MIXTURE OF GFLOWNETS

Anonymous authors
Paper under double-blind review

ABSTRACT

Generative Flow Networks (GFlowNets) learn a policy that samples compositional objects in proportion to an unnormalized reward $R(x)$. This makes them useful wherever many diverse, high-reward candidates are needed and oracle queries are expensive, as in AI for Science. In practice, a single GFlowNet often fails to cover all modes of a complex distribution, and the obvious ways to scale compute saturate: larger batches and data-parallel gradient averaging draw every trajectory from one, possibly biased, policy, so extra rollouts rediscover known modes instead of finding new ones. We explore whether this mode-bias problem (mode collapse in the extreme) can be addressed with a population of asynchronously trained policies which divide the state space between them. **Mixture of GFlowNets (MoG)** replaces the single forward policy with a *community* of K independent GFlowNets coupled through one jointly trained soft classifier, which assigns each member its own region of the state space through reward shaping. Mixing the members under their learned partition functions provably recovers the target $p(x) \propto R(x)$. **MoG exchange (MoGX)** also lets the classifier route sampled trajectories: a trajectory the classifier assigns to another member is sent to that member and trained on there, with no importance sampling. We evaluate both methods by keeping the environment-interaction budget fixed while scaling the community size, turning parallel *compute* into mode *coverage* without extra oracle calls. On HyperGrid, across six reward landscapes and communities of up to $K{=}64$, MoG and MoGX keep finding new modes as K grows, while data-parallel training saturates. On a genomic ideotype-design task, MoGX covers 39% of the 1024 optima at $K{=}64$, against 31% for independent agents and 20% for data-parallel training. MoG also has a lower per-iteration synchronization cost than data-parallel training.

1 INTRODUCTION

Many *AI for Science* problems, such as drug design, materials discovery and biological sequence engineering, require searching huge combinatorial spaces for many distinct, high-reward objects. Here *sample diversity* is the goal, and the dominant cost is the environment query rather than gradient compute, so the number of sampled trajectories should be minimized. Generative Flow Networks (GFlowNets) fit this regime (Bengio et al., 2021; Jain et al., 2022; Bengio et al., 2023; Pan et al., 2024). They amortize sampling from an unnormalized reward $R(x)$, so the trained policy samples $x \propto R(x)$ and spreads over the high-reward regions instead of collapsing onto one, as *return-maximizing* RL tends to (Malkin et al., 2022; Madan et al., 2023).[1]

In practice, trained GFlowNets often miss modes and concentrate on a subset of the high-reward regions; Dall'Antonia et al. (2025) report this on synthetic benchmarks and on peptide design. Two lines of work try to fix this. The first adds exploration to a single policy: intermediate rewards, intrinsic bonuses, Thompson sampling, local search, replay prioritization, or exploratory "sibling" networks (Pan et al., 2023; Kim et al., 2023; Rector-Brooks et al., 2023; Shen et al., 2023; Schaul et al., 2016; Vemgal et al., 2023; Madan et al., 2025). These raise trajectory diversity, but all exploration happens around one learned sampler, and a biased sampler limits what can be reached at any iteration. The second uses several GFlowNets. Dall'Antonia et al. (2025) fit a chain of GFlowNets, each targeting the reward its predecessors left uncovered, which helps coverage but is serial, and therefore cannot leverage parallel compute. In contrast, Ikram et al. (2025) (EGFN) evolve a paral-

[1] Entropy-regularized RL and samplers for unnormalized densities are closely related; see section A.

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

lel population of GFlowNet agents by weight mutation and distill the population's best trajectories into one gradient-trained sampler through a prioritized replay buffer. The population's search is uncoordinated, and so it serves as an exploration front for that one sampler.

Adding parallel rollouts during training of a single policy gives *saturating* returns. Obando Ceron et al. (2023) found that *smaller* batches improve value-based deep RL, and Singla et al. (2024) show that on-policy policy-gradient training plateaus at tens of thousands of parallel environments, because extra samples from one policy add diminishing trajecotory diversity. SAPG overcomes this plateau by dividing the environments among several sub-policies, but the split helps only when each sub-policy also learns, off-policy, from the data the others collect. What scales is data that a single policy would not produce. GFlowNets saturate for the same reason: Every trajectory in a batch comes from the same forward policy $P_F^{\pi_\theta^i}$, so extra trajectories land in modes the policy already knows and carry no additional signal about missing modes. If the policy's *reachable set* $\mathcal{R}^{\pi_\theta^i}(s_0)$ at iteration i is a strict subset of the terminal space $\mathcal{X}$, no batch size closes the gap. This is known as the *mode-bias* problem (which leads to *mode collapse* in the extreme).

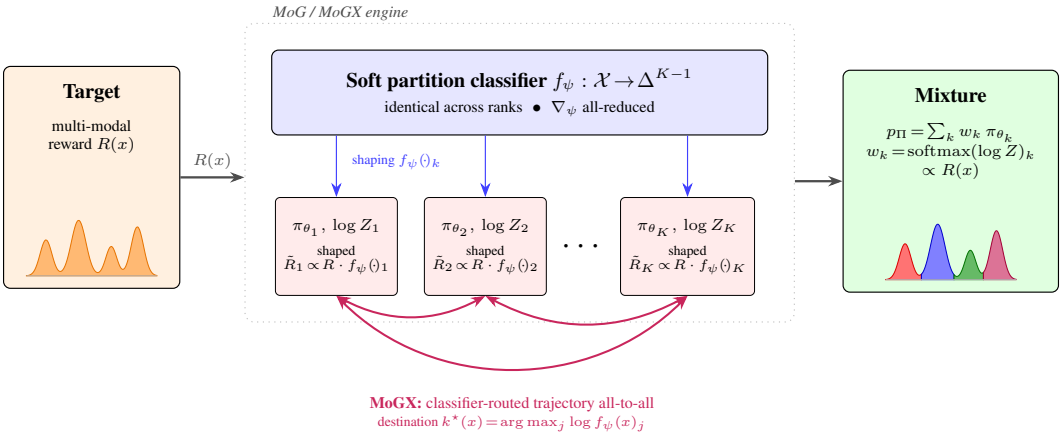


Figure 1: **Mixture of GFlowNets (MoG) and MoG exchange (MoGX) at a glance.** A community of K GFlowNets is trained in parallel against a target $R(x)$, coupled to one jointly trained soft classifier f_ψ that softly partitions the state space. Each GFlowNet trains on the *shaped* reward (blue arrows). The classifier is the *only* object whose gradients cross rank boundaries. **MoGX** routes part of the on-policy trajectory to other ranks based on the classifier. Post-training, the Z-weighted mixture provably recovers $p(x) \propto R(x)$.

