treat content creators as strategic agents that respond to platform incentives? **Online adaptation**: does the platform adjust its mechanism between epochs based on observed feedback? **Welfare objective**: is the platform-level user welfare the explicit optimization target? **Learned policy**: is the platform's update rule learned from data rather than hand-derived? Our work combines all four.

4 Problem Statement

Concretely, the problem we address is the platform-side welfare optimization in the C^3_{ext} game of Yao et al. [2024]. At each epoch $e \in \{0, 1, \ldots, T-1\}$, a platform observes the per-group average user utility $\bar{\boldsymbol{\pi}}^{(e)} \in \mathbb{R}^L$. It sees neither individual creator positions nor user identities and chooses an importance weight vector $\mathbf{w}^{(e+1)} \in \mathbb{R}^L_{\geq 0}$. The choice of $\mathbf{w}^{(e+1)}$ shapes how

n strategic creators reposition in the next epoch under Local Better Response (LBR) dynamics, which in turn determines the mean user-content match quality the platform achieves through softmax matching over a tent-kernel relevance score. The platform's objective is to maximize the mean user welfare averaged across an episode of T epochs. Note that the finite horizon T is a modelling convenience rather than a fundamental limitation: in practice, a platform can deploy any of these strategies as a discrete weighting campaign that runs for a fixed period to redistribute creator attention toward a desirable equilibrium, then reset the game state and launch a new campaign with updated objectives as the user or creator population evolves. Each such campaign maps onto one T-epoch episode in our formulation, so optimizing over a finite horizon does not preclude continuous real-world deployment and instead it formalizes the natural unit of strategic intervention the platform would actually run.

Yao et al. prove that their analytical User Importance Reweighting (UIR) rule monotonically improves welfare under three idealized conditions: orthogonal cluster centers, single-cluster user membership, and noise-free LBR creator response. The central question of this paper is whether a *state-conditional learned controller*, one that conditions on signals Yao's hand-derived rule discards, including cluster geometry, recent utility history, and episode time, can extract additional welfare beyond UIR when those idealizations fail to hold, and in which regimes the gap is largest.

5 Simulator and Environment Setup

A central part of this project was implementing the C^3_{ext} simulator from Yao et al. [2024] as a vectorized Python environment, then extending it to support three stress-environment configurations that selectively violate the structural idealizations Yao's analysis requires. This section describes the game formalism, our implementation, two intentional deviations from Yao's reference setup that trade fidelity for substantial speedups, the standard environment configuration we use for the ideal-condition baseline, and the three stress variants.

5.1 The C^3_{ext} Game Formalism

The simulator models a single episode of the three-player game between a platform, n strategic content creators, and m users. Each entity lives in a shared d-dimensional content-embedding space.

Users and clusters. The user population consists of m query points $\{x_j\}_{j=1}^m$ distributed across L Gaussian interest clusters. Each cluster has a center $\boldsymbol{\mu}_l \in \mathbb{R}^d$ and a shared per-cluster standard deviation σ_{cluster}; user j in cluster l is sampled as $x_j \sim \mathcal{N}(\boldsymbol{\mu}_l, \sigma^2_{\text{cluster}} I)$. The cluster centers themselves are sampled uniformly from the surface of the unit ball in $\mathbb{R}^d$ at the start of each episode (alternatively, they can be set explicitly to support the sparse-embedding stress environment described in Section 5.6.3, where the centers are constrained to a 2D sub-manifold). Each user j is assigned to a primary cluster index $g_j \in \{1, \ldots, L\}$; in the standard environment, this assignment is hard (single-cluster membership). The cluster sizes need not be balanced and we use group sizes $\mathbf{c} = (c_1, c_2, \ldots, c_L)$ which together sum to m. The user population is sampled once per episode and held fixed thereafter.

The simulator additionally supports a *centroid drift* mechanism in which cluster centers undergo a Gaussian random walk between epochs, $\boldsymbol{\mu}_l^{(e+1)} = \boldsymbol{\mu}_l^{(e)} + \boldsymbol{\eta}_l$, $\boldsymbol{\eta}_l \sim \mathcal{N}(0, \rho^2 I)$, controlled by the drift rate ρ. This is intended to model platforms where user interest distributions evolve over time the rise of a new content genre, a shift in news coverage, or seasonal interest patterns. For all main experiments we set $\rho = 0$ (no drift) to isolate the effect of the three stress perturbations from any non-stationarity in the user population. Therefore, the drift mechanism is a feature of the simulator code we do not exercise in the reported results. We discuss enabling drift as a direction for future work in Section 8.

A second simulator feature we exercise specifically for the *multi-cluster* stress environment is the multi-modal user query distribution: each user maintains a categorical distribution over clusters and their query point $x_j^{(e)}$ is resampled from this distribution each epoch (Section 5.6.2). In the standard environment, x_j is sampled once and held fixed.

Creators. The creator population consists of n content producers with positions $\{\mathbf{s}_i\}_{i=1}^n \in \mathbb{R}^d$. The positions represent the type of content each creator produces in the embedding space, and they evolve across epochs as creators reposition in response to the platform's weights. At episode start, creators are initialized near the largest cluster center, modeling the natural state of a platform before intervention, where the rational best response for every creator is to target the dominant audience. This is a simplification of real-world conditions: in practice, the creator population on a live platform is already heterogeneous, with some creators producing for niche audiences out of personal interest, because they have found a niche where competition is lower, or because they were already producing that content before joining the platform. A more faithful initial condition would sample creator positions from a distribution that reflects this existing heterogeneity, for example by drawing from the same Gaussian mixture as the user population, possibly with bias toward larger clusters. We discuss this

as a direction for future work in Section 8 and note that any welfare gap we report between intervention and no-intervention may be an *upper bound* on what real-world intervention would recover, since a heterogeneously initialized creator population would already partially serve niche clusters, leaving less room for any platform mechanism to improve welfare.

Relevance and matching. For each (user, creator) pair, the simulator computes a relevance score via the tent kernel

$$\sigma(\mathbf{s}, x) = \max\left(0,\ 1 - \frac{\|\mathbf{s} - x\|}{r}\right), \tag{2}$$

where r is the kernel radius beyond which relevance is zero. The matching of a user x to creator $\mathbf{s}_i$ is then a softmax over relevance scores across all creators,

$$P(\mathbf{s}_i \mid x) = \frac{\exp\left(\sigma(\mathbf{s}_i, x)/\beta\right)}{\sum_{i'=1}^{n} \exp\left(\sigma(\mathbf{s}_{i'}, x)/\beta\right)}, \tag{3}$$

where β is a temperature controlling how sharply matching is concentrated on the most relevant creators. The mechanisms can be visualized in Figure 3.

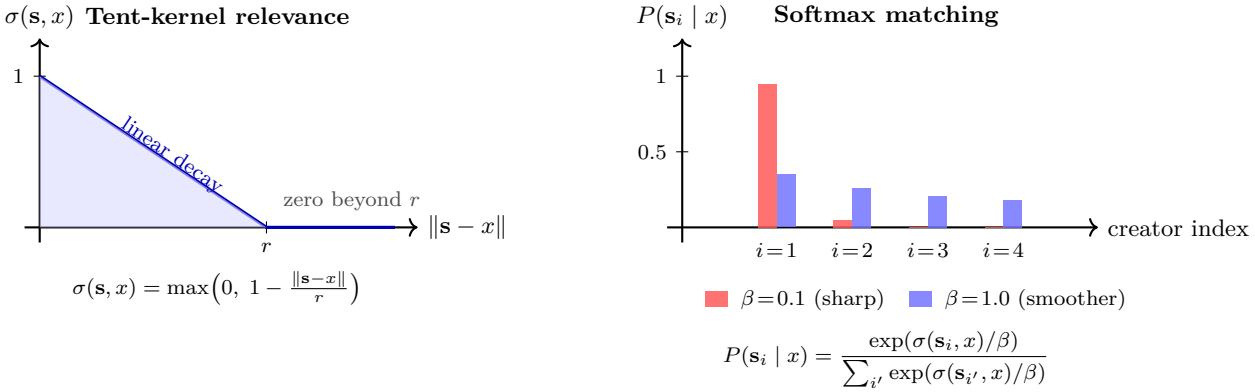


Figure 3: Visualizing the two mathematical primitives of the simulator's matching mechanism. **Left:** the tent-kernel relevance score (Eq. 2) decreases linearly from 1 at zero distance to 0 at distance r, and remains exactly 0 beyond. This formalizes the intuition that a creator whose content is very different from a user's interest provides no expected match quality. **Right:** given four hypothetical creators ranked by relevance ($\sigma_1 = 0.8$, $\sigma_2 = 0.5$, $\sigma_3 = 0.3$, $\sigma_4 = 0.1$), the softmax matching distribution (Eq. 3) concentrates mass sharply on the most-relevant creator when the temperature β is small (red, $\beta = 0.1$, our default) and distributes attention more evenly when β is larger (blue, $\beta = 1.0$). The low-temperature regime mirrors how real recommender systems aggressively favor top-relevance candidates without becoming fully deterministic.

Per-epoch dynamics. Each epoch e proceeds in three stages:

1. **Matching and welfare computation.** For each user x_j, the simulator computes the matched creator $\mathbf{s}_j^{(e)}$ by sampling from the softmax distribution (3). The realized welfare for user j is the relevance of the match, $\sigma(\mathbf{s}_j^{(e)}, x_j)$. The per-group average utility is then

$$\bar{\pi}_l^{(e)} = \frac{1}{c_l} \sum_{j:g_j=l} \sigma\left(\mathbf{s}_j^{(e)}, x_j\right) \cdot P\left(\mathbf{s}_j^{(e)} \mid x_j\right), \tag{4}$$

and the platform-level welfare is the mean across users,

$$W^{(e)} = \frac{1}{m} \sum_{j=1}^{m} \sigma\left(\mathbf{s}_j^{(e)}, x_j\right) \cdot P\left(\mathbf{s}_j^{(e)} \mid x_j\right). \tag{5}$$

While both $\bar{\boldsymbol{\pi}}^{(e)}$ and $W^{(e)}$ are mathematically defined here, they play distinct roles in the game. The per-group utilities $\bar{\boldsymbol{\pi}}^{(e)} \in \mathbb{R}^L$ are what the platform *observes* between epochs and in real-world deployment this corresponds to engagement metrics (watch-time, completion rate, click-through rate, dwell time) aggregated per user segment, which are standard analytics that every major platform tracks. The platform-level welfare $W^{(e)}$ is the *objective* the platform aims to maximize across an episode, and is mathematically derivable from the observation via $W^{(e)} = \sum_l (c_l/m)\, \bar{\pi}_l^{(e)}$. The per-group breakdown is informationally richer than the scalar $W^{(e)}$ since it tells the platform *which* groups are under-served, and therefore where to direct creator attention via the next-epoch weight $\mathbf{w}^{(e+1)}$. Yao's UIR rule and all of our learned controllers consume $\bar{\boldsymbol{\pi}}^{(e)}$ for exactly this reason.

This modeling choice does make two implicit simplifications relative to real-world conditions. First, we assume the platform knows the cluster assignment g_j for each user. In practice, the segmentation is itself a design choice that introduces error (users mis-clustered, segments that don't cleanly capture interest boundaries). Second, we treat the per-group engagement signal as noise-free; real engagement metrics carry sampling noise (particularly for small segments like our 10-user niche clusters). A more realistic treatment would model noisy and miscategorized observations of $\bar{\boldsymbol{\pi}}^{(e)}$. We discuss this as a direction for future work in Section 8.

2. **Creator response (LBR).** Each creator $\mathbf{s}_i$ independently performs K_{LBR} rounds of Local Better Response. In each round, the creator samples a small neighborhood of candidate next positions $\{\tilde{\mathbf{s}}_k\}$ (just 1 in our main experiments) at a fixed step size around their current $\mathbf{s}_i$, evaluates each candidate's expected $\mathbf{w}$-weighted utility $\sum_l w_l \tilde{\bar{\pi}}_l(\tilde{\mathbf{s}}_k)$ under the matching that would result, and adopts the candidate that yields the highest score (or stays put if none improves on the current position). LBR is a bounded-rationality search rule that generalizes both best-response and gradient-ascent updates while avoiding the assumption that creators have full visibility into the welfare landscape.

3. **Platform feedback.** The platform observes $\bar{\boldsymbol{\pi}}^{(e)} \in \mathbb{R}^L$, the per-group average utilities, and chooses the next weight vector $\mathbf{w}^{(e+1)}$ via the controller under test (Yao's UIR, a learned RL policy, CMA-ES, etc.). The next epoch begins with this updated weight vector.

5.2 Simulator Implementation and Hardware Considerations

We implement the simulator in Python as an environment with the following structure. User positions and cluster assignments are sampled once per episode in `Simulator.reset()`. The `Simulator.run_epoch(w, K_LBR)` method runs the three per-epoch stages above and returns a dictionary containing the per-group utilities, per-user match qualities, the realized welfare, and other diagnostics. The wrapper `YaoEnv` maintains episode state (the current weight vector, the past four epochs' $\bar{\boldsymbol{\pi}}$ history, the current epoch index) and exposes a standard `step(action)` interface that maps a controller-provided action $\mathbf{a}_e$ to the next weight vector according to a chosen action parameterization (Section 6). To support vectorized rollouts for PPO, we additionally implement `SyncVectorYaoEnv`, which runs multiple independent simulator instances in parallel and aggregates their observations and rewards into a single batched interface.

The simulator's compute scales as $O(nmd)$ per epoch since the matching step computes a relevance score for every (user, creator) pair, which at our standard configuration ($n = 200$, $m = 2{,}000$, $d = 5$) is 4×10^5 pair evaluations per epoch. A full $T = 200$ epoch episode takes roughly 1–2 seconds on a single CPU core under our NumPy implementation. This makes $\sim 10^5$ environment steps reachable in a few hours of wall-clock per RL training run, which is what bounds the training budgets we report in Section 6.

We run the entire training pipeline, both the simulator and the neural-network policy updates, on a single CPU. This is a deliberate choice motivated by the structure of on-policy RL: PPO repeatedly alternates between environment steps and policy forward passes, so a GPU-trained policy would incur a CPU→GPU→CPU round-trip per transition (the environment runs on CPU, the policy on GPU). At our 128-step × 8-environment rollout granularity, this host-device transfer overhead would dominate the actual forward-pass time. Our policy and value networks are small (2 hidden layers × 64 units; see Section 6), so a single-core CPU forward pass is fast and comparable in wall-clock to the environment step. The same economics apply to SAC, TD3, and CMA-ES and for the off-policy methods, environment steps and gradient updates are decoupled, but the small network size still makes CPU inference gap against GPU small after factoring in the fact simulation of transitions still happens in the CPU so there are still host-device transfers that happen and for CMA-ES, there is no neural network at all and since we rely on a library, its implementation in GPU seems unlikely. In general, running the full pipeline on CPU eliminates host-device synchronization and communication overhead without sacrificing throughput. Figure 4 makes the timing differences more concrete.