Existing work scales GFlowNet training asynchronously without addressing this bias. Asynchronous trajectory balance (Bartoldson et al., 2025) spreads off-policy searchers across nodes that feed a shared buffer, but the searchers are snapshots of one policy and share its reachable set. Embarrassingly Parallel GFlowNets (Da Silva et al., 2024) train one local GFlowNet per factor of a product reward $R(\cdot) \propto R_1(\cdot) \cdots R_N(\cdot)$ to distribute reward evaluation, not to cover a shared target better. Distributed RL (Espeholt et al., 2018; Horgan et al., 2018) and population-based training (Jaderberg et al., 2017) parallelize agents without encouraging specialization (section A).

We attack mode bias with a population of policies that divide the state space between them. **Mixture of GFlowNets (MoG)** replaces the single forward policy with a community of K GFlowNets $\Pi_\theta = \{\pi_{\theta_1}, \ldots, \pi_{\theta_K}\}$ trained in parallel and coupled to one jointly trained soft classifier $f_\psi : \mathcal{X} \to \Delta^{K-1}$ that partitions $\mathcal{X}$. Component k trains on the shaped reward $\log \tilde{R}_k(x) = \log R(x) + \log f_\psi(x)_k$, and mixing the components under their learned partition functions recovers $p(x) \propto R(x)$ exactly. Only the small classifier's gradients are all-reduced across ranks, bounding communication costs, and the classifier pushes each GFlowNet toward its own subregion. This removes both the parameter coupling that limits data-parallel training and the redundancy that limits independent training. **MoG exchange (MoGX)** goes further and sends each on-policy trajectory to the rank the classifier assigns it to, through an MPI all-to-all keyed on $\arg\max_j \log f_\psi(x)_j$. GFlowNet losses accept off-policy data, so the exchange needs no importance correction. Figure 1 shows the full pipeline.

2 BACKGROUND

GFlowNets. Let $\mathcal{G} = (\mathcal{S}, \mathcal{A})$ be a finite directed acyclic graph with a unique source s_0 and terminal states $\mathcal{X} \subseteq \mathcal{S}$ carrying a non-negative reward $R : \mathcal{X} \to \mathbb{R}_{\geq 0}$. A GFlowNet (Bengio et al., 2021; 2023) parameterizes a forward policy $P_F(s' \mid s)$ and a backward policy $P_B(s \mid s')$, trained so that the marginal over terminal states induced by following P_F from s_0 satisfies $p_\theta(x) \propto R(x)$. The trajectory balance (TB) objective (Malkin et al., 2022) enforces, for every complete trajectory $\tau = (s_0, \ldots, s_T = x)$,

$$Z \cdot \prod_{t=0}^{T-1} P_F(s_{t+1} \mid s_t) \;=\; R(x) \cdot \prod_{t=1}^{T} P_B(s_{t-1} \mid s_t),$$

with $Z > 0$ a learnable estimate of $\sum_x R(x)$. Detailed balance (Bengio et al., 2023) and sub-trajectory balance (Madan et al., 2023) give alternative decompositions of the same constraint; our framework does not depend on the choice of training objective.

Single-learner coverage and the mode-bias problem. The guarantee $p_\theta(x) \propto R(x)$ holds at the global optimum of the balance objective, which requires the balance conditions on *every* trajectory. However, training only enforces them on trajectories the behavior distribution visits. On multi-modal landscapes, that distribution often concentrates on a few high-reward regions, so the conditions can be badly violated on undiscovered modes while the empirical loss looks converged (Dall'Antonia et al., 2025). The remedies in section 1 change how a single learner explores; some of them sample off-policy, but in all of them the behavior distribution is generated by, or seeded from, one set of learned parameters. The learner's current beliefs alone decide which regions receive gradient signal. We make this precise with the (ϵ, T)-*reachable set* at iteration i,

$$\mathcal{R}_{\epsilon,T}^{\pi_\theta^{(i)}}(s_0) \;:=\; \left\{ x \in \mathcal{X} : \Pr\left[s_0 \rightsquigarrow x \text{ in} \leq T \text{ steps under } \pi_\theta^{(i)}\right] \geq \epsilon \right\},$$

where $\pi_\theta^{(i)}$ is the forward policy after i training iterations. A softmax policy gives every x positive probability, so coverage failure is about non-negligible probability under a realistic budget: $\mathcal{R}_{\epsilon,T}^{\pi_\theta^{(i)}}(s_0) \subsetneq \mathcal{X}$ whenever the path to some x needs transitions whose product of probabilities falls below ϵ. Larger batches help only marginally. Extra trajectories are drawn i.i.d. from the same behavior distribution, so the expected cost of reaching an excluded mode scales as $1/\epsilon$, exponentially in the number of low-probability transitions on its path. On-policy deep RL shows the same saturation under batch scaling (Singla et al., 2024).

Decorrelating failures across a population. Every policy will miss some modes, so we aim for policies that miss different ones, leaving a mode lost only when all of them miss it. N independently initialized policies $\{\pi_\theta^i\}_{i=1}^N$ sample from the union $\bigcup_i \mathcal{R}^{\pi_\theta^i}(s_0)$, which is larger than a single reachable set only if the policies fail *differently*. Independent initialization does not guarantee that, since policies trained on the same reward can still settle on the same modes. So we push them apart explicitly, making their reachable sets overlap as little as possible.

3 METHODS

3.1 MIXTURE FRAMEWORK

We train a community $\Pi_\theta = \{\pi_{\theta_1}, \ldots, \pi_{\theta_K}\}$ in parallel on a target reward $R : \mathcal{X} \to \mathbb{R}_{\geq 0}$ over a finite DAG $\mathcal{G} = (\mathcal{S}, \mathcal{A})$ with source s_0. Component k owns forward and backward policies and a learnable log-partition $\log Z_k$ (for DB and SubTB, the source log-flow $\log F^{(k)}(s_0)$ plays the same role). The total budget is fixed: instead of T trajectories from one policy, each component gets T/K. The components are coupled through one soft classifier $f_\psi : \mathcal{X} \to \Delta^{K-1}$ trained jointly with them, and component k trains on the *shaped* log-reward

$$\log \tilde{R}_k(x) \;=\; \log R(x) \;+\; \log f_\psi(x)_k, \tag{1}$$

so that $\pi_{\theta_k}(x) \propto \tilde{R}_k(x)$ at convergence. Mixing the components under their learned partition functions,

$$\mathcal{P}_\Pi(\pi_k) \;=\; \frac{Z_k}{\sum_{j=1}^K Z_j}, \qquad p_\Pi(x) \;=\; \sum_{k=1}^K \mathcal{P}_\Pi(\pi_k)\, \pi_{\theta_k}(x) \;\propto\; R(x), \tag{2}$$

recovers the original target.

Proposition (mixture correctness). *If each component reaches its shaped target,* $\pi_{\theta_k}(x) = \tilde{R}_k(x)/Z_k$ *with* $Z_k = \sum_x R(x) f_\psi(x)_k$*, and* $\sum_k f_\psi(x)_k = 1$ *pointwise, then* $p_\Pi(x) = R(x)/\sum_{x'} R(x')$.

Proof. $p_\Pi(x) = \sum_k \frac{Z_k}{\sum_j Z_j} \cdot \frac{R(x) f_\psi(x)_k}{Z_k} = \frac{R(x)}{\sum_j Z_j}$, and $\sum_j Z_j = \sum_x R(x) \sum_j f_\psi(x)_j = \sum_x R(x)$. □

The result holds for *any* pointwise simplex assignment; f_ψ only redistributes mass. Since $\tilde{R}_k$ is supported on $\{x : f_\psi(x)_k > 0\}$, a set the classifier can align with component k's reachable set, each component's convergence assumption can be *easier* to meet than reaching all of $\mathcal{X}$. After training, we form $w_k = \mathrm{softmax}(\log Z)_k$, sample $\pi_k \sim \mathrm{Categorical}(w)$ and then $x \sim \pi_{\theta_k}$; section 4 checks that this reproduces $p(x) \propto R(x)$ (fig. 4). We train with trajectory balance, but any objective with a log-partition estimate works.

3.2 Mixture of GFlowNets (MoG)

MoG jointly trains K GFlowNets and one soft classifier. The classifier is the *only* object whose gradients are all-reduced across ranks. Each GFlowNet otherwise samples its own data and takes its own updates, which keeps communication minimal and lets MoG train asynchronously.

Partition classifier and reward shaping. f_ψ reads the same preprocessed state $\phi(x)$ as the policies and outputs K logits. Its gradients are all-reduced before each step, so its parameters stay identical across ranks and the shaped rewards stay coherent (section C). Rank k replaces the log-reward of every terminating state with eq. (1) at loss time, so the shaping always reflects the *current* classifier, even for states buffered many iterations ago.

Classifier objective. Each rank trains f_ψ by cross-entropy, labeling every on-policy state with its own component index:

$$\mathcal{L}^{(k)}_{f_\psi}(\psi) \;=\; -\frac{1}{|\mathcal{B}^{(k)}|} \sum_{x \in \mathcal{B}^{(k)}} \log f_\psi(x)_k \;+\; \beta\, \mathcal{H}\big(f_\psi(x)\big), \tag{3}$$

with $\mathcal{H}$ the softmax entropy and $\beta \geq 0$. All-reduced across ranks, this is supervised classification on the union $\bigcup_k \{(x, k) : x \in \mathcal{B}^{(k)}\}$, whose Bayes-optimal solution is $f_\psi(x)_k = P(k \mid x) \propto \pi_{\theta_k}(x)$. As each π_{θ_k} specializes, the classifier sharpens, and eq. (1) reinforces that specialization on the next iteration, with eq. (2) as the fixed point.

3.3 Trajectory Exchange (MoGX)

MoG changes each component's *training signal* but not its *experience collection*: rank k still samples from π_{θ_k} alone, which early in training may rarely visit its assigned subregion. MoGX sends each on-policy trajectory to the component the classifier assigns it to.

Selection and transport. After sampling its batch $\mathcal{B}^{(k)}$, rank k computes each terminating state's destination

$$k^\star(x_i) \;=\; \arg\max_{j \in \{1,\dots,K\}} \log f_\psi(x_i)_j. \tag{4}$$

From the states assigned elsewhere, $\mathcal{E}^{(k)} = \{x_i \in \mathcal{B}^{(k)} : k^\star(x_i) \neq k\}$, it sends the $\min(N, |\mathcal{E}^{(k)}|)$ trajectories with the *smallest* $\log f_\psi(x_i)_k$ (those the classifier is most sure do not belong to k) to their destination ranks. N caps the per-step communication volume, and components outside an exchange keep sampling and training independently. TB (Malkin et al., 2022), DB (Bengio et al., 2023), SubTB (Madan et al., 2023) and ZVar (Zhang et al., 2023) reach zero loss only when the balance conditions hold on every trajectory the training distribution can produce. Assuming the model can reach zero loss and the training distribution has full support, every such distribution gives the same global minimizer, so exchanged trajectories need no importance-sampling correction. See Section 3.3 for the MoGX algorithm. Steps unique to MoGX are marked †, removing them recovers MoG, removing both the exchange step and the reward shaping recovers NoComm, and adding a

gradient all-reduce on policy parameters recovers DDP. Section C additional and communication details.

Algorithm 1: One MoGX iteration on training rank k.

1. **Sample.** Roll out $|\mathcal{B}|$ trajectories from π_{θ_k}.
2. **Exchange**. Compute $k^\star(x_i)$ and self-log-probs via f_ψ; select n_{exch} low-self-probability misrouted trajectories per eq. (4); `MPI_Alltoall` across the training group; merge received trajectories into the local batch.
3. **Store.** Insert the merged batch into the local replay buffer $\mathcal{D}$.
4. **Mix.** $\mathcal{B} \leftarrow \mathcal{B}_{\text{on}}^{(1-\rho)B} \cup \mathcal{B}_{\mathcal{D}}^{\rho B}$, i.e., a fraction ρ of the batch is sampled from the replay buffer $\mathcal{D}$ and the rest from the current on-policy data ($\rho = 0$: on-policy only).
5. **Shape rewards.** Replace $\log R(x)$ by $\log \tilde{R}_k(x)$ via eq. (1), using current f_ψ.
6. **GFlowNet loss.** Compute $\mathcal{L}_{\text{GFN}}^{(k)}(\theta_k)$ on the shaped batch; do a backward pass and the optimizer step.
7. **Classifier step.** Compute eq. (3); backward pass; all-reduce $\nabla_\psi \mathcal{L}_{f_\psi}^{(k)}$ across training ranks; optimizer step on f_ψ.