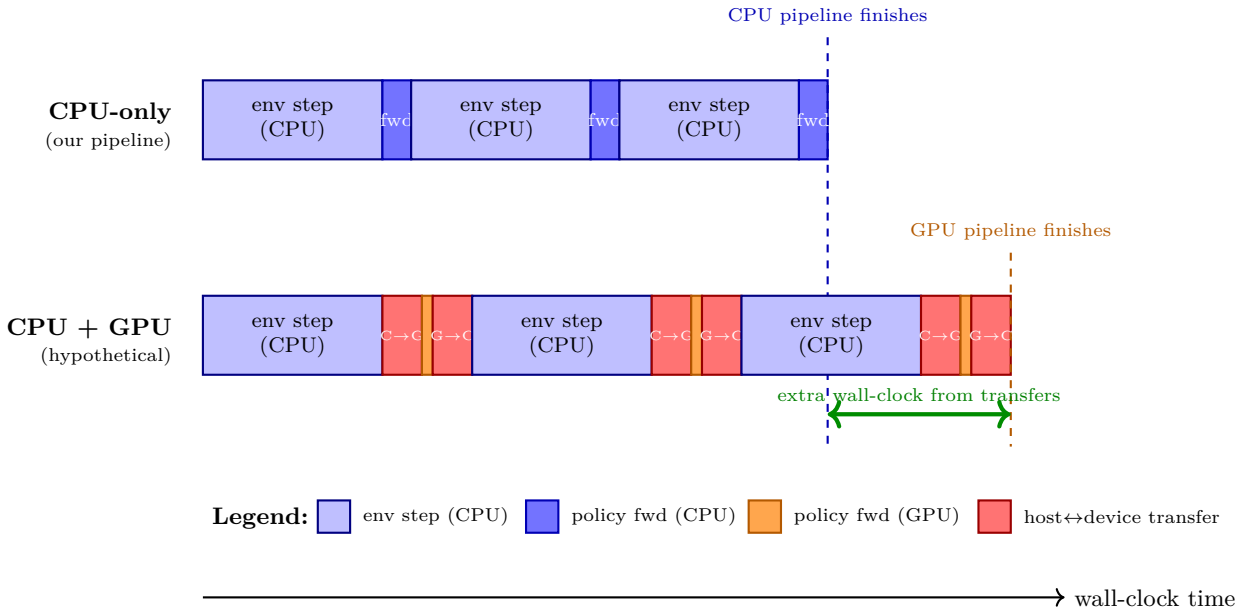


Figure 4: Why we run the entire training pipeline on CPU. **Top:** in our CPU-only setup, each training iteration is just an environment step (matching + LBR computation, the dominant cost) followed by a policy forward pass on the same device, and the time is less due to relatively small matrix-multiplications so no data leaves the CPU. **Bottom:** a hypothetical GPU-trained policy would require a CPU→GPU transfer of the observation before each forward pass and a GPU→CPU transfer of the action back to the environment. Although the GPU forward pass itself is faster than the CPU one (narrower orange bar), the two transfers per iteration more than cancel out the savings for our small (2×64 unit) networks. Three iterations finish on CPU before the GPU pipeline does, and the green arrow marks the accumulated synchronization overhead we avoid by keeping everything on CPU. Box widths are proportional but not to scale, the main point is that the environment step actually dominates both pipelines by roughly an order of magnitude over the forward-pass and transfer costs.

5.3 Differences from Yao et al.'s Reference Implementation

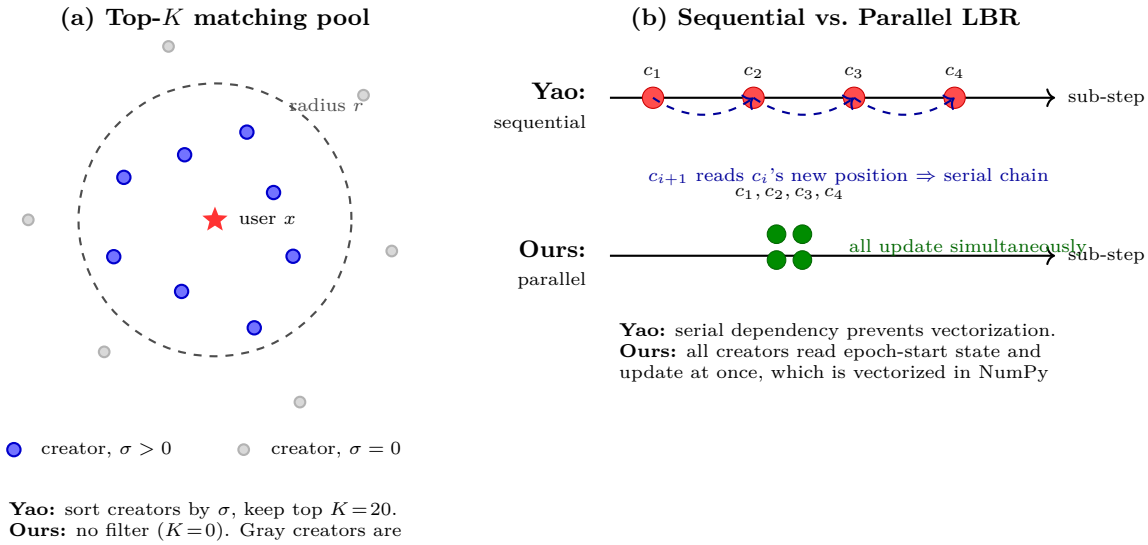


Figure 5: Two implementation choices that differ from the reference implementation of Yao et al. [2024]. **(a)** For the matching softmax, Yao restricts the pool to each user's top-K creators by relevance, requiring a per-user sort with cost $O(nm \log n)$. We disable the filter ($K = 0$). The tent kernel's hard cutoff at radius r already zeroes out the relevance of all creators outside the dashed circle (gray dots), so the unfiltered softmax assigns them zero probability mass either way and the two matching distributions agree to within numerical noise at our population scale. **(b)** For the per-epoch LBR step, Yao updates creators sequentially within an epoch: creator i's decision depends on the just-updated positions of creators $1, \ldots, i-1$, creating a serial dependency chain (dashed blue arrows). We instead update all n creators in parallel from epoch-start positions, removing the dependency chain and admitting a vectorized NumPy implementation that runs several times faster. Both choices preserve the LBR fixed points and the analytical-rule guarantees of Yao et al. [2024].

Symbol	Value	Description
L	10	Number of user clusters / weight-vector dimension
n	200	Number of strategic creators
m	2,000	Total number of users
d	5	Embedding dimension
$\mathbf{c}$	$(1000, 500, 200, 100, 100, 50, 20, 10, 10, 10)$	Per-cluster user counts
σ_l (`cluster_std`)	0.5	Per-cluster Gaussian spread
r (`sigma_radius`)	3.0	Tent-kernel radius (Eq. 2)
β	0.1	Softmax matching temperature
K	0	Top-K matching filter (0 = disabled)
LBR step	0.2	Per-iteration creator move magnitude
K_{LBR}	5	LBR iterations per epoch
T	200	Episode length in epochs
$\alpha_{\text{start}}, \alpha_{\text{end}}$	$0.5, 0.1$	Yao UIR step size (decays at $T/2$)
$w_{\min}, w_{\max}$	$0.2, 5.0$	Per-coordinate weight clip range
`creator_init`	near_largest_cluster	Initial creator positions
`creator_init_std`	0.3	Spread around initial cluster

Table 2: Standard environment configuration. After each controller's update, weights are renormalized so that $\sum_l w_l = L$, fixing the mean weight at 1.

Our simulator differs from the reference implementation described in Yao et al. [2024] along two implementation choices, both of which trade off small fidelity to the reference setup for substantially faster training. We document and justify each below as well as visualize the differences in Figure 5.

Top-K matching. Yao et al. restrict the softmax matching of Eq. (3) to each user's K most relevant creators (typically $K = 20$ in their experiments), with all other creators excluded from the softmax pool. This requires sorting all n creators by relevance score for each of the m users every epoch which is an $O(nm \log n)$ operation that dominates the per-epoch wall-clock cost at the populations we run. We disable top-K filtering by setting $K = 0$, which keeps every creator in the softmax pool for every user. This is justified by the structure of the softmax and the tent kernel: a creator more than the kernel radius r from a user contributes zero relevance and therefore zero probability mass under the softmax, regardless of whether they are formally excluded. The top-K filter is in effect a computational optimization for populations large enough that the unfiltered softmax becomes expensive; at our scale, removing it changes the matching distribution by less than the per-epoch noise floor while reducing simulator wall-clock substantially.

Parallel vs. Sequential creator updates. Yao et al.'s reference implementation updates creators *sequentially* within each epoch: creator i's LBR step uses the current epoch's already-updated positions for creators $1, \ldots, i-1$ and the previous epoch's positions for creators $i+1, \ldots, n$. We instead update all creators in *parallel*: every creator's LBR step uses the previous epoch's positions for all other creators, and all n creators commit their moves simultaneously at the end of the epoch. This batched update admits vectorized implementation in NumPy and is several times faster than the serial version, which we found essential for running the $\sim 10^5$ environment steps required by the deep-RL training budgets. The parallel update is also more faithful to real-world platforms, where creators do not in practice observe each other's content adjustments within a single decision cycle: a real-world "epoch" corresponds to whatever time window the platform aggregates engagement signals over (e.g. a day or a week), and creators make their next-content decisions based on the state of the system at the start of that window rather than on each other's intra-window adjustments.

Together these two choices preserve the structure of the C^3_{ext} game such as the welfare dynamics, the LBR fixed points, and the analytical-rule guarantees of Yao et al. [2024], while reducing simulator overhead enough to make deep-RL training practical on a single CPU.

5.4 Standard Environment Configuration

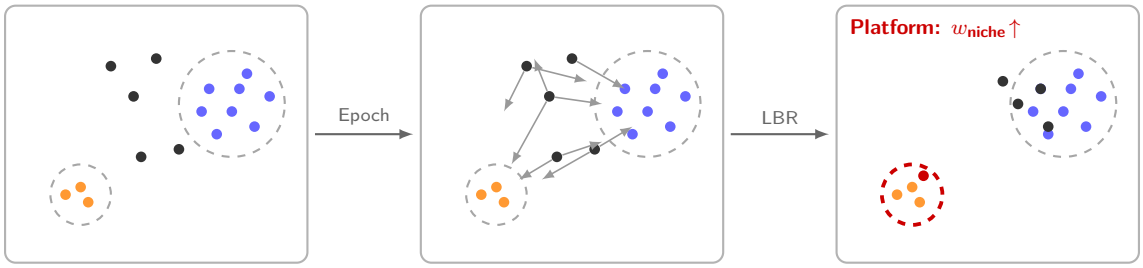
Our standard configuration is defined in `src/config.py` and serves as the "ideal" regime against which all stress environments are perturbations. We report the full hyperparameter list in Table 2.

The cluster sizes span two orders of magnitude cluster 1 holds 1,000 users (50% of the population), while the three smallest clusters hold only 10 users each (0.5% each). This produces a strong popularity gradient: cluster 1 is the natural destination for a profit-maximizing creator population, and the three smallest clusters are precisely the niche audiences Yao's UIR works hardest to re-attract creator attention to. The relatively low softmax temperature $\beta = 0.1$ makes the matching meaningfully concentrated on each user's most relevant creators without collapsing to deterministic best-match, and the $d = 5$ embedding

dimension is small enough that creator positions remain interpretable while still allowing more than two geometric degrees of freedom.

5.5 Mechanism Visualization

To make the per-epoch dynamics concrete, Figure 6 illustrates one epoch of the C^3_{ext} game as a three-panel storyboard. Users sit in their respective clusters (colored dots) and creators are scattered in the embedding space (black dots). Each creator evaluates K_{LBR} candidate moves under the current $\mathbf{w}$-weighted utility and commits to the best one. The platform then observes the resulting $\bar{\boldsymbol{\pi}}$ and raises w_l on the under-served clusters for the next epoch.



(a) Users in L Gaussian clusters; creators at $\{\mathbf{s}_i\}$

(b) Each creator scores $\sum_l w_l \bar{\pi}_l$ at K candidates

(c) Most pile on the popular cluster; platform raises w_l on the underserved one

Figure 6: One epoch of the C^3_{ext} game. **(a)** Users in L Gaussian clusters and creators are scattered in the embedding space. **(b)** Each creator evaluates K_{LBR} candidate moves and adopts the one maximizing $\sum_l w_l \bar{\pi}_l$. **(c)** Most creators move toward the largest cluster. The platform observes that the niche cluster is under-served and raises w_{niche} for the next epoch, redirecting creator attention.

5.6 Stress Environments

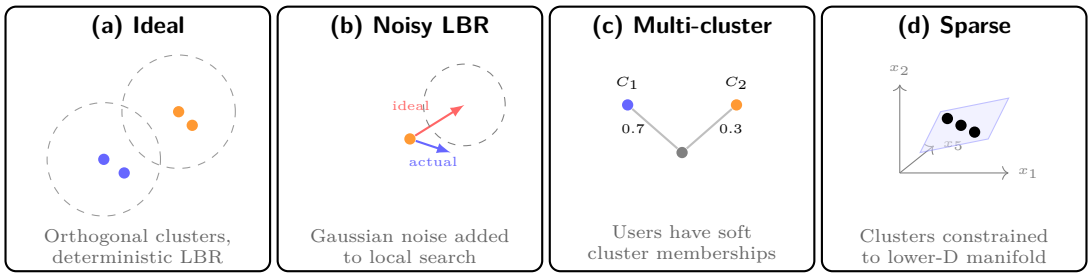


Figure 7: Visualization of the three stress environments introduced to break the assumptions of the analytical Yao UIR solution. While the **Ideal** (a) setting features strictly disjoint user clusters and perfectly rational creators, the stress environments inject realism via **Noisy LBR** (b) creator movement, **Multi-cluster** (c) soft user memberships, and **Sparse-embedding** (d) where $L = 10$ clusters are embedded onto a $K = 2$ dimensional manifold within a 5D feature space.

The three idealizations that Yao et al.'s analytical guarantee requires such as orthogonal cluster centers, single-cluster user membership, and noise-free LBR are precisely the structural properties that real recommender environments routinely violate. We constructed three stress environments, each of which selectively ablates *one* of these idealizations while leaving the others intact. This allows us to attribute any welfare gap between RL and Yao's UIR to the specific assumption (or multiple assumptions) that has (or have) been broken. It turns out that Noisy LBR can be reduced to just higher variance LBR hence due to compute constraints did not pursue experimenting on it further. It is a functionality the simulator has however. Furthermore, Figure 7 visualizes these non-idealities that we considered.