4 Experiments

We compare MoG and MoGX with three population baselines on HyperGrid under six reward landscapes, with community sizes spanning two orders of magnitude. The rewards differ in how local or non-local their structure is, which lets us see how the landscape interacts with each method. We then turn to a genomic ideotype-design environment whose optima are placed by a real genetic map (section 4.1).

Environments. HyperGrid (Bengio et al., 2021) with `ndim`=10 under six rewards in two regimes. The *non-tiered* rewards follow prior GFlowNet work: *original* (corner modes, Bengio et al., 2021, H=10, R_0=0.01), *cosine* (a regular cosine-shaped pattern of high-reward regions, H=16) and *sparse* (tiny isolated corner modes from GAFN, Pan et al., 2023, H=16). The *tiered/compositional* rewards (H=16) make structure found in one region determine where reward lies elsewhere. In *bitwise_xor*, a global $\mathrm{GF}(2)$ parity rule, enforced at progressively stricter bit-planes, scatters modes with no spatial gradient; each policy must infer the same non-local rule, which then predicts every remaining mode. In *multiplicative_coprime*, each tier adds a different number-theoretic constraint (prime support, exponent caps, cross-coordinate coprimality, LCM targets), so policies must compose heterogeneous constraints, and factorizations learned near the origin are prerequisites for uncovering the rarer high-tier modes. In *conditional_multiscale*, each scale's constraint depends on the coarser scales already resolved (a coarse-to-fine shifted-digit filter), so modes must be found in a fixed coarse-to-fine order. The partition function grows as H^{10}, which puts all but the smallest grids in the mode-bias regime of section 2. These *tiered/compositional* rewards are much harder than earlier HyperGrid variants (see the appendix of Richardson et al., 2026 for an initial version of these environments); section D.1 gives full details.

Baselines. All five methods share the per-policy architecture, and each agent runs for 1.31×10^8 trajectories (section D). They differ only in how policies coordinate. **NoComm**: K independent GFlowNets. **DDP**: K agents with all-reduced gradients, i.e., an effective batch of $K \cdot 1024$. **EGFN** (Ikram et al., 2025): rank 0 is the only gradient-trained GFlowNet; the other $K-1$ ranks form a population evolved every iteration by selection, crossover and mutation, and each member sends its 64 best trajectories to the learner's prioritized replay buffer (ported from the official code and adapted to our setup; see below and table 2). **MoG** (ours): classifier-driven reward shaping, with only classifier gradients all-reduced. **MoGX** (ours): MoG plus exchange of 128 of 1024 trajectories per step. We sweep $K \in \{2, 4, 8, 16, 32, 64\}$ and read every run at the same *total* budget, summed over agents: 2.62×10^8 trajectories, where a K=2 community stops. This is deliberate: we target settings where oracle queries are the scarce resource, so the question is how best to spend a fixed

number of them. The cost is that agents in larger communities train for less: at $K{=}64$ each agent sees 4.1×10^6 trajectories (4,000 gradient steps), against 128,000 steps at $K{=}2$. A single GFlowNet ($K{=}1$, where NoComm, DDP, MoG and MoGX coincide) stops at 1.31×10^8 trajectories, half the shared budget, so it is not in the endpoint plots. We report the cumulative number of unique modes the community finds, as mean ± 1 s.d. over 4 seeds; discovery curves are in section F.

Adapting EGFN. EGFN is hard to compare fairly. The original pairs a learner batch of 16 with 5 population agents sampling 4 trajectories each. For fairness, we match the algorithm hyperparameters: K networks, 1024 trajectories per rank per iteration, and the same number of gradient steps. At this batch size EGFN has a weakness. Its mutation adds $\mathcal{N}(0, 1)$ noise to whole weight tensors, which makes a population policy nearly deterministic: after one mutation, 1024 samples cover 2 distinct terminal states, so most of the population budget is wasted. We run EGFN only off-policy, because the population's trajectories reach the learner only through the replay buffer.

Results (off-policy, replay mix ratio $\rho = 0.5$). Each training batch mixes fresh on-policy trajectories and replayed samples half and half. Figure 2 shows the headline result: MoG and MoGX generally improve with K, while NoComm and DDP use trajectories less efficiently as agents are added. The discovery curves (section F) show why. NoComm plateaus quickly because independent agents rediscover the same modes, and DDP saturates soon after because every agent shares the same stale parameters and reachable set, as Singla et al. (2024) found for on-policy RL. MoG scales better because the soft partition pushes each component into its own subregion. MoGX generally improves over MoG, since exchange gives every component in-subregion samples from the first iteration. The gap grows with the number of modes: on *conditional_multiscale* and *bitwise_xor*, where policy bias hurts most, MoGX keep its coverage as K grows while NoComm and DDP drop by up to an order of magnitude. On *sparse*, no method gains from larger K. With few, distant modes, the bottleneck is reaching any mode at all, and partitioning cannot help with that. EGFN is strongest at small K: it leads on *original* at $K{=}2$, and on *sparse* it is the only method to find every mode, up to $K{=}8$. It then degrades quickly, because at a fixed total budget its single learner takes fewer steps as K grows and the collapsed population does not make up for it.

Pure on-policy ($\rho = 0.0$). We rerun the sweep without a replay buffer, so every batch comes from the current policies (fig. 3). Dropping replay does not change how the methods scale: each method loses coverage with K at almost the same rate as in fig. 2, and the methods rank in the same order. MoG and MoGX keep their lead over NoComm and DDP without replay.

Per-iteration cost. Table 1 gives the mean wall-clock time per iteration on *original*, pure on-policy, over 4 seeds (per-reward and per-phase numbers in section H; table 9 is the phase breakdown for the same runs). Rank-local work (sampling, the loss forward pass with reward shaping, the backward pass) stays flat in K at $\sim$700 ms for every method, so a larger community adds no per-rank compute. What grows with K is cross-rank traffic. DDP's policy-gradient all-reduce climbs from 24 ms at $K{=}2$ to 549 ms at $K{=}64$. MoG has *no* policy-gradient sync, only all-reduce on a two-hidden-layer MLP classifier. At $K{=}64$ MoG's is 22% faster (976 vs. 1249 ms). MoGX is slower than DDP at every K, and at $K{=}64$ its exchange (760 ms) costs more than the rank-local work. The overhead comes from our implementation, which calls mpi4py's pickle-based `alltoall`, serializing every payload as a Python object, and makes every rank wait for the slowest one. A buffer-based, asynchronous exchange would avoid both costs. Small differences between methods are run-to-run noise.

Method	$K{=}1$	2	4	8	16	32	64
NoComm	702 ± 4	729 ± 6	726 ± 7	728 ± 7	732 ± 3	758 ± 23	713 ± 9
DDP	702 ± 4	715 ± 3	743 ± 2	784 ± 1	823 ± 2	950 ± 4	1249 ± 11
EGFN	702 ± 4	712 ± 3	748 ± 23	853 ± 14	876 ± 70	803 ± 6	975 ± 3
MoG	702 ± 4	721 ± 4	749 ± 7	800 ± 3	818 ± 6	923 ± 8	976 ± 7
MoGX	702 ± 4	762 ± 30	804 ± 15	841 ± 12	998 ± 12	1215 ± 49	1477 ± 12

Table 1: Mean per-iteration wall-clock time (ms) by method and K, pure on-policy, *original* reward, over 4 seeds and all steps. NoComm stays flat, MoG grows with the non-blocking classifier all-reduce, DDP with the ∇_θ all-reduce, and MoGX adds the exchange all-to-all. At $K{=}1$ all methods are the same loop.

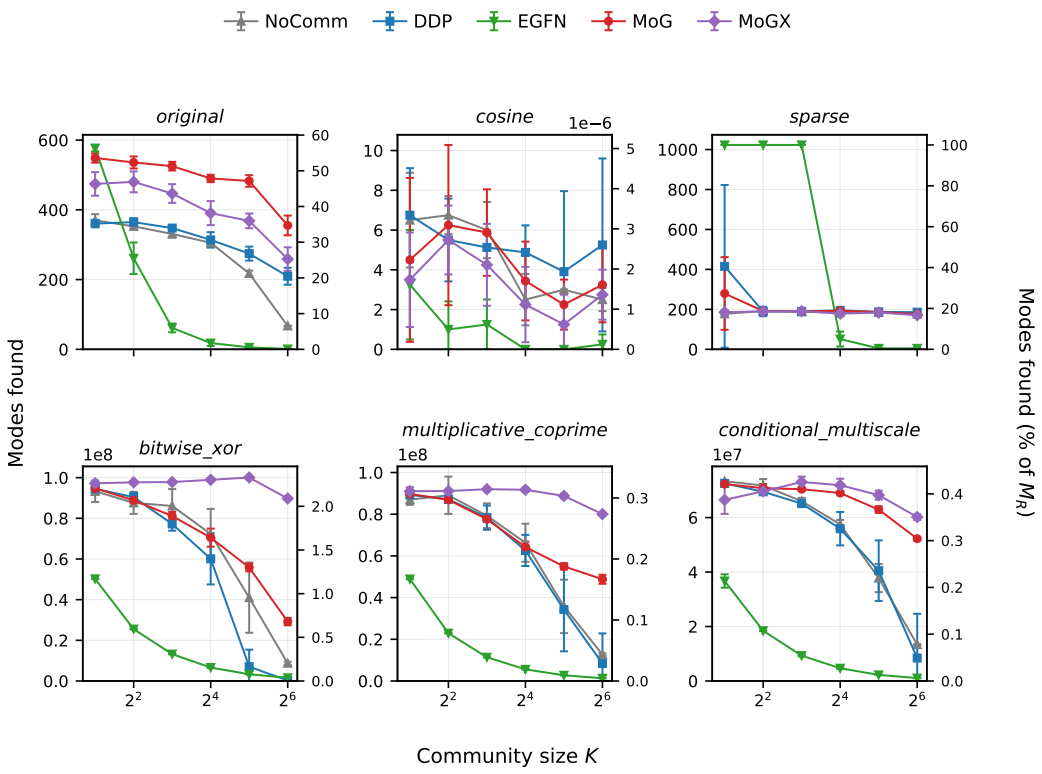


Figure 2: Modes found at a fixed total-trajectory budget vs community size K (log K axis; right axis: % of all modes), **off-policy regime** (replay mix ratio 0.5). MoG (red) and MoGX (purple) scale with K while DDP (blue), NoComm (grey), and EGFN (green) tend to decline; MoGX gains most on the structured rewards.

Empirical validation of mixture correctness. The proposition in section 3.1 assumes every component exactly reaches its shaped target. We test whether a finitely trained $K{=}4$ MoGX checkpoint (*original*, ndim=10, $H{=}7$, 1024 corner modes) still gives $p(x) \propto R(x)$ (setup in section E). Figure 4 shows each rank far from the optimum, as intended: each rank targets its own $\tilde{R}_k$ and puts mass on a distinct, sparse subset of the modes. The Z-weighted mixture nonetheless spreads mass almost uniformly over all 1024 modes, with $\mathrm{KL}(p_\Pi \,\|\, \pi^\star) = 1.51 \times 10^{-4}$. The learned $\log Z_k$ are accurate enough that biased, classifier-shaped sub-policies combine into an unbiased sampler.

4.1 Genomic ideotype design

Plant breeding is an AI-for-Science problem that needs a large set of high-reward states. Breeders want *many* distinct genotypes that meet a target trait profile, because each candidate must be crossed, grown and phenotyped over several field seasons before its value is known (Tester & Langridge, 2010), and because a genetically diverse portfolio protects against disease and climate (Esquinas-Alcázar, 2005). Donald (1968) introduced the *ideotype*, a model plant defined by the traits expected to maximize yield. Genomic ideotype design carries the idea to the DNA level: one assembles a target genotype by pyramiding beneficial alleles at known loci. Alleles close together on a chromosome are inherited together, though, and cannot be recombined freely. We evaluate MoG and MoGX in this setting.