5.6.1 Noisy LBR creators

In the noisy-LBR variant, each creator's LBR move is perturbed by Gaussian noise after the best-candidate selection:

$$\mathbf{s}_i^{(e+1)} = \mathbf{s}_i^{\text{LBR}} + \boldsymbol{\eta}_i, \quad \boldsymbol{\eta}_i \sim \mathcal{N}(0, \sigma_\eta^2 I). \tag{6}$$

This models the fact that real-world creators do not deterministically converge on the locally-best content and they explore, experiment, and respond to short-term randomness in their feedback. The per-epoch $\bar{\boldsymbol{\pi}}$ signal the platform observes is correspondingly noisier than under deterministic LBR, which stresses Yao's analytical update because its $\bar{\boldsymbol{\pi}}^{(e)}$-dependent rule may over-react to the transient noise component.

5.6.2 Multi-cluster user membership

In the multi-cluster variant, each user has *soft* membership across multiple clusters: a primary cluster g_j^{primary} and one or more partner clusters $g_j^{\text{partner},k}$. Note that the code supports an arbitrary number of secondary clusters per user via the `multi_modal_n_secondary` parameter, and our reported multi-cluster results use `multi_modal_n_secondary = 3`, so each user has soft membership across one primary and three secondary clusters (four clusters total), with the primary receiving weight $1 - 0.6 = 0.4$ and the three secondaries each receiving $0.6/3 = 0.2$. The secondary clusters are sampled for each user with niche bias toward smaller clusters via inverse-size weighting ($w_{\text{partner}} \propto 1/c_l$).

At each epoch, the user's query point is resampled from a *categorical mixture* over their active clusters:

$$x_j^{(e)} \sim \mathcal{N}\big(\boldsymbol{\mu}_{l_j^{(e)}},\, \sigma^2_{\text{cluster}} I\big), \quad l_j^{(e)} \sim \text{Cat}\big(\boldsymbol{p}_j\big), \tag{7}$$

where the per-user categorical distribution $\boldsymbol{p}_j$ has mass $1 - \xi$ on the primary cluster and ξ/K_{sec} on each of the $K_{\text{sec}} = 3$ partner clusters, with mixture parameter $\xi = 0.6$ set via `multi_modal_mix`. This models real-world users who do not belong to a single interest category, for instance a user interested in both classical music and cooking pulls queries from both regions of the embedding space across different sessions, with the relative frequency given by $\boldsymbol{p}_j$. Yao's UIR rule treats each user as belonging entirely to its primary cluster, which discards information about the partner-cluster contribution to that user's welfare.

5.6.3 Sparse-embedding cluster geometry

In the sparse-embedding variant, the cluster centers are not distributed orthogonally in the full d-dimensional embedding space. The non-orthogonality constraint can be manifested in many ways, we chose to also show that when dynamics are more deterministic RL performance improves so instead, they lie in a 2-dimensional sub-manifold even though $d = 5$. Concretely, we place the $L = 10$ cluster centers in the first two coordinates of $\mathbb{R}^5$ and set the remaining three coordinates to zero. The creator and user populations live in the full 5-dimensional space, but the welfare-relevant geometry is degenerate: any creator who moves in the direction of the cluster centers does so by changing only the first two coordinates of $\mathbf{s}_i$. This violates Yao's orthogonality assumption hardest, because the centers are now correlated in the projection-onto-clusters subspace.

6 Methods

We extend Yao et al.'s C^3_{ext} framework in three ways that the analytical UIR rule does not address. **(i)** We cast the platform's intervention as a Markov Decision Process (MDP), enabling state-conditional weight updates. **(ii)** We introduce a *Yao-residual* action parameterization that warm-starts the policy at Yao's analytical solution and lets it learn a small correction on top, which we find to be essential for making deep RL tractable in this regime. **(iii)** We train four controllers spanning two algorithmic design axes: on-policy vs. off-policy and stochastic vs. deterministic actor, plus a gradient-free baseline. This section describes the MDP formulation, the two action parameterizations, each of the four controllers in turn (including the training loops and all algorithmic tricks they employ), and a consolidated hyperparameter summary.

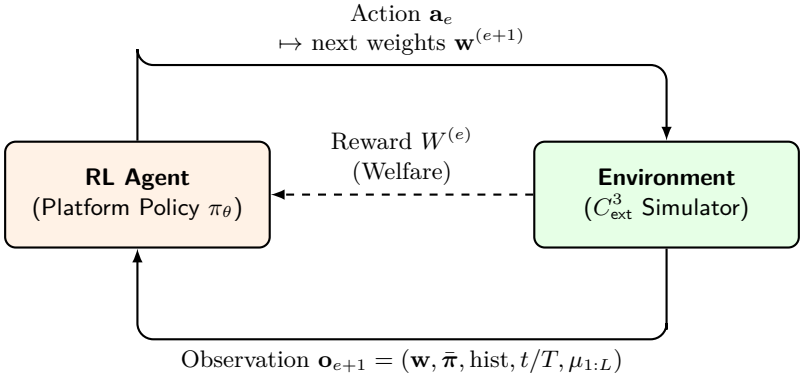


Figure 8: The Deep RL problem formulation for the C^3_{ext} game. At each epoch e, the agent observes the system state $\mathbf{o}_e$, including the current weights, historical aggregate user utilities $\bar{\boldsymbol{\pi}}$, and cluster demographics. It outputs an action $\mathbf{a}_e$ mapped to the new weighting vector $\mathbf{w}^{(e+1)}$. The environment steps forward by running creator LBR dynamics and user matching, returning the next state and a reward equal to the platform's overall welfare $W^{(e)}$.

6.1 MDP Formulation

We formulate Yao's game as a Markov Decision Process (MDP) as shown in Figure 8. At each epoch $e \in \{0, 1, \ldots, T-1\}$ the platform observes a state vector

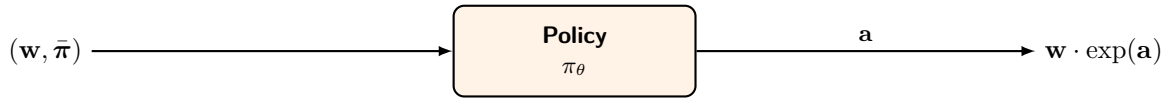
$$\mathbf{o}_e = \big(\mathbf{w}^{(e)},\ \bar{\boldsymbol{\pi}}^{(e)},\ \mathrm{hist}_4^{(e)},\ t/T,\ \boldsymbol{\mu}_{1:L}\big), \tag{8}$$

where $\mathbf{w}^{(e)} \in \mathbb{R}^L$ is the current weight vector, $\bar{\boldsymbol{\pi}}^{(e)} \in \mathbb{R}^L$ is the per-group utility from the previous epoch (Eq. 4), $\mathrm{hist}_4^{(e)} \in \mathbb{R}^{4L}$ stacks the past four epochs' $\bar{\boldsymbol{\pi}}$ vectors, $t/T \in [0, 1]$ is the fraction of the episode completed, and $\boldsymbol{\mu}_{1:L} \in \mathbb{R}^{L \times d}$ are the cluster centroids. The cluster centroids are constant within an episode but vary across episodes (the user population is resampled at the start of each rollout). Importantly, none of hist_4, t/T, or $\boldsymbol{\mu}$ are consumed by Yao's UIR rule so they are signals our controllers have access to that the analytical baseline does not.

We note that the choice of observation $\mathbf{o}_e$ is itself a design axis that meaningfully affects training outcomes. In pilot experiments we ablated several state representations like omitting the cluster centroids $\boldsymbol{\mu}_{1:L}$, varying the history window depth from the 4-epoch buffer above, and including additional features such as per-cluster creator-position summaries and found measurable welfare differences between observation formulations that often exceeded the welfare differences between RL algorithms run on the same observation. This suggests that state representation, not the choice of RL algorithm, is the more promising direction for further welfare gains in this regime. We return to this point with quantitative evidence in Section 7 and discuss richer observation spaces (e.g., learned encoders of cluster geometry, pairwise creator distances) as a direction for future work in Section 8.

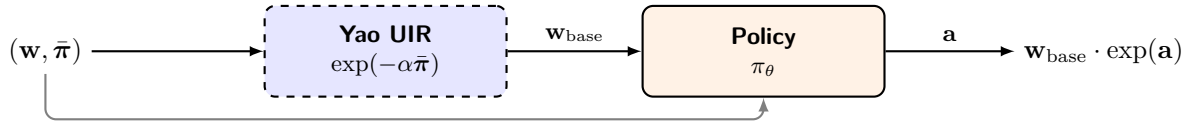
The policy $\pi_\theta : \mathbf{o}_e \mapsto \mathbf{a}_e$ outputs an action $\mathbf{a}_e \in \mathbb{R}^L$ which is then mapped to the next weight vector via one of two parameterizations (Section 6.2). The action is clipped to $[-c_a, c_a]$ before being applied, where $c_a = 1.0$ this bounds the per-epoch multiplicative change to a factor of $\exp(c_a) \approx 2.7$ in either direction. We employ this clipping to again try to reduce the variance in the actions taken and to also not take "too-drastic" up- or down-weighting. **The reward at each epoch is the mean user welfare** $W^{(e)} = \frac{1}{m}\sum_j \sigma(\mathbf{s}_j^{(e)}, x_j) \cdot P(\mathbf{s}_j^{(e)} | x_j)$ defined in Section 5.1, and the episode return $\sum_e \gamma^e W^{(e)}$ approximates the discounted long-run welfare under a per-epoch discount $\gamma = 0.95$.

6.2 Two Action Parameterizations



(a) Multiplicative

Policy learns the full update



(b) Yao-residual

Policy learns a correction on top of Yao

Figure 9: Comparison of the Multiplicative and Yao-residual action parameterizations. Both architectures map the policy action $\mathbf{a}_e$ to the next weight vector $\mathbf{w}^{(e+1)}$ within the family of exponential reweighting updates. In the **Multiplicative** approach (a), the policy is entirely responsible for generating the update. In the **Yao-residual** approach (b), the policy outputs a correction factor over the analytical Yao base step. The Yao-residual mode relies on near-zero final-layer initialization to warm-start the agent with Yao UIR behavior, bypassing the poor convergence associated with learning the multiplicative update from scratch. Both parameterizations strictly enforce bounds via identical clipping and renormalization steps.

As dislayed in Figure 9, we compare two parameterizations of how $\mathbf{a}_e \in \mathbb{R}^L$ maps to the next weight vector. Both preserve $\mathbf{w}^{(e+1)} \succ 0$ through an exponential, and both renormalize and clip the result via

$$\mathbf{w}^{(e+1)} \leftarrow L \cdot \mathbf{w}^{(e+1)} / \textstyle\sum_l w_l^{(e+1)}, \quad w_l^{(e+1)} \in [w_{\min}, w_{\max}] = [0.2, 5.0], \tag{9}$$

matching the clip bounds Yao et al. use. The renormalization fixes $\sum_l w_l = L$ so the platform cannot trivially boost welfare by scaling all weights up, and the clip prevents the entropy-regularized policies (SAC) from exploiting unbounded actions as well as drastic weight changes.

Multiplicative. The policy learns the full weight update from scratch:

$$\mathbf{w}^{(e+1)} \propto \mathbf{w}^{(e)} \cdot \exp(\mathbf{a}_e). \tag{10}$$

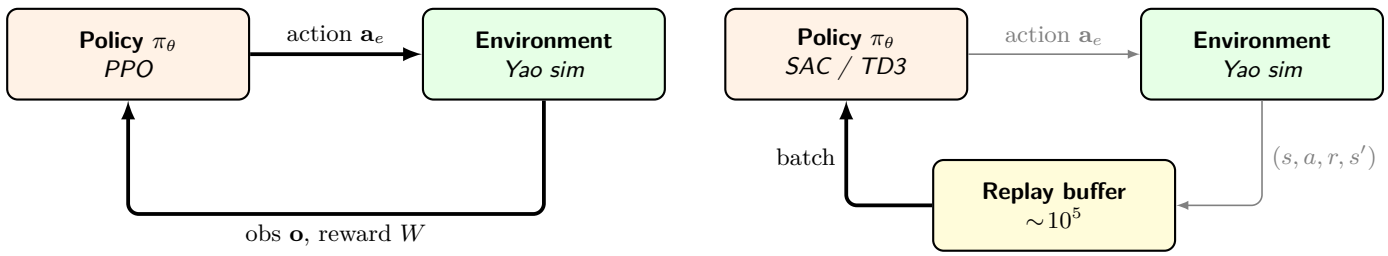
Yao-residual. The policy learns a correction on top of Yao's analytical update:

$$\mathbf{w}^{(e+1)} \propto \underbrace{\mathbf{w}^{(e)} \cdot \exp\left(-\alpha^{(e)}\, \bar{\boldsymbol{\pi}}^{(e)}\right)}_{\text{Yao's UIR step}} \cdot \exp(\mathbf{a}_e), \tag{11}$$

where the step size follows Yao et al.'s decay schedule, $\alpha^{(e)} = 0.5$ for $e < T/2$ and $\alpha^{(e)} = 0.1$ thereafter. Setting $\mathbf{a}_e = \mathbf{0}$ recovers Yao UIR exactly. To exploit this, we initialize the actor's final-layer weights with an *orthogonal* init with very small gain (`nn.init.orthogonal_(last.weight, gain=0.01)`) and bias zero, so the policy's initial action distribution is tightly centered at $\mathbf{a}_e \approx \mathbf{0}$. This means the *initial behavior of every deep-RL agent is approximately Yao UIR*. One caveat is that those weights are the final weights in a Yao simulation so not neccesarily the same beginning weights. The Yao-residual mode is the default for all reported deep-RL results as we saw vanilla deep RL under the multiplicative parameterization without warm-start converges to welfare below no-intervention within our training budget.

We also note that both parameterizations above fall within the family of *exponential reweighting* updates of the form $\mathbf{w}^{(e+1)} \propto \mathbf{w}^{(e)} \cdot \exp(\cdot)$, differing only in whether the policy produces the full update (multiplicative) or a residual on top of Yao's analytical base (Yao-residual). Their effective action spaces, the set of next-weight vectors the policy can produce as a function of state, are therefore structurally similar. Alternative parameterizations outside this family could give the policy access to qualitatively different update directions: learned step sizes that modulate Yao's α adaptively, additive corrections $\mathbf{w}^{(e+1)} = \mathbf{w}^{(e)} + \mathbf{a}_e$, or constrained projections onto the empirical gradient direction of welfare etc. Each of these would express updates the exponential family cannot represent, and any of them could plausibly extract additional welfare. We discuss these as a direction for future work in Section 8.

6.3 Overview and Choice of Methods



(a) On-policy (PPO)

Tight env loop, fresh rollouts per gradient update

(b) Off-policy (SAC, TD3)

Loose env loop, many updates per env step from buffer

Figure 10: Architectural comparison of the deep-RL methods evaluated, trade-off between stability and sample efficiency. **(a) On-policy (PPO):** Acts as a stable baseline where the policy interacts with the environment in a tight loop. Its trust-region structure prevents catastrophic divergence from the warm-start behavior, trading sample efficiency for stability. **(b) Off-policy (SAC, TD3):** Utilizes a loose environment loop by storing past transitions in a replay buffer. This allows the policy to reuse data across many gradient updates, drastically improving sample efficiency. Within this off-policy structure, SAC encourages exploration via entropy regularization, while TD3 deploys a deterministic actor and twin critics for explicit gradient-variance reduction.

We selected three methods that span the continuous-control RL design space along two principal axes: *sample efficiency* and *gradient variance*. Both axes matter specifically for the C^3_{ext} environment, where the per-epoch welfare signal is noisy (stochastic matching, randomized LBR creator response, seed-to-seed variance in user populations) and the training budget is small ($\sim 10^3$–10^5 env steps depending on method). Different algorithms make different trade-offs along these axes, and we wanted to test which class is best suited to this regime since initially we believed perhaps plugging-and-playing with algorithms could lead to drastic differences.
We motivate each choice below.

	Stochastic actor	Deterministic actor
On-policy	PPO	–
Off-policy	SAC	TD3

Table 3: The four deep-RL methods positioned on the on/off-policy and stochastic/deterministic-actor design axes. CMA-ES (not shown) is a gradient-free evolutionary baseline that operates outside this design space.

PPO: the stable on-policy baseline. PPO is the most widely used continuous-control method and the algorithm least likely to catastrophically diverge in our setting. The clipped-surrogate through the "trust-region" structure means a single bad gradient step cannot move the policy too far from its warm-started near-Yao behavior which we thought would be useful when learning a small residual on top of an already-strong analytical solution, where the worst-case risk is destabilizing the warm-start. In this sense, PPO trades sample efficiency for stability since it cannot reuse transitions across rollouts, so we expected it to provide a reliable floor that matches Yao but is capped by its rollout budget on stress environments where the optimal residual is harder to find.

SAC: off-policy sample efficiency with entropy regularization. Our SAC implementation adds two ingredients PPO lacks: off-policy data reuse via a replay buffer (every transition contributes to many gradient updates), and an entropy regularization term that explicitly encourages exploration of the residual action space. The latter of which may have not been too helpful since the dynamics of the environment are already quite stochastic. Initially, we expected SAC to converge faster than PPO and potentially find better residuals on stress environments where the optimum is non-trivial to locate at the cost of a small welfare hit on the ideal env, where the entropy term keeps the policy stochastic even when the optimal action converging to be near-deterministic.

TD3: deterministic actor with explicit variance reduction. Our TD3 implementation deploys three explicit variance-reduction techniques simultaneously: twin critics with $\min(Q_1, Q_2)$ targets (reduces overestimation bias), target-policy smoothing (regularizes the critic against sharp peaks), and delayed policy updates (lets the critic partially converge before the actor moves). Combined with a deterministic actor (lower-variance policy gradient than a stochastic actor's score-function gradient), TD3 has the lowest gradient variance of any method we test. Given the inherent stochasticity of the C^3_{ext} env, we hypothesized that gradient-variance reduction would be a dominant factor in this regime, and TD3 lets us test that hypothesis directly. Whether this hypothesis bears out empirically is exactly the question Section 7 addresses.

CMA-ES: gradient-free open-loop ceiling. CMA-ES is structurally different from the other three methods: it has no policy network, no temporal structure, and produces a *constant* weight vector $\mathbf{w}^\star$. We include it not as a competing controller but to answer a specific question: *is there state-conditional welfare to recover beyond the best constant rule?* CMA-ES finds the best static $\mathbf{w}$ in the basin around Yao's UIR fixed point so the welfare gap between CMA-ES and any deep-RL method is therefore exactly the gain from conditioning the weight update on observed state.

Together, these four methods let us isolate how much of any observed welfare gap is attributable to (a) sample efficiency (PPO vs. SAC, TD3), (b) gradient variance reduction (TD3 vs. SAC, PPO), and (c) the very fact of state-conditioning (deep RL vs. CMA-ES). The empirical pattern we observe and discuss in Section 7 shows that variance reduction proves to be the dominant axis among the deep-RL methods, more so than the on/off-policy distinction. A summary of the different methods can be found in Table 3 and Figure 10

6.4 PPO

PPO is the on-policy, stochastic-actor method in our comparison (Section 6.3).

Objective. PPO maximizes the clipped importance-ratio surrogate of [Schulman et al., 2017],

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}_t\Big[\min\big(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\big)\Big], \quad r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, \tag{12}$$

with $\epsilon = 0.2$. The clipping bounds how far the policy can shift in a single update by capping the importance ratio, which serves as a first-order proxy for a trust-region constraint.

Advantage estimation. We estimate advantages with Generalized Advantage Estimation (GAE) [Schulman et al., 2016],

$$\hat{A}_t = \sum_{l=0}^{T_{\text{roll}}-t-1} (\gamma\lambda)^l \delta_{t+l}, \quad \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t), \tag{13}$$

computed backward through each independent environment's trajectory. We use $\lambda = 0.95$, which weights the temporally distant rewards heavily and also a higher-λ choice that we found stabilized training relative to lower values. Advantages are normalized to zero mean and unit variance *per minibatch* before the gradient step, which is a PPO trick that we employed and we saw further reduces gradient scale variance across updates.

Architecture. The actor is a separate MLP with two hidden layers of 64 units (using tanh activations), with a final linear head outputting a mean $\mu_\theta(s) \in \mathbb{R}^L$. The standard deviation of the Gaussian policy is parameterized by a *global learnable* log-std vector $\log \sigma \in \mathbb{R}^L$ initialized at $\log \sigma_0 = -1.0$ (initial action noise $e^{-1.0} \approx 0.37$). We relied on the global rather than state-dependent log-std because it is the standard choice for continuous-control PPO since it has fewer parameters than a state-dependent log-std head and is empirically more stable in our setting (our SAC has per action standard-deviation). The critic is a separate MLP with the same hidden architecture and a scalar value head. The final layer of the actor's mean head is initialized with orthogonal init at gain 0.01 and zero bias, implementing the warm-start to $\mathbf{a}_e \approx \mathbf{0}$ that Section 6.2 describes.

Reward signal. At each environment step (one epoch of the simulator), the agent receives the platform-level welfare $r_e = W^{(e)}$ as defined in Eq. (5): the mean user-creator match quality across the entire user population, computed directly by the simulator. This is the per-step reward that enters PPO's GAE advantage estimation and the value-function bootstrap. We do not modify, normalize, or shape this reward so the policy learns directly to maximize cumulative welfare across the $T = 200$-epoch episode.

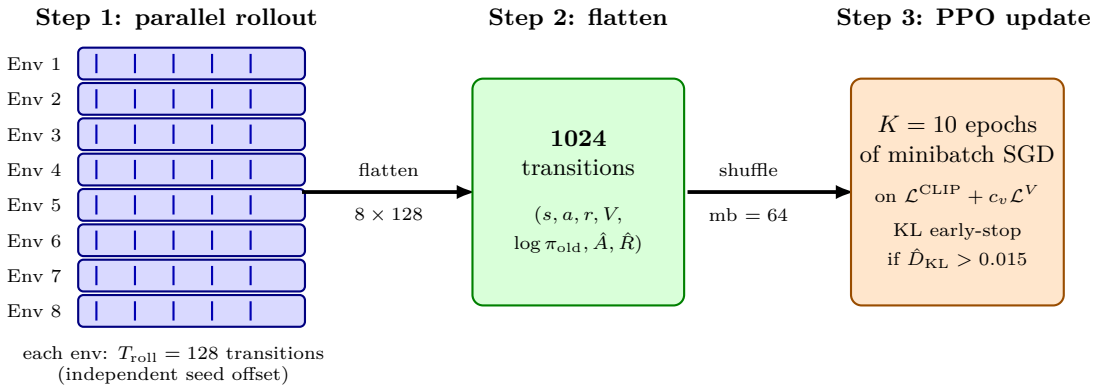


Figure 11: Vectorized PPO rollout collection. At each outer iteration, **Step 1** runs $n_{\text{env}} = 8$ independent simulator instances in parallel via `SyncVectorYaoEnv`, each with its own seed offset, collecting $T_{\text{roll}} = 128$ transitions per environment. **Step 2** flattens the (rollout, env) tensor into a single batch of 1,024 transitions, each tagged with the state, action, reward, value estimate, log-probability of the action under the rollout policy π_{old}, GAE advantage $\hat{A}$, and bootstrapped return $\hat{R}$. **Step 3** shuffles the batch and takes $K = 10$ epochs of minibatch SGD with minibatch size 64 on the composite loss $-\mathcal{L}^{\text{CLIP}} + c_v \mathcal{L}^V$, aborting the inner loop early if the empirical KL divergence exceeds $\hat{D}_{\text{KL}}^{\text{target}} = 0.015$.

Training loop. At each outer iteration, the algorithm:

1. Collects $T_{\text{roll}} = 128$ transitions in each of $n_{\text{env}} = 8$ vectorized environments running in parallel via `SyncVectorYaoEnv`. Each environment has its own seed offset so the populations and creator initializations are independent across environments. Vectorizing the rollouts is a *variance reduction* technique we employed: collecting transitions from n_{env} independent environment seeds in parallel reduces the gradient noise from any single environment's correlated trajectory by roughly $\sqrt{n_{\text{env}}}$, while parallel execution keeps wall-clock per iteration comparable to a single-environment rollout.

2. Computes GAE advantages backward through each environment's trajectory independently. It essentially allows us to have more than jsut a single monte-carlo sample trajectory affect the gradient at every rollout step. This mechanism is visualized in Figure 11

3. Flattens the (rollout, env) tensor to a batch of 1,024 transitions and takes $K = 10$ epochs of minibatch SGD with minibatch size 64 on the composite loss

$$\mathcal{L}(\theta, \phi) = -\mathcal{L}^{\text{CLIP}}(\theta) + c_v \mathcal{L}^V(\phi) - c_e \mathcal{H}[\pi_\theta], \tag{14}$$

where $\mathcal{L}^V(\phi) = (V_\phi(s) - \hat{R})^2$ is the value loss, $\mathcal{H}[\pi_\theta]$ is the entropy bonus, and the weights are $c_v = 1.0$, $c_e = 0$. We use no entropy bonus ($c_e = 0$) because the global learnable $\log \sigma$ already provides exploration. Gradient norms are clipped at 0.5 before the Adam optimizer step ($\texttt{lr} = 3 \times 10^{-4}$).

4. After each inner epoch, computes the empirical approximate KL divergence $\hat{D}_{\mathrm{KL}}(\pi_{\mathrm{old}} \,\|\, \pi_\theta) = \mathbb{E}[(r-1) - \log r]$ where $r = \pi_\theta(a|s)/\pi_{\mathrm{old}}(a|s)$, and *aborts the inner loop* if it exceeds the target threshold $D_{\mathrm{KL}}^{\mathrm{target}} = 0.015$. This KL early-stop safeguards against catastrophic policy collapse if the clipped surrogate's first-order constraint is violated, but is rarely triggered in practice in our runs.

We run $N_{\mathrm{updates}} = 200$ outer iterations per training run, evaluating on a held-out environment every 10 updates and saving the best-evaluating checkpoint via early-stopping.

6.5 SAC

SAC is our off-policy, stochastic-actor method (Section 6.3). We chose it for its sample-efficiency advantages from replay-buffer reuse and the exploration benefits of the entropy bonus.

Objective.

$$J(\pi) = \mathbb{E}_\pi\Big[\sum_e \gamma^e \big(r_e + \alpha\, \mathcal{H}(\pi(\cdot|s_e))\big)\Big], \tag{15}$$

where α trades exploitation against exploration. The soft Q-values are trained to fit the Bellman target

$$y = r + \gamma\,(1-d)\big[\min(Q_{\bar{\phi}_1}, Q_{\bar{\phi}_2})(s', a') - \alpha\, \log \pi(a'|s')\big]\,, \tag{16}$$

where $\bar{\phi}_1, \bar{\phi}_2$ are target network parameters, d is the done flag, and $a' \sim \pi(\cdot|s')$ is sampled fresh in the target computation. The $\min(Q_1, Q_2)$ is the **clipped double-Q** trick [Fujimoto et al., 2018]: using the minimum of two independent critics in the target combats the overestimation bias that a single bootstrapped Q-network would compound over training. Both Q_1 and Q_2 are trained jointly to minimize the squared Bellman error on y.

Architecture. The actor is a Gaussian policy with a shared MLP trunk (2 hidden layers $\times$ 64 ReLU) feeding two parallel linear heads, a mean $\mu_\theta(s)$ and a log-std $\log \sigma_\theta(s)$ clamped to $[-5, 2]$. Actions are produced via the **reparameterization trick**:

$$u = \mu_\theta(s) + \sigma_\theta(s) \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I), \quad a = c_a \cdot \tanh(u), \tag{17}$$

with action scale $c_a = 1.0$. The reparameterization gives a strictly lower-variance gradient than the REINFORCE score-function gradient: it lets us backpropagate through the sampled action by sampling the random noise ϵ instead of the action itself. The log-density of the squashed action requires the tanh-Jacobian correction.

$$\log \pi_\theta(a|s) = \log \mathcal{N}(u; \mu_\theta(s), \sigma_\theta(s)^2) - \sum_{l=1}^{L} \log\big(1 - \tanh^2(u_l)\big)\,, \tag{18}$$

which accounts for how the tanh squashes a unit interval of u into a smaller interval of a near the saturation regions $|u| \gg 1$. Without this correction, the gradient would systematically over-reward saturating actions since it would reward hack as the densities are not aligned. Figure 12 gives more intuition, essentially the correction allows the density modeled by the normal distribution to be sampled from the tanh distribution. This is good because the range of tanh is bounded, so our actions can be bounded while the actual probabilities are also aligned so the entropy-bonus is not gamed.

Finally, the critics are two independent MLPs (each: $\mathrm{cat}(s, a) \to$ 2 hidden layers $\times$ 64 ReLU $\to$ scalar), with separate target networks $Q_{\bar{\phi}_1}, Q_{\bar{\phi}_2}$ initialized identically to Q_1, Q_2 and whose gradients are detached.

Auto-tuned entropy. Following [Haarnoja et al., 2018], we treat $\log \alpha$ as a learnable scalar parameter optimized by minimizing

$$\mathcal{L}(\alpha) = \mathbb{E}_\pi\big[-\alpha\,(\log \pi(a|s) + \mathcal{H}_{\mathrm{target}})\big], \tag{19}$$

with $\mathcal{H}_{\mathrm{target}} = -L$ (i.e. negative of action dimension, the standard heuristic). This auto-tuning eliminates the need to tune α by hand instead, the only knob is the target entropy.