Environment. A state is a dosage vector $x \in \{0, \ldots, P{-}1\}^L$ over $L = B \cdot b$ loci, starting from $\mathbf{0}_L$. Each action raises the dosage at one locus by one (masked at the top dosage), and termination is legal at every state, so $|\mathcal{X}| = P^L$. The loci are partitioned into B blocks of size b; a block is *satisfied* when its dosage sums to exactly $t^\star$, with any excess counting as the double-dosage failure

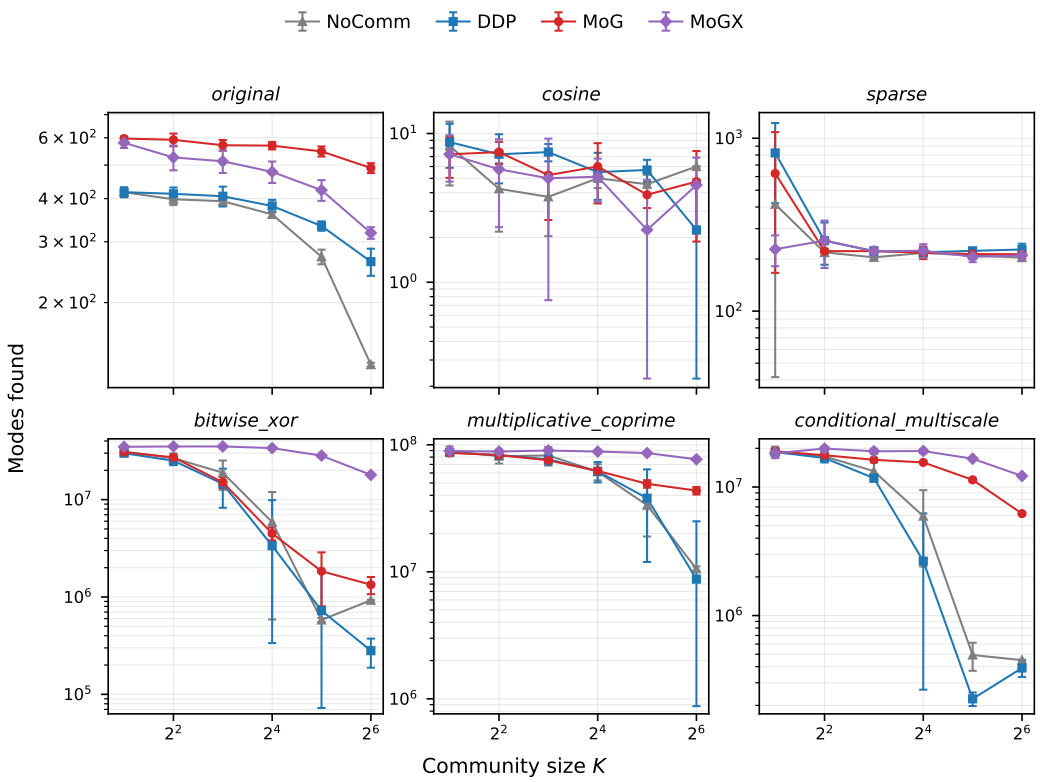


Figure 3: As fig. 2, **pure on-policy regime**. MoG and MoGX still separate from DDP and NoComm, and the MoGX–MoG gap is wider without replay.

case. Writing $s(x)$ for the number of satisfied blocks,

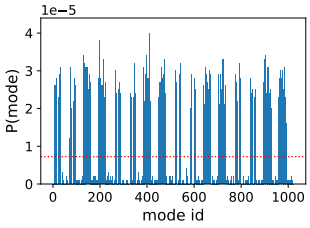
$$\log R(x) = \begin{cases} \beta\, s(x) \;-\; \gamma \sum_{i=1}^{L} w_i\, x_i, & s(x) > 0, \\ \log R_0, & \text{otherwise.} \end{cases} \tag{5}$$

The weights $w_i = -\log(r_i + \epsilon)$ tie the landscape to biology: r_i is the Haldane recombination fraction (Haldane, 1919) between locus i and its block's anchor, read off the wheat genetic map of Wang et al. (2014), as shipped with ChromaX (Younis et al., 2023). Tightly linked loci get large weights, so the penalty measures how improbable the needed recombination is. We use B=10, b=2, P=2, $t^\star$=1, β=4, γ=1.

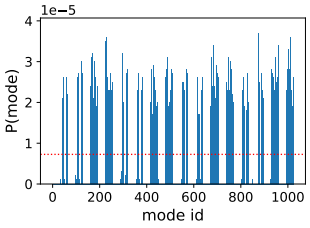
Results. Figure 5 reports modes found at a budget of 2.62×10^8 trajectories, with and without replay, over 4 seeds (curves in figs. 18 and 19). Both regimes agree: MoGX leads at every K, MoG matches NoComm, and DDP falls behind, as the mode-bias argument of section 2 predicts.

5 Conclusion

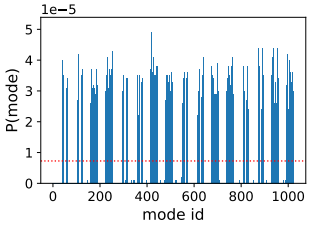
Larger batches and data-parallel gradient averaging do not enlarge the set of regions a distributed system can reach. **MoG** targets that reachable set directly: a community of K GFlowNets, coupled through one jointly trained classifier, each learns a distinct subregion of the target. The mixture is provably correct, and we showed empirically that biased, classifier-shaped sub-policies recombine into an unbiased sampler. **MoGX** adds classifier-driven trajectory exchange, which gives each component in-subregion experience from the first iteration. With the environment-interaction budget held fixed, both methods turn parallel compute into mode coverage. Across six HyperGrid rewards and up to 64 policies, MoG and MoGX generally keep finding modes as K grows while DDP saturates and independent agents plateau, with or without replay, and MoG synchronizes more cheaply than DDP. On ideotype design, with optima separated by linkage from a real wheat map, MoGX reaches 39% of the 1024 optima at K=64, while MoG alone only matches independent agents, likely because of the reward's geometry.



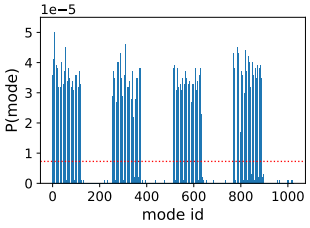
(a) Rank 0, $\log Z_0$=12.20, $\mathrm{KL}(\pi_{\theta_0} \,\|\, \pi^\star) = 8.4\times10^{-3}$.



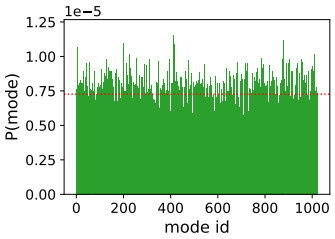
(b) Rank 1, $\log Z_1$=12.18, $\mathrm{KL}(\pi_{\theta_1} \,\|\, \pi^\star) = 8.1\times10^{-3}$.