Reward signal. At each environment step, the agent receives the platform-level welfare $r_e = W^{(e)}$ as defined in Eq. (5). This reward is stored in the replay buffer alongside the transition (s, a, r, s', d) and enters the Bellman target through the soft Q-value update.

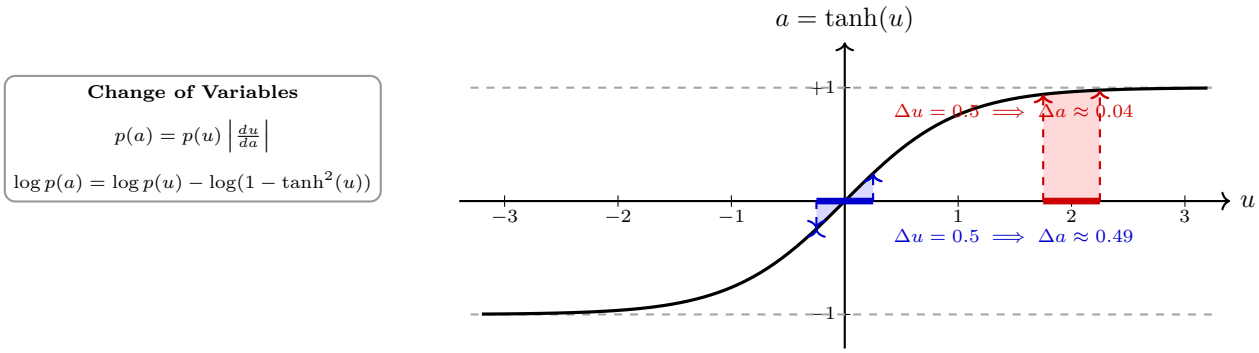


Figure 12: Why the tanh-Jacobian correction is needed. The solid curve shows $a = \tanh(u)$, mapping the unbounded pre-squash variable u to the bounded action $a \in (-1, +1)$. Two intervals of equal width $\Delta u = 0.5$ are highlighted on the u-axis: the blue interval near $u = 0$ maps to nearly the same width on the a-axis ($\Delta a \approx 0.49$), while the red interval near $u = 2$ is heavily compressed to a width of only $\Delta a \approx 0.04$. When a Gaussian density on u is transformed through tanh, the resulting density on a must therefore be *much higher* near the saturation boundaries $a \approx \pm 1$, where probability mass is squeezed into a smaller range. The Jacobian correction $-\log(1-\tanh^2(u))$ added to $\log \pi_\theta(a|s)$ is the log-density penalty that mathematically reverses this compression. Without it, the actor loss would systematically over-reward saturating actions.

Training loop. At each environment step:

1. For the first 200 *warmup steps*, sample actions uniformly from $[-1, 1]^L$ rather than from the policy. This populates the replay buffer with diverse initial transitions before policy gradient updates begin.
2. After warmup, sample $a_e \sim \tanh(\mu_\theta(s) + \sigma_\theta(s) \odot \epsilon)$ (stochastic, not the deterministic mean), step the environment, and store (s, a, r, s', d) in the replay buffer (capacity $|B| = 5 \times 10^4$ transitions).
3. Sample a minibatch of 64 transitions from B.
4. Update both Q-networks by minimizing $\mathcal{L}_{Q_i} = \mathbb{E}[(Q_{\phi_i}(s, a) - y)^2]$ for $i \in \{1, 2\}$, jointly with a single backward pass on the sum.
5. Update the actor by maximizing $\mathbb{E}[\min(Q_{\phi_1}, Q_{\phi_2})(s, a_\theta) - \alpha \log \pi_\theta(a_\theta|s)]$, where a_θ is freshly sampled via reparameterization at the current actor parameters.
6. Update $\log \alpha$ via its auto-tuning loss.
7. **Polyak averaging** the target networks: $\bar{\phi} \leftarrow (1-\tau)\bar{\phi} + \tau\phi$ with $\tau = 0.005$. The soft target update prevents the target from chasing the online network too aggressively, which would destabilize bootstrapping.

We use Adam (`lr` $= 3 \times 10^{-4}$ for actor, critic, and $\log \alpha$), running for $N_{\text{steps}} = 5{,}000$ env steps per training run with early-stopping on a held-out environment every 10 steps.

6.6 TD3

TD3 is our off-policy, deterministic-actor method (Section 6.3), chosen specifically to test the hypothesis that gradient-variance reduction is the dominant factor in this regime to get improvement in RL-based methods. Its design composes three explicit variance-reduction techniques discussed below.

Architecture. The actor is a deterministic mapping $\mu_\theta : s \mapsto a \in [-c_a, c_a]^L$ implemented as an MLP (2 hidden $\times$ 64 ReLU) with a tanh output and action scale c_a set by the environment. The deterministic policy is trained via the *deterministic policy gradient*:

$$\nabla_\theta J = \mathbb{E}\Big[\nabla_a Q_{\phi_1}(s, a)\big|_{a=\mu_\theta(s)} \cdot \nabla_\theta \mu_\theta(s)\Big]. \tag{20}$$

Note that only Q_{ϕ_1} is used in the actor gradient the second critic is not. The final layer of the actor is initialized with orthogonal gain 0.01, so the initial deterministic action is near zero.

The critic is a twin-Q architecture sharing the same input format as SAC's, with corresponding target networks $\bar{\phi}_1$, $\bar{\phi}_2$ and a target actor $\bar{\theta}$.

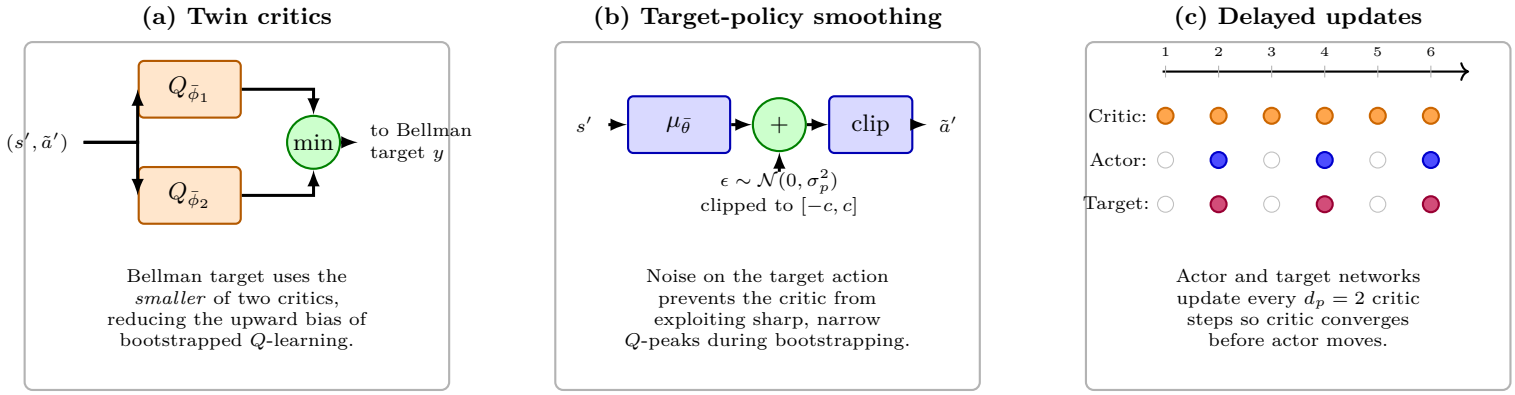


Figure 13: The three variance-reduction tricks that distinguish TD3 from DDPG. **(a)** *Twin critics:* two independently initialized Q-networks are trained jointly, and the Bellman target uses the minimum of their estimates, bounding the upward overestimation bias that single-network bootstrapping compounds across training. **(b)** *Target-policy smoothing:* the target action used in the Bellman update is perturbed by clipped Gaussian noise with $\sigma_p = 0.05$ and clip $c = 0.1$. The noise prevents the critic from exploiting narrow Q-peaks during bootstrapping, since neighboring actions must produce similar Q-values for the target to remain stable. **(c)** *Delayed policy updates:* filled circles mark when each component is updated. The critic updates every step (orange), while the actor and all target networks update only every $d_p = 2$ critic steps (blue and purple). This lets the critic partially converge before the actor moves, reducing the variance of the deterministic policy gradient. Together these three mechanisms give TD3 the lowest gradient variance among the deep-RL methods in our comparison.

The three TD3 tricks as shown in Figure 13

1. **Twin critics with clipped double-Q targets.** Train Q_1 and Q_2 jointly. The Bellman target uses $\min(Q_{\bar{\phi}_1}, Q_{\bar{\phi}_2})$, the same overestimation-bias fix as SAC. The critic loss is the sum of squared errors on both Q-networks, computed with a single backward pass.

2. **Target-policy smoothing.** The target action is perturbed by clipped Gaussian noise:

$$\tilde{a}' = \text{clip}(\mu_{\bar{\theta}}(s') + \text{clip}(\epsilon, -c, c),\ -c_a, c_a)\,, \quad \epsilon \sim \mathcal{N}(0, \sigma_p^2 I), \tag{21}$$

   with smoothing std $\sigma_p = 0.05$ and clip $c = 0.1$. This smoothing acts as a regularizer that stops the critic from exploiting sharp narrow Q-peaks: without it, a deterministic actor can drive Q to fit pathological local maxima that don't reflect the true expected return. So we essentially only learn toward regions that do not have sharp changes in the curve.

3. **Delayed policy updates.** The actor (and all three target networks) update only every $d_p = 2$ critic updates. This lets the critic partially converge before the actor moves, reducing the variance of the actor gradient. We follow [Fujimoto et al., 2018] in the choice of $d_p = 2$.

Exploration. Since the actor is deterministic, exploration must be injected externally:

$$a_e = \text{clip}(\mu_\theta(s_e) + \boldsymbol{\eta}_e,\ -c_a, c_a)\,, \quad \boldsymbol{\eta}_e \sim \mathcal{N}(0, \sigma_e^2 I), \tag{22}$$

where σ_e follows a linear decay schedule from $\sigma_{\text{init}} = 0.1$ to $\sigma_{\text{final}} = 0.02$ over the first 50% of training, then stays at σ_{final}. The decaying schedule explores broadly early and exploits the learned policy later. The relatively small σ_{init} relative to TD3's published default (0.1) reflects the warm-start to a near-optimal residual: aggressive exploration moves us away from a known-good region of the action space.

Reward signal. At each environment step, the agent receives the platform-level welfare $r_e = W^{(e)}$ as defined in Eq. (5). This reward is stored in the replay buffer and enters the Bellman target through the twin-Q update with $\min(Q_{\bar{\phi}_1}, Q_{\bar{\phi}_2})$ bootstrapping. No reward shaping is applied and the policy learns directly from the welfare signal.

Training loop. Same structure as SAC except: (a) no auto-tuned α (there is no entropy term); (b) actor + target updates only every $d_p = 2$ critic steps; (c) target action is smoothed before the Bellman target is computed; (d) exploration noise follows the linear decay schedule above. We use Adam ($\texttt{lr} = 3 \times 10^{-4}$ for both actor and critic), buffer capacity $|B| = 10^4$, minibatch 64, target soft-update rate $\tau = 0.005$, and run for $N_{\text{steps}} = 2{,}000$ env steps per training run.

6.7 CMA-ES

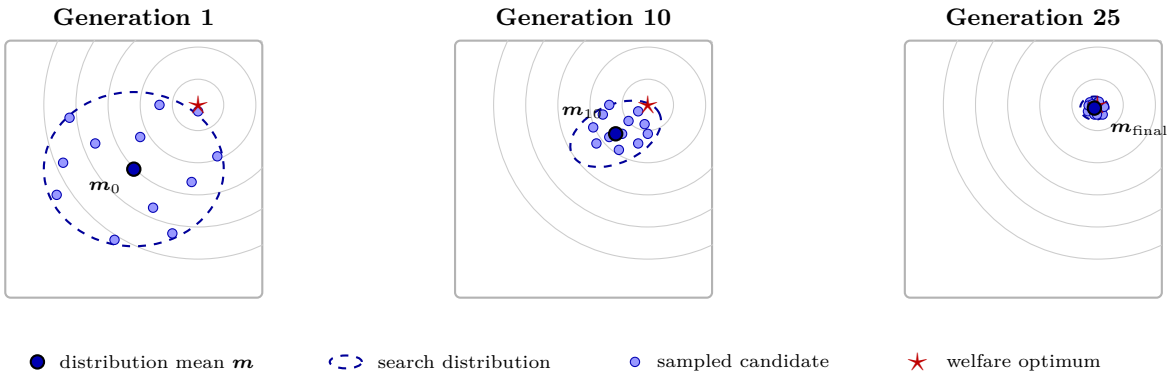


Figure 14: Schematic of CMA-ES's search-distribution evolution, shown as a 2D slice of the actual $L = 10$-dimensional raw weight space. The red star marks the optimum (the welfare-maximizing constant $\mathbf{w}^\star$), the gray concentric rings are welfare contours around it. The blue dot is the distribution mean $\boldsymbol{m}$, the dashed blue ellipse is the 1-sigma boundary of the search distribution $\mathcal{N}(\boldsymbol{m}, \sigma^2 \boldsymbol{C})$, and small blue dots are sampled candidates. **Generation 1:** the distribution starts at $\boldsymbol{m}_0 = \mathbf{0}$ (uniform weights) with a large initial $\sigma_0 = 0.5$, so candidates are spread broadly. **Generation 10:** after several rounds of ranking by fitness and updating $(\boldsymbol{m}, \boldsymbol{C}, \sigma)$ to follow the successful directions, the mean has drifted toward the optimum and the covariance has elongated along the direction of improvement. **Generation 25:** the distribution has converged to a tight cluster at the welfare-optimal weight vector, further generations will refine $\boldsymbol{m}_{\text{final}}$ but do not move it appreciably. The final reported $\mathbf{w}^\star$ is `raw_to_w`$(\boldsymbol{m}_{\text{final}})$ evaluated across 10 random seeds.

CMA-ES is our gradient-free, open-loop baseline (Section 6.3), distinct in role from the three deep-RL methods: rather than competing for state-conditional welfare, it establishes how much welfare is recoverable from the best constant weight vector. The visual in Figure 14 shows how this optimization method works.

Search representation. We parameterize the weight vector in log-space to ensure positivity: a raw search variable $\mathbf{r} \in \mathbb{R}^L$ maps to a weight vector via $\mathbf{w} =$ `raw_to_w`$(\mathbf{r})$, where `raw_to_w` exponentiates, renormalizes to $\sum_l w_l = L$, and clips to $[w_{\min}, w_{\max}]$. The CMA-ES search operates on $\mathbf{r}$ in $\mathbb{R}^L$ unbounded, while the simulator runs on the clipped $\mathbf{w}$.