(c) Rank 2, $\log Z_2$=11.92, $\mathrm{KL}(\pi_{\theta_2} \,\|\, \pi^\star) = 1.03\times10^{-2}$.

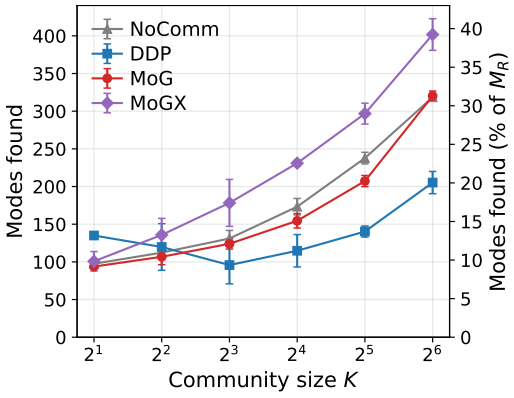


(d) Rank 3, $\log Z_3$=11.90, $\mathrm{KL}(\pi_{\theta_3} \,\|\, \pi^\star) = 1.11\times10^{-2}$.

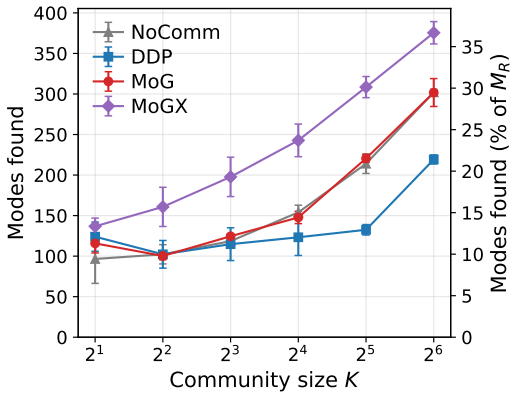


(e) Z-weighted mixture, $\mathrm{KL}(p_\Pi \,\|\, \pi^\star) = 1.51\times10^{-4}$.

Figure 4: Mixture correctness (eq. (2)) for a K=4 MoGX checkpoint on HyperGrid *original*. Bars are empirical per-mode marginals from 10^6 on-policy trajectories; the dotted red line is the optimal policy $\pi^\star$ (uniform, 7.26×10^{-6} per mode). Each rank covers a distinct sparse subset of modes; the Z-weighted mixture recovers $\pi^\star$.



(a) Replay regime (ρ=0.5).



(b) Pure on-policy regime (ρ=0).

Figure 5: Modes found at a fixed total-trajectory budget vs K on the ChromaX ideotype environment (B=10, β=4; 1024 optima), mean ±1 s.d. over 4 seeds. MoGX (purple) leads at every K, MoG (red) tracks NoComm (grey), and DDP (blue) is generally last. Dropping replay costs MoGX, MoG and NoComm 5−7% of their K=64 coverage and gains DDP 7%; no ordering changes.

Two limitations point to future work. Both environments share the same state-space structure, so we have not tested whether a learned partition works for graph or sequence generation with larger action spaces and variable trajectory lengths. And the community is static: K is fixed in advance, with no way to retire or reallocate stalled components; a discovery-rate signal could drive that resizing. The obvious next step is molecule and biological-sequence design, where oracle calls are expensive and structural diversity is the point. The framework does not depend on the balance objective or the form of the partition, and we expect its core idea, giving each parallel agent its own learned sub-target, to carry over.

6 Reproducibility Statement

We will release all code, including model training code, environments, and analysis scripts. These artifacts are intended to allow the community to inspect the work, extend the framework to new domains, and benchmark the method easily on more challenging problems.

7 AI Assistance Disclosure

Large language models (LLMs) were used to assist with manuscript drafting (primarily consistency checks, accuracy of claims validation, and LaTeX automation), and for code generation for model evaluations. Many elements of this codebase predate the advent of human-level coding agents, so most core methods were implemented entirely by hand. LLMs were used under heavy supervision for figure and table generation. They were not used for research ideation, method design, or experiment design. The authors reviewed all code underlying reported results and rewrote any generated manuscript text by hand.

REFERENCES

Brian R Bartoldson, Siddarth Venkatraman, James Diffenderfer, Moksh Jain, Tal Ben-Nun, Seanie Lee, Minsu Kim, Johan Obando-Ceron, Yoshua Bengio, and Bhavya Kailkhura. Trajectory balance with asynchrony: Decoupling exploration and learning for fast, scalable LLM post-training. *arXiv preprint arXiv:2503.18929*, 2025.

Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation. *Advances in Neural Information Processing Systems*, 34:27381–27394, 2021.

Yoshua Bengio, Salem Lahlou, Tristan Deleu, Edward J Hu, Mo Tiwari, and Emmanuel Bengio. GFlowNet foundations. *Journal of Machine Learning Research*, 24(210):1–55, 2023.

Tiago Da Silva, Luiz Max Carvalho, Amauri Souza, Samuel Kaski, and Diego Mesquita. Embarrassingly parallel gflownets. *arXiv preprint arXiv:2406.03288*, 2024.

Pedro Dall'Antonia, Tiago da Silva, Daniel Augusto de Souza, César Lincoln C. Mattos, and Diego Mesquita. Boosted GFlowNets: Improving exploration via sequential learning, 2025. URL https://arxiv.org/abs/2511.09677.

Tristan Deleu, Padideh Nouri, Nikolay Malkin, Doina Precup, and Yoshua Bengio. Discrete probabilistic inference as control in multi-path environments. In *UAI*, volume 244 of *PMLR*, 2024.

C. M. Donald. The breeding of crop ideotypes. *Euphytica*, 17(3):385–403, 1968.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, 2018.

José Esquinas-Alcázar. Protecting crop genetic diversity for food security: political, ethical and technical challenges. *Nature Reviews Genetics*, 6(12):946–953, 2005.

Timur Garipov, Sebastiaan De Peuter, Ge Yang, Vikas Garg, Samuel Kaski, and Tommi Jaakkola. Compositional sculpting of iterative generative processes. In *Advances in Neural Information Processing Systems*, 2023.

John Burdon Sanderson Haldane. The combination of linkage values, and the calculation of distances between the loci of linked factors. *Journal of Genetics*, 8(29):299–309, 1919.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay. In *International Conference on Learning Representations*, 2018.