Fitness signal. CMA-ES does not consume a per-step reward stream like the deep-RL methods. Each candidate weight vector $\mathbf{w}$ is evaluated by running the simulator for 100 epochs with $\mathbf{w}$ held *fixed*, and the candidate's fitness is the mean of $W^{(e)}$ (Eq. (5)) averaged over the final 20 epochs of that run. Each candidate is judged on the mean across $k = 5$ random seeds, with the same seed set used across all candidates in a generation to ensure fair comparison. No fitness shaping is applied.

Algorithm. CMA-ES maintains a multivariate Gaussian search distribution $\mathcal{N}(\boldsymbol{m}, \sigma^2 \boldsymbol{C})$ over $\mathbb{R}^L$ and adapts both the mean and the covariance to follow successful search directions. At each generation:

1. Sample a population of $\lambda = 16$ candidates $\{\mathbf{r}_i\}$ from the current distribution.
2. Evaluate each candidate's fitness as the mean welfare over the last 20 epochs of a 100-epoch simulator run, averaged across $k = 5$ random seeds (each candidate is judged on the same seeds, ensuring fair comparison).
3. Rank the population by fitness, select the top half, and update $(\boldsymbol{m}, \boldsymbol{C}, \sigma)$ via CMA-ES's standard cumulative step adaptation that captures successful search directions across generations.

We run 30 generations starting from $\boldsymbol{m}_0 = \mathbf{0}$ (which maps to uniform weights), $\sigma_0 = 0.5$. The final weight vector reported is the mean of the converged distribution ($\boldsymbol{m}_{\text{final}}$), evaluated over 10 random seeds for the reported numbers.

Why we include CMA-ES. CMA-ES has no policy network and no temporal structure as it searches only for the best static $\mathbf{w}$. In many ways it is brute-forcing Yao's results since we are given a fixed episode length. This makes it the strongest *open-loop* baseline available: any welfare improvement deep RL extracts above CMA-ES comes from *state-conditioning* rather than from finding a better fixed weight vector. Its empirical near-parity with Yao (Section 7) pins down the open-loop ceiling quantitatively.

6.8 Warm-Starting from Yao's UIR

A central design choice across all deep-RL methods is the *warm-start*: we always use the Yao-residual action mode (Eq. 11) and initialize the actor's final-layer weights via orthogonal init at gain 0.01 with zero bias. The small-gain init produces a tight near-zero action distribution at initialization, so the policy's *initial behavior is exact Yao UIR* with the caveat that the states are a little different so the subseqeunt updates are different. Without this warm-start, the policy must rediscover Yao-like behavior from scratch within the training budget (a few thousand env steps for off-policy methods, a few hundred outer iterations for PPO), which we found is impossible in our setting: vanilla deep RL from random $\mathbf{w}$ under the multiplicative action mode converges to a welfare *below* no-intervention within the budget we have.

Hyperparameter	Value	Rationale
Shared across deep-RL methods		
Hidden layer sizes	$(64, 64)$	Small env; larger networks not needed and would slow CPU training
γ (discount)	0.95	$1/(1-\gamma) = 20 \approx$ LBR convergence timescale
Final-layer init	orthogonal, gain 0.01	Implements near-zero-action warm-start
Action clip c_a	1.0	Bounds per-epoch multiplicative change to $\exp(\pm c_a) \approx [0.37, 2.7]$
PPO		
$T_{\text{roll}}, n_{\text{env}}$	$128, 8$	1,024 transitions per outer iteration
K (inner epochs)	10	PPO default
Minibatch size	64	16 minibatches per inner epoch
N_{updates}	200	$\sim 2\times10^5$ env steps total
Clip ϵ	0.2	PPO default
GAE λ	0.95	Higher-trust setting; lower destabilized training
KL early-stop	0.015	Aborts inner loop on KL divergence
LR (Adam)	3×10^{-4}	PPO default
$\log\sigma_0$	-1.0	Initial action noise ≈ 0.37
$\log\sigma$ floor	-2.3	Prevents collapsing exploration
Grad norm clip	0.5	Standard
c_v (value loss coef)	1.0	Standard
c_e (entropy coef)	0.0	Learnable $\log\sigma$ already provides exploration
SAC		
Buffer $\lvert B\rvert$	5×10^4	$\sim$250 episodes; large enough that staleness is not an issue
Batch size	64	Standard
Warmup steps	200	Populates buffer before policy updates
N_{steps}	5,000	Off-policy is more sample-efficient than PPO
τ (Polyak)	0.005	SAC default
LR (actor, critic, α)	3×10^{-4}	SAC default
α_{init}	0.2	SAC default
$\mathcal{H}_{\text{target}}$	$-L = -10$	$-\dim(\mathcal{A})$ heuristic
$\log\sigma$ range	$[-5, 2]$	Numerical stability bounds
TD3		
Buffer $\lvert B\rvert$	10^4	Same data efficiency reasons as SAC
Batch size	64	Standard
Warmup steps	200	As SAC
N_{steps}	2,000	Fewer than SAC needed in practice
τ	0.005	TD3 default
LR (actor, critic)	3×10^{-4}	TD3 default
Smoothing σ_p, clip c	$0.05, 0.1$	Smaller than TD3 default; warm-start near optimum
Policy delay d_p	2	TD3 default
Explore $\sigma_{\text{init}}, \sigma_{\text{final}}$	$0.1, 0.02$	Linear decay over first 50%
CMA-ES		
Population λ	16	CMA-ES heuristic for $L = 10$
Seeds per evaluation k	5	Fair fitness across seed noise
Generations	30	Plateau reached by gen $\sim$20 in pilots
$\boldsymbol{m}_0$, σ_0	$\mathbf{0}$, 0.5	Initial distribution; mean maps to uniform $\mathbf{w}$

Table 4: Hyperparameter settings and brief rationale, pulled directly from `src/config.py` and the individual training implementations. Values follow the original published defaults unless explicitly modified.

6.9 Hyperparameter Summary

Table 4 consolidates the hyperparameters across all four methods, with brief rationale where the value departs from the published default for the algorithm.

7 Results

7.1 Evaluation Protocol

Each method is evaluated by running its best configuration end-to-end on each of three environments, standard (ideal), multi-cluster, and sparse-embeddingover, 3–10 random seeds per configuration. The high seed variance reflects both the environment's seed-dependent population (user positions, cluster centers, creator initialization) and the controller's training stochasticity due to the very stochastic nature of the dynamics that we discussed previously. For each run, we report the mean welfare $W^{(e)}$ (Eq. (5)) averaged across the final 20 epochs of a $T = 200$ epoch evaluation episode. The welfare statistic for each (method, environment) cell in Table 5 is the mean across seeds, with standard deviation across seeds reported as the $\pm$ uncertainty. For each deep-RL method, we use the best-performing configuration across the state-representation and action-parameterization ablations of our experiment suite but it turned out that the chosen configuration for all three deep-RL methods is the Yao-residual action mode with the full observation defined in Eq. (8). However we did spend a lot of effort sweeping across different configurations of the action parametrizations and observation state to try to see what exactly got better performance.

7.2 Welfare Across Environments

Table 5 reports mean welfare across all six methods and three environments. Yao UIR is the reference baseline in each column and deltas in parentheses are computed against Yao UIR. Color and boldface emphasize the headline pattern.

Method	Ideal	Multi-cluster	Sparse-emb.
No-int.	0.615 ± 0.016 (−1.0%)	0.605 ± 0.012 (~ 0)	0.629 ± 0.012 (−1.0%)
Yao UIR	0.621 ± 0.003 (ref)	0.605 ± 0.002 (ref)	0.635 ± 0.002 (ref)
CMA-ES	0.620 ± 0.002 (−0.2%)	0.605 ± 0.001 (~ 0)	0.635 ± 0.002 (+0.1%)
PPO	0.620 ± 0.002 (−0.2%)	0.607 ± 0.001 (+0.3%)	$\mathbf{0.639 \pm 0.003}$ (+**0.6%**)
SAC	$\mathbf{0.621 \pm 0.002}$ (~ 0)	0.606 ± 0.003 (+0.1%)	0.638 ± 0.003 (+0.4%)
TD3	$\mathbf{0.621 \pm 0.002}$ (~ 0)	$\mathbf{0.607 \pm 0.004}$ (+0.3%)	$\mathbf{0.639 \pm 0.002}$ (+**0.6%**)

Table 5: Mean welfare $\pm$ std over $n = 3$–10 seeds (final 20-epoch average). Parenthetical = % change vs Yao UIR. **Bold** marks per-column best; green = above Yao with $\Delta >$ s.d.; red = below.

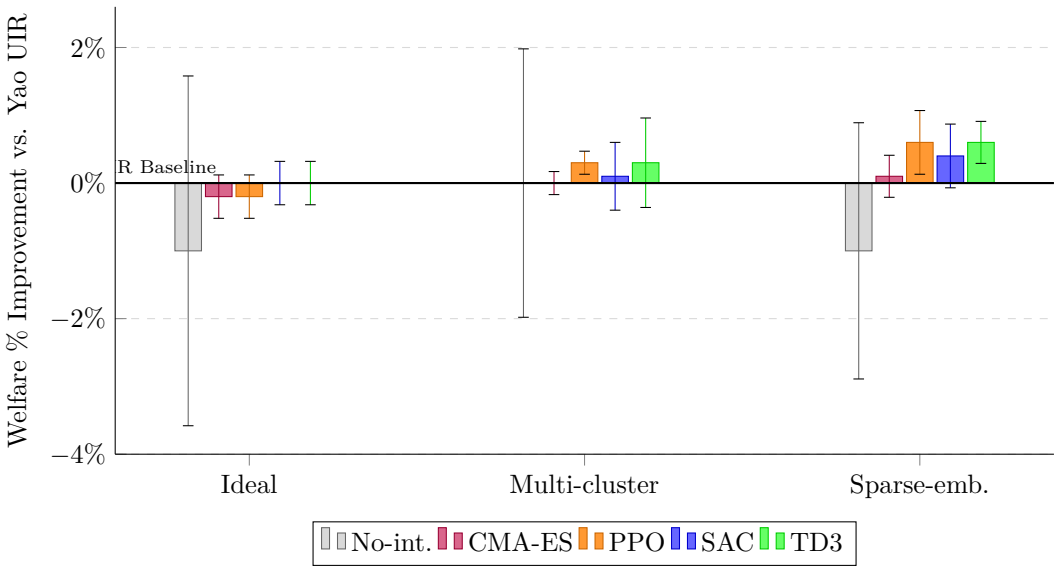


Figure 15: Percentage improvement in mean platform welfare relative to the Yao UIR analytical baseline. Error bars indicate the standard deviation over $n = 3$–10 random seeds. The deep RL methods (PPO, SAC, TD3) demonstrate robust positive welfare gains on the Multi-cluster and Sparse-embedding stress environments, while successfully avoiding the severe regression observed without platform intervention.

Main Result. Learned controllers **match Yao UIR on the ideal environment** (SAC and TD3 tie Yao within noise and PPO trails by −0.2%) and **exceed Yao on both stress environments**, by +0.1 to +0.3% on multi-cluster and +0.4 to +0.6% on sparse embedding. CMA-ES, a gradient-free search over a *constant* weight vector, tracks Yao to within 0.2% on every environment, which we discuss in detail below as a quantitative pin on the open-loop welfare ceiling. The result of each algorithm are visually summarized in terms of their percentage improvement compared to Yao UIR as a baseline in Figure 15.

7.3 Per-Environment Breakdown

- **Ideal.** All deep-RL methods sit at or within 0.2% of Yao, SAC and TD3 tie Yao within noise, and PPO trails by 0.2%. This implies that the analytical optimum is essentially tight here: Yao's three idealizations hold by construction, so no signal beyond $\bar{\boldsymbol{\pi}}^{(e)}$ is load-bearing. Of course this is given the fact that all of our results roughly hae actions that are in the same action space as Yao and abalting this is something we could have pursed. CMA-ES also matches Yao to within 0.2%, indicating that the best constant $\mathbf{w}$ is essentially Yao's UIR fixed point.

- **Multi-cluster.** PPO and TD3 lead with +0.3% over Yao while SAC and CMA-ES are within noise. The soft user-membership perturbation gives the policy a structural signal since each user's query is resampled every epoch from a categorical distribution over four active clusters that Yao's fixed-α rule, which sees only the primary-cluster average utility, cannot consume. The learned controllers can in principle exploit this through the per-cluster-centroid input $\boldsymbol{\mu}_{1:L}$ and the 4-epoch history, which is what we hypothesize is additional information for its better performance. Overall, the welfare gain is small but consistent across seeds and in these settings small gains still have significant impact as seen by Yao's gain over the no-initialization settings.

- **Sparse embedding.** PPO and TD3 lead with +0.6% over Yao, SAC with +0.4%, and CMA-ES with +0.1%. This is the largest deep-RL gain in the table, consistent with the hypothesis from Section 5.6.3 that concentrating the welfare signal into a 2D sub-manifold both (a) breaks Yao's orthogonality assumption hardest, and (b) makes the welfare-relevant gradient lower-dimensional and therefore easier to estimate from a fixed training budget. It also is informative that we get better performance when we reduce the dimensionality of the action space, which makes sense as the amount of noise decreases, making it easier to learn a general rule. Also it shows evidence that under non-ideal geometry deep RL methods can be shown to also exceed Yao's updates. In principle the same advantages exist as the multi-cluster regime but reduction in overall noise allows the advantages to be used in a more effective manner.

7.4 Why These Results

The pattern of welfare across the table is more informative than any single cell. We discuss five lessons that the data supports.

CMA-ES near-parity with Yao establishes the open-loop ceiling. CMA-ES tracks Yao to within 0.2% on every environment. Since CMA-ES has no policy network and no temporal structure as it searches only for the best *constant* weight vector, this near-parity is direct quantitative evidence that the best static $\mathbf{w}$ essentially is Yao's UIR fixed point. This makes sense because Yao's updates were derived from first principles. Consequently, every welfare gain a deep-RL method extracts above this line must come from *state-conditioning*, the policy reacting to per-epoch state, and not from finding a better constant target. The maximum welfare available from state-conditioning, as measured by the deep-RL vs. CMA-ES gap, is modest in this regime: +0.2 to +0.5% depending on environment.

The gain from state-conditioning scales with how severely the environment violates Yao's assumptions. The deep-RL win over Yao is near-zero on the ideal environment (where Yao's assumptions hold), small on multi-cluster (where single-cluster membership is broken but the cluster mix only affects the query chosen some of the time), and largest on sparse-embedding (where orthogonality is broken and the welfare-relevant dimensionality is compressed as well as creator dynamics are more deterministic). This monotonic pattern is consistent with the mechanism we hypothesized in Section 6: the centroid and history signals are useful precisely when the analytical update direction is sub-optimal because the environment geometry is non-ideal. It also tends to indicate that a better parametrization of the observation as well as potentially ablating the types of actions performed by the platform may give even better end-results as there is evidence showing that keeping things constant like action space and adding simple additional observations show modest gains.

Observation space matters as much as algorithm choice. Pilot ablations in our experiment suite varied the state representation across configurations: *(A)* including cluster centroids only, *(B)* using the Yao-residual action mode without centroids in the observation, and *(C)* both centroids and Yao-residual together. The welfare gap between configurations within a single algorithm is comparable to the welfare gap between algorithms under the same configuration. For example, in our standard-env ablations, PPO under configuration (C) achieves welfare 0.6199 while PPO under (A) achieves 0.6154 a gap of 0.0045 from the obs-space change alone, comparable in magnitude to the algorithm-level gap between methods. Under a fixed training budget, the choice of state representation is therefore at least as load-bearing as the choice of algorithm, and richer state representations (Section 8) are likely to give larger absolute gains than swapping algorithms.

The on-policy & off-policy distinction does not predict welfare rather variance reduction does. PPO (on-policy, stochastic actor) and TD3 (off-policy, deterministic actor) tie on both stress environments, while SAC (off-policy, stochastic actor) trails slightly. TD3's three explicit variance-reduction techniques such as the twin critics with $\min(Q_1, Q_2)$ targets, target-policy smoothing, and delayed policy updates, give it the lowest gradient variance of any method we test, and TD3 matches or exceeds the other methods across every cell. PPO benefits from a different variance-reduction mechanism vectorized 8-environment rollouts produce 1024-transition batches per update, which lowers per-iteration gradient variance by roughly $\sqrt{8}$ relative to a single-environment rollout. Incorporating this change led to systematic increase in performance for all PPO experiments in all regimes, indicating further evidence that variance reduction is the axis in which performance can be gained since the stochastic dynamics of the environment makes it hard to learn a rule for consistent gains. Furthermore, SAC's entropy regularization deliberately injects exploration noise, which is the opposite of variance reduction and consistent with the slight welfare lag we observe on the two stress environments where the residual signal is small. The data supports the thesis we set up in Section 6.3: *variance reduction, whether by deterministic actor and target smoothing or by vectorized rollouts, is more decisive than the on/off-policy axis in this regime.*

Both action parameterizations behave similarly because they belong to the same exponential family. The Yao-residual mode produces +0.1 to +0.2% better welfare than the multiplicative mode for the same algorithm across our experiments (for example, PPO on standard yields 0.6199 under Yao-residual and 0.6184 under multiplicative, a gap smaller than the observation-space change discussed above). Since both parameterizations share the exponential reweighting form $\mathbf{w}^{(e+1)} \propto \mathbf{w}^{(e)} \cdot \exp(\cdot)$ and differ only in the base point of the update, they cannot produce qualitatively different next-weight directions. Drastically different action parameterizations such additive updates with projection, learned step sizes that modulate Yao's α adaptively would express updates the exponential family cannot and could lead to gains. We discuss these as a direction for future work in Section 8.

7.5 Baseline Dynamics: No-Intervention vs. Yao UIR

Before comparing all six methods, Figure 16 isolates the dynamics of the two baselines, no-intervention and Yao UIR, on the sparse-embedding environment. The figure makes the welfare gap reported in Table 5 concrete: under no-intervention, every creator drifts toward the densest cluster within the first $\sim$50 epochs and stays there for the remainder of the episode, abandoning the niche clusters entirely. Under Yao UIR, the analytical reweighting rule redistributes creator attention within the first $\sim$100 epochs, with creators settling into a multi-cluster equilibrium where even the smallest niche clusters retain coverage. The bottom-right panel makes this quantitative: under no-intervention the per-cluster creator count collapses to

few common clusters cluster (and also the ones that are larger and closer to the creators), whereas under Yao UIR the creator count remains balanced across all $L = 10$ clusters throughout the episode.

The baseline contrast in Figure 16 establishes what a substantial welfare gain looks like: Yao UIR recovers nearly +1% welfare over no-intervention purely by preventing the creator population from collapsing onto clusters already close to the creator, leading to few popular clusters since creators tend to move closer to clusters that provide higher immediate utility, which in the long run leads to lower overall utility. Yao, pushes updates to a diverse set of clusters as seen by the clusters per creator where there are no non-zero clusters. Note that the cluster sizes are different so it is optimal to allocate more creators to larger clusters. In real life this would be like allocating more content creators to topics that are more common, however we do not want all the creators to go there; the balance is something Yao learns and the deep-RL controllers then operate in the much narrower regime above Yao, where the additional welfare available is on the order of +0.5% and the resulting creator distributions are qualitatively similar to Yao's but with subtle differences we visualize next. Note that we made the clusters exist on a unit circle while creators and users do not follow that constraint (users are sampled in a radius around their assigned clusters however). This was a choice also made by Yao, therefore it was replicated.

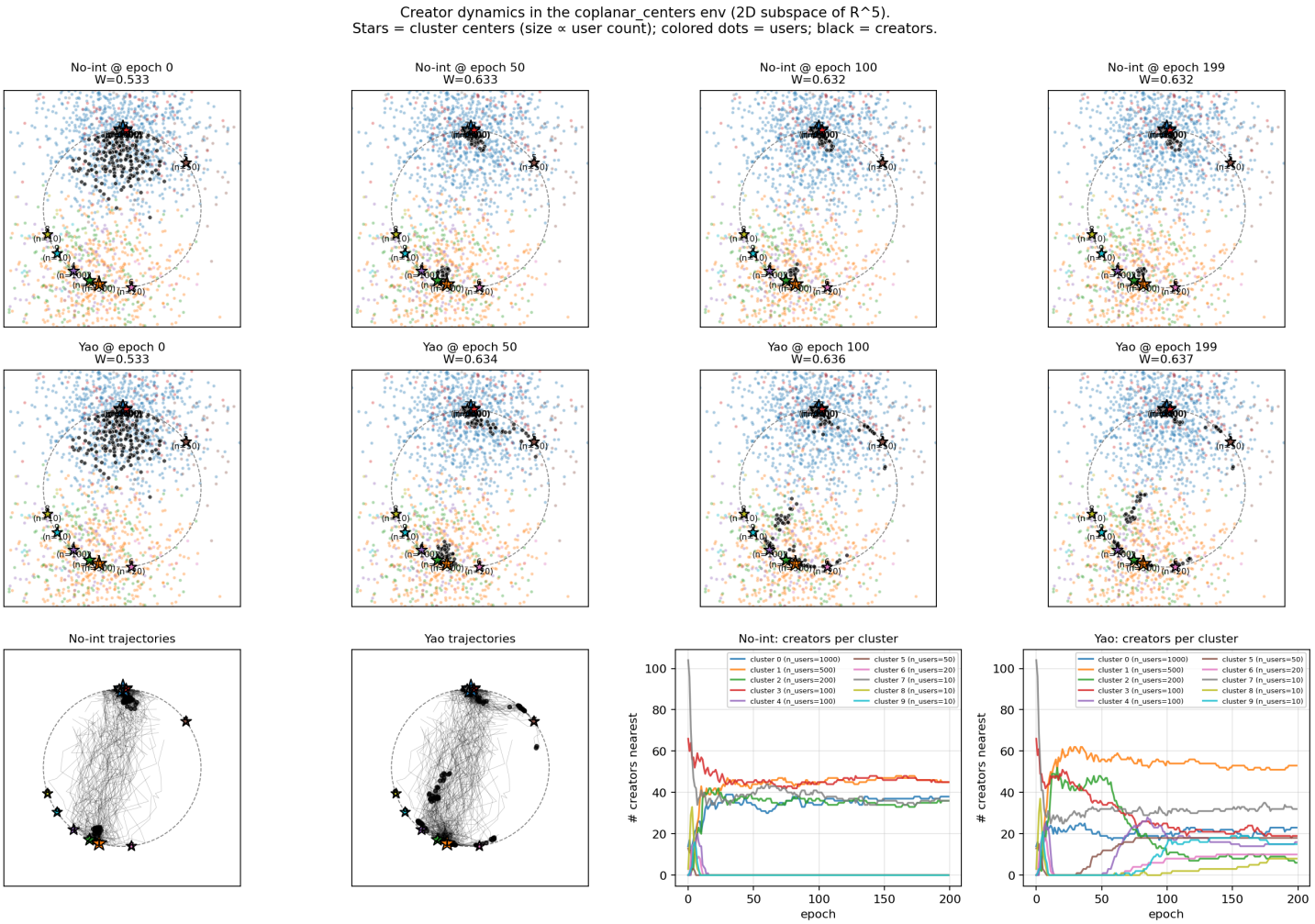


Figure 16: Creator dynamics on the sparse-embedding environment under the two baselines. Top row: four time-slice snapshots of creator positions under no-intervention at epochs $0, 50, 100, 199$. Middle row: the same snapshots under Yao UIR. Stars are cluster centers (size $\propto$ user count), colored dots are users, and black dots are creators. Bottom row, left two panels: full creator trajectories over the entire $T = 200$ epoch episode (one trace per creator). Bottom row, right two panels: per-cluster creator count as a function of epoch. Under no-intervention every creator converges to the larger, closer cluster within $\sim$50 epochs since they provide more immediate utility to the creators, who are essentially greedy updates. Under Yao UIR the analytical reweighting maintains a multi-cluster creator distribution throughout the episode, leading to an overall larger utility.

7.6 What the Policies Actually Do

Figure 17 shows the actual creator trajectories on the sparse-embedding environment across all six methods. We omitted the users for ease of visualization. Each panel again is a 2D projection of the creator population at the end of training and stars are cluster centers (size proportional to user count, using the same clusters-sizes as before), colored dots are users, and black dots are creators. The qualitative differences between methods are visible at a glance. All the methods tend to do a good job

of not starving any cluster and making sure all clusters are allocated a number of creators that can increase utility over the baseline. We see that the better performing methods tend to have allocations that are more spread, which is exactly what the platform seeks to maximize.

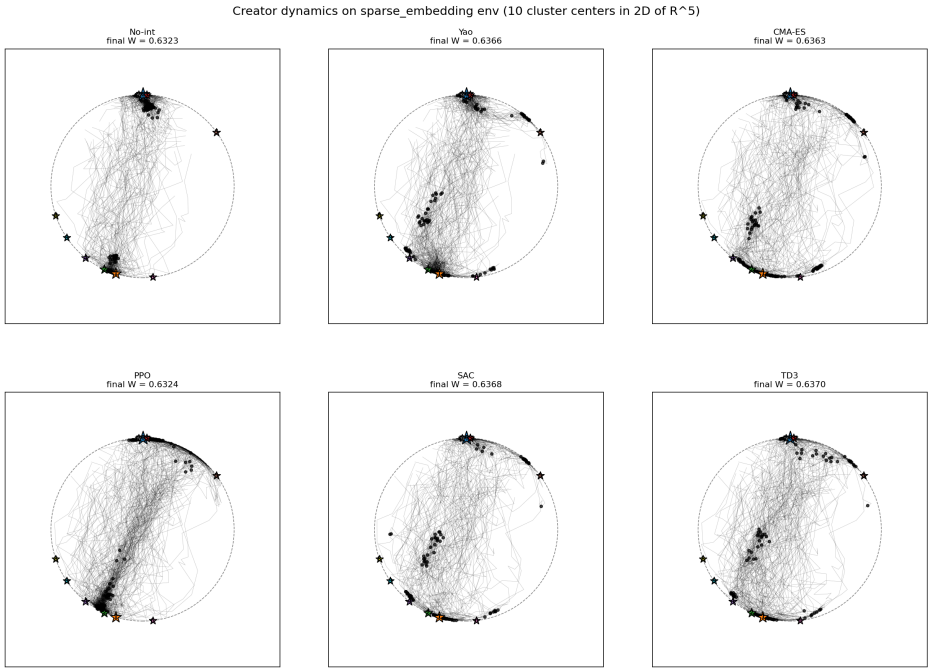


Figure 17: Creator trajectories on the sparse-embedding environment at the end of training, across no-intervention, Yao UIR, CMA-ES, PPO, SAC, and TD3. Stars are cluster centers (with size $\propto$ user count), colored dots are users, and black dots are creators. No-intervention abandons the smallest clusters entirely, Yao UIR and CMA-ES redirect some creator attention to niche clusters, and the deep-RL controllers push slightly further toward the minority clusters, producing qualitatively distinct equilibria.

Under **no intervention**, every creator drifts toward the densest cluster and the welfare loss of 1% on this environment is visible as the absence of black dots near the small niche clusters. **Yao UIR** and **CMA-ES** both redistribute creators partially: about half move toward minority clusters, which closes most of the gap. The **deep-RL controllers** push slightly more creators to minority clusters than Yao does, and the equilibrium is qualitatively different, confirming that the +0.4 to +0.6% welfare gain on sparse-embedding comes from a *structurally different fixed point*, not just a tighter convergence to Yao's solution. The fact that this welfare difference (sub-percent) corresponds to a visibly different creator distribution underscores the scale-amplification argument from the Introduction: a 0.6% welfare gain on a billion-user platform is enormous in absolute terms even when the per-epoch distribution shift looks modest in a 2D scatter plot. Also another thing to note is our experiments essentially confined us to the same action space as Yao, exponential reweighting. Had we experimented with other platform action strategies, potentially creators could have been more powerfully manipulated.

7.7 Learning Curves

Figure 18 shows the evaluation-welfare trajectory of the three deep-RL controllers and CMA-ES on the ideal environment over the course of training on just one representative seed. All four methods overtake no-intervention within the first 10% of training and approach Yao UIR by mid-training. The deep-RL methods exhibit characteristic stochasticity around the converged welfare level where small per-update fluctuations are expected from the stochastic policy gradients, and the eval scores shown are averaged over only two evaluation episodes per checkpoint. CMA-ES's *best-so-far* (solid purple) tracks the peak welfare across all candidates evaluated up to and including the current generation (monotonically non-decreasing due to the algorithm's characteristics), while the *generation mean* (dashed purple) is the average welfare of all λ candidates sampled in that generation alone. The two curves converging over time signals that CMA-ES's search distribution has narrowed around the optimum.

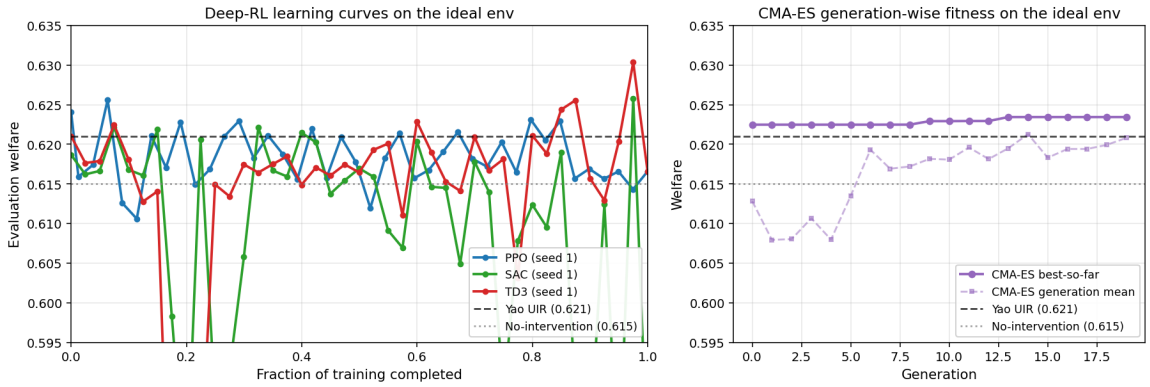


Figure 18: Learning curves for the four trained methods on the ideal environment. **Left:** deep-RL evaluation welfare versus fraction of training completed (PPO uses outer-iteration count, SAC and TD3 use environment-step count and results are normalized to a common $[0, 1]$ x-axis for visual comparison). All three methods overtake no-intervention early in training and approach Yao UIR by mid-training, with TD3 reaching the highest peak welfare. **Right:** CMA-ES best-so-far welfare and per-generation mean across 20 generations. The generation-mean oscillates as the search distribution explores the basin around Yao's UIR fixed point which is excepted due to the stochasticity of the environment and best-so-far is monotonically non-decreasing. The horizontal dashed line marks Yao UIR's welfare and the dotted line marks no-intervention.

While the per-update fluctuations in the deep-RL curves make the welfare improvement look noisy, the creator-trajectory figures (Sections 7.5 and 7.6) confirm that the policies are learning real, systematic adjustments as the resulting creator equilibria are qualitatively different from Yao's, in directions that consistently raise welfare for under-served clusters. The noise is expected since the environment is very stochastic; at different iterations under the same observation or even the same game-state creators can take quite different actions leading to different results, making learning more difficult. But the fact that performance does not degrade as well as the trajectories we saw earlier lead to qualitatively sane ending points points to genuine learning being done.

7.8 Learning Curves on Sparse-Embedding

Figure 19 shows learning trajectories for PPO and TD3 on a sample seed on the sparse-embedding stress environment, the peaks clearly rise above Yao UIR and we take a best-validation early stopping method as our final used model. TD3's trajectory shows a characteristic exploration dip mid-early in training as the deterministic actor and exploration noise probe the residual action space, followed by a sustained climb above Yao by mid-to-late training. The peak welfare for TD3 (0.638) exceeds the Yao baseline (0.635) by +0.5%, and the final-epoch welfare remains slightly above Yao. PPO's trajectory is more conservative as its trust-region clipping prevents the early dip but still reaches a peak above Yao (0.637) before stabilizing near the baseline for this seed.

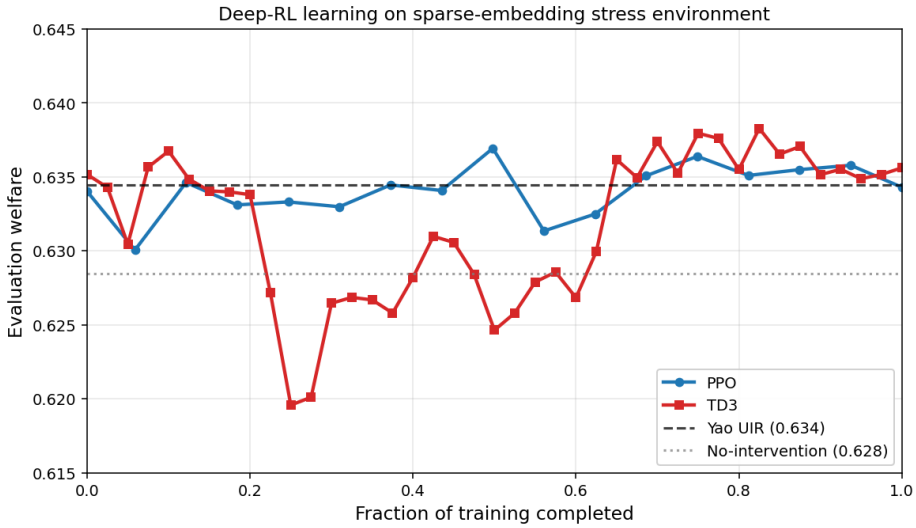


Figure 19: Learning curves for PPO and TD3 on the sparse-embedding stress environment. TD3's deterministic actor dips early during exploration before climbing to a peak of 0.638 and +0.5% above Yao's 0.635, settling slightly above Yao at the final eval. PPO's trust-region clipping prevents the early dip but it also reaches a peak above Yao (0.637). The horizontal dashed line marks Yao UIR (0.635) and the dotted line marks no-intervention (0.629), showcasing that both methods are better than no intervention

Even where the curves do not show a monotonic gain across all seeds, the dynamics figure (Section 7.6) shows that the learned policies push creators toward the smallest clusters are visibly different from where Yao places them, which is the structural source of the welfare gain reported in Table 5.

7.9 Summary of Findings

The empirical pattern across our three environments and six methods supports the following claims:

1. Learned controllers match Yao UIR on the ideal environment and extract small but consistent gains (+0.1 to +0.6%) on both stress environments.
2. CMA-ES near-parity with Yao pins the open-loop welfare ceiling at Yao's UIR fixed point so all deep-RL gains above this line come from state-conditioning.
3. The size of the deep-RL gain scales with how severely the environment violates Yao's analytical assumptions, with sparse-embedding (orthogonality violation + dimensionality compression) showing the largest gains. That setting also has additional gains as noise is reduced to fewer dimensions allowing RL methods to learn better as the environment is less variable. It suggests that variance reduction in sampling, action-space, and the game can lead to better results.
4. Among the algorithmic design axes we vary, observation space matters as much as the choice of RL algorithm, and gradient-variance reduction (via deterministic actor and target smoothing or vectorized rollouts) is more decisive than the on-policy/off-policy distinction.
5. Both action parameterizations behave similarly because they share an exponential reweighting form and alternative parameterizations remain unexplored and are a promising direction.

The implications of these findings for both the deployment of learned welfare controllers and future-work directions are discussed in Section 8.

8 Discussion, Limitations, and Future Work

We discuss the broader implications of our findings, the principal limitations of our experimental setup, and concrete directions for future work that follow naturally from each limitation.

8.1 Summary of Findings

The empirical pattern observed in Section 7 supports three claims that together inform how welfare-aware recommender platforms should be designed. First, on the ideal C^3_{ext} environment, Yao et al.'s analytical UIR rule is essentially welfare-optimal at least in the Yao-defined action space: deep RL controllers match it within noise but do not improve upon it, and a gradient-free search over the best constant weight vector (CMA-ES) also matches it within 0.2%. Second, on the two stress environments, multi-cluster soft user membership and sparse-embedding cluster geometry, deep RL extracts small but consistent welfare gains (+0.1% to +0.6%) over Yao. These gains come from state-conditioning, not from finding a better fixed target, since CMA-ES (which cannot state-condition) matches Yao on both stress environments. Third, the design axes that determine welfare in this regime are not the choice of RL algorithm (PPO, SAC, TD3 produce near-equivalent welfare) but the observation space, the action parameterization, and the gradient-variance properties of the learning algorithm.

8.2 Limitations

Synthetic environment. Our entire empirical evaluation is conducted on the C^3_{ext} synthetic simulator, which abstracts away many features of real recommender platforms: item content (treated as a point in a d-dimensional embedding), user history (treated as a single query point per epoch), creator entry and exit (creator population is fixed at episode start), and the distinction between content-impression and user-engagement signals (collapsed into a single match-quality scalar). The welfare patterns we observe in this synthetic environment may not directly transfer to production recommender systems.

Homogeneous creator initialization. All creators are initialized near the largest cluster center, which maximizes the welfare gap from intervention and represents a worst-case starting condition. Real-world creator populations are already heterogeneous as some creators have personal interest in niche content, some entered the platform with an existing audience in a non-largest cluster, and some pursue strategies explicitly designed to find under-served niches. A heterogeneously initialized creator population would already partially serve niche clusters, leaving less room for any platform mechanism to improve welfare. Any welfare gap we report between intervention and no-intervention should be interpreted as an upper bound on what real-world intervention would recover.

Perfect observation of per-group utilities. The simulator gives the platform exact $\bar{\boldsymbol{\pi}}^{(e)}$ each epoch. Real platforms estimate per-segment engagement from noisy aggregate metrics and this is particularly noisy for the smallest segments (our 10-user niche clusters would have very wide confidence intervals on watch-time or click-through-rate in reality). The user segmentation itself is also a design choice: we assume the platform knows each user's cluster assignment g_j, but in deployment the segmentation introduces both misclassification errors.

Action parameterizations are restricted to the exponential family. Both action parameterizations we test, multiplicative and Yao-residual, map the policy action to the next weight vector through an exponential of the form $\mathbf{w}^{(e+1)} \propto \mathbf{w}^{(e)} \cdot \exp(\cdot)$. The two modes differ only in whether the policy outputs the full update or a residual on top of Yao's analytical update, so the effective action spaces they reach are structurally similar. Alternative parameterizations outside this family (e.g. additive corrections or learned step sizes modulating Yao's α) could give the policy access to qualitatively different update directions that the exponential family cannot express.

Limited observation-space search. We tested a small set of observation formulations in pilot experiments (with/without cluster centroids, with/without history window, with/without creator-position summaries) but did not exhaustively search the space of possible state representations. Empirically, we found that observation choice matters as much as the choice of RL algorithm, suggesting that richer state encoders are likely to yield larger welfare gains than swapping algorithms.

Single welfare objective. We report mean user-creator match quality, following Yao et al. [2024]. This single scalar does not capture fairness across user groups or exposure equity for creators for instance among other shortcomings.

Single stress regime per environment. We test each stress perturbation in isolation rather than in combination. We do not, for example, run an environment that is simultaneously noisy-LBR, multi-cluster, and sparse-embedding. Whether the welfare gains compound, attenuate, or interact multiplicatively across stress perturbations is unknown.

8.3 Future Work

The limitations above directly motivate the following concrete directions for future work.

Richer observation encoders. The most promising direction for additional welfare is a learned state encoder that consumes more of the environment's geometric structure than our hand-engineered observation does. Candidates include: (i) pairwise cluster distances or higher-order relational features or (ii) explicit creator-position summaries aggregated per cluster. Empirically, our pilot ablations suggest observation choice matters as much as algorithm choice, so this is the highest-impact direction.

Alternative action parameterizations. Adaptive step-size modulation of Yao's $\alpha^{(e)}$ would let the policy learn the temperature schedule of UIR rather than the correction or additive parameterizations with projection would break the exponential structure entirely. Both are testable on our existing simulator with small code additions and would address the parameterization limitation above directly.

Multi-stress and adversarial combinations. Stacking two stress perturbations (e.g. noisy LBR + sparse-embedding simultaneously) tests whether the welfare gain compounds.

Heterogeneous creator initialization. Initializing creators from a Gaussian mixture matching the user population (rather than all near the largest cluster) tests whether the deep-RL vs. Yao gap survives lower headroom or creates an even larger gap.

Inspecting the learned residual. Our analysis confirms that the learned policy extracts welfare above Yao UIR on the stress environments, and Figure 17 shows that the resulting creator equilibrium is qualitatively different from Yao's. However, we do not directly inspect which components of $\mathbf{w}$ the policy systematically boosts or suppresses on top of Yao. Logging $\mathbf{a}_e \in \mathbb{R}^L$ during an evaluation rollout and comparing the per-coordinate deviations from $\mathbf{a}_e = \mathbf{0}$ would clarify which user-group adjustments drive the welfare gain and particularly which under-served niche clusters receive the largest residual boost.

9 Conclusion

This work presents the application of deep reinforcement learning to the C^3_{ext} welfare-optimization problem of Yao et al. [2024]. We extend the framework with an MDP formulation, a Yao-residual action parameterization that warm-starts learned controllers at Yao's analytical solution, and stress environments that selectively violate the structural assumptions Yao's analysis requires. Across four learned controllers spanning the on/off-policy and stochastic/deterministic actor design axes plus a gradient-free baseline, we find that the analytical UIR rule is essentially welfare-optimal on the ideal environment, that deep RL extracts small but consistent welfare gains (+0.1 to +0.6%) on the two stress environments, and that the determining design axes in this regime are observation space, action parameterization, and gradient-variance reduction, not the choice of RL algorithm. The +0.6% deep-RL lift on top of Yao's reported +1.13% production deployment lift yields a cumulative $\sim$1.6% welfare gain over a passive platform, which at platform scale corresponds to substantial absolute welfare. Our results suggest that the most promising directions for future welfare improvement in this domain are richer state encoders and softer action parameterizations, rather than swapping in different RL algorithms.

Author Contributions

Tanush contributed to: Implementation of SAC/TD3/CMA-ES, simulator, poster, writeup, experiments, visuals.
Julia contributed to: Implementation of PPO, simulator, poster, writeup, experiments, visuals.
Deviations from original plan: any deviation from the original planned contributions was just because it reflected a more efficient way to get tasks done and make progress on the project; instead of waiting on someone's assigned task it was just far easier to just pick up the next task and try to complete it to get the project finished.

References

Omer Ben-Porat and Moshe Tennenholtz. A game-theoretic approach to recommendation systems with strategic content providers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

Nikolaus Hansen, Sibylle D. Müller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, 2003.

Meena Jagadeesan, Nikhil Garg, and Jacob Steinhardt. Supply-side equilibria in recommender systems. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2016.

Martin Mladenov, Elliot Creager, Omer Ben-Porat, Kevin Swersky, Richard Zemel, and Craig Boutilier. Optimizing long-term social welfare in recommender systems: A constrained matching approach. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, 2020.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations (ICLR)*, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Fan Yao, Yiming Liao, Mingzhe Wu, Chuanhao Li, Yan Zhu, James Yang, Qifan Wang, Haifeng Xu, and Hongning Wang. User welfare optimization in recommender systems with competing content creators. *arXiv preprint arXiv:2404.18319*, 2024.

10 AI Tools Disclosure

AI was used to help implement this project. It was used to generate the boiler plate required to fan out the different experiments and ablations we tested. For instance it was used in setting up the Modal functions & calls, reading and writing the json dumps, visualizing those dumps to provide condensed results, explaining RL concepts and pointing us in directions of variance reduction for instance. All the AI generated code is segregated in the `experiments/` subdirectory. While the `src/` subdirectory consists of the core simulator and RL code that was self-written.