Zarif Ikram, Ling Pan, and Dianbo Liu. Evolution guided generative flow networks, 2025. URL https://arxiv.org/abs/2402.02186.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Moksh Jain, Emmanuel Bengio, Alex Hernandez-Garcia, Jarrid Rector-Brooks, Bonaventure FP Dossou, Chanakya Ajit Ekbote, Jie Fu, Tianyu Zhang, Michael Kilgour, Dinghuai Zhang, et al. Biological sequence design with GFlowNets. In *International Conference on Machine Learning*, 2022.

Minsu Kim, Taeyoung Yun, Emmanuel Bengio, Dinghuai Zhang, Yoshua Bengio, Sungsoo Ahn, and Jinkyoo Park. Local search GFlowNets. *arXiv preprint arXiv:2310.02710*, 2023.

Salem Lahlou, Tristan Deleu, Pablo Lemos, Dinghuai Zhang, Alexandra Volokhova, Alex Hernández-García, Léna Néhale Ezzine, Yoshua Bengio, and Nikolay Malkin. A theory of continuous generative flow networks. In *ICML*, PMLR, 2023.

Kanika Madan, Jarrid Rector-Brooks, Maksym Korablyov, Emmanuel Bengio, Moksh Jain, Andrei Cristian Nica, Tom Bosc, Yoshua Bengio, and Nikolay Malkin. Learning GFlowNets from partial episodes for improved convergence and stability. In *International Conference on Machine Learning*, 2023.

Kanika Madan, Alex Lamb, Emmanuel Bengio, Glen Berseth, and Yoshua Bengio. Towards improving exploration through sibling augmented GFlowNets. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=HH4KWP8RP5.

Nikolay Malkin, Moksh Jain, Emmanuel Bengio, Chen Sun, and Yoshua Bengio. Trajectory balance: Improved credit assignment in GFlowNets. In *Advances in Neural Information Processing Systems*, 2022.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 2016.

Johan Obando Ceron, Marc Bellemare, and Pablo Samuel Castro. Small batch deep reinforcement learning. *Advances in Neural Information Processing Systems*, 36:26003–26024, 2023.

Johan Obando-Ceron, Ghada Sokar, Timon Willi, Clare Lyle, Jesse Farebrother, Jakob Foerster, Gintare Karolina Dziugaite, Doina Precup, and Pablo Samuel Castro. Mixtures of experts unlock parameter scaling for deep RL. In *International Conference on Machine Learning*, 2024.

Ling Pan, Dinghuai Zhang, Aaron Courville, Longbo Huang, and Yoshua Bengio. Generative augmented flow networks. In *ICLR*, 2023.

Ling Pan, Moksh Jain, Kanika Madan, and Yoshua Bengio. Pre-training and fine-tuning generative flow networks. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ylhiMfpqkm.

Jack Parker-Holder, Aldo Pacchiano, Krzysztof Choromanski, and Stephen Roberts. Effective diversity in population based reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

Jarrid Rector-Brooks, Kanika Madan, Moksh Jain, Maksym Korablyov, Cheng-Hao Liu, Sarath Chandar, Nikolay Malkin, and Yoshua Bengio. Thompson sampling for improved exploration in GFlowNets. *arXiv preprint arXiv:2306.17693*, 2023.

Oliver E Richardson, Mandana Samiei, Mehran Shakerinava, Joseph D Viviano, Abdessamad El Kabid, Ali Parviz, and Yoshua Bengio. Local inconsistency resolution: The interplay between attention and control in probabilistic models. *arXiv preprint arXiv:2604.17140*, 2026.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *International Conference on Learning Representations (ICLR)*, 2016. arXiv:1511.05952.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.

Max W. Shen, Emmanuel Bengio, Ehsan Hajiramezanali, Andreas Loukas, Kyunghyun Cho, and Tommaso Biancalani. Towards understanding and improving GFlowNet training. In *ICML*, 2023.

Tiago Silva, Amauri H Souza, Luiz Max Carvalho, Samuel Kaski, and Diego Mesquita. Federated contrastive GFlowNets. OpenReview preprint, 2024. URL https://openreview.net/forum?id=VJDFhkwQg6.

Tiago Silva, Amauri Souza, Omar Rivasplata, Vikas Garg, Samuel Kaski, and Diego Mesquita. Generalization and distributed learning of GFlowNets. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=PJNhZoCjLh.

Jayesh Singla, Ananye Agarwal, and Deepak Pathak. SAPG: Split and aggregate policy gradients. *arXiv preprint arXiv:2407.20230*, 2024.

Mark Tester and Peter Langridge. Breeding technologies to increase crop production in a changing world. *Science*, 327(5967):818–822, 2010.

Daniil Tiapkin, Nikita Morozov, Alexey Naumov, and Dmitry Vetrov. Generative flow networks as entropy-regularized RL. In *International Conference on Artificial Intelligence and Statistics*, 2024a.

Daniil Tiapkin, Nikita Morozov, Alexey Naumov, and Dmitry P. Vetrov. Generative flow networks as entropy-regularized rl. In *AISTATS*, volume 238 of *PMLR*, pp. 4213–4221, 2024b.

Francisco Vargas, Will Grathwohl, and Arnaud Doucet. Denoising diffusion samplers. In *ICLR*, 2023.

Nikhil Vemgal, Elaine Lau, and Doina Precup. An empirical study of the effectiveness of using a replay buffer on mode discovery in GFlowNets. In *ICML 2023 Workshop on Structured Probabilistic Inference & Generative Modeling (SPIGM)*, 2023. arXiv:2307.07674.

Joseph D. Viviano, Omar G. Younis, Sanghyeok Choi, Victor Schmidt, Yoshua Bengio, and Salem Lahlou. torchgfn: A PyTorch GFlowNet library. *arXiv preprint arXiv:2305.14594*, 2023.

Shichen Wang et al. Characterization of polyploid wheat genomic diversity using a high-density 90,000 single nucleotide polymorphism array. *Plant Biotechnology Journal*, 12(6):787–796, 2014.

Omar G Younis, Matteo Turchetta, Daniel Ariza Suarez, Steven Yates, Bruno Studer, Ioannis N Athanasiadis, Andreas Krause, Joachim M Buhmann, and Luca Corinzia. Chromax: a fast and scalable breeding program simulator. *Bioinformatics*, 39(12):btad691, 2023.

David W. Zhang, Corrado Rainone, Markus Peschl, and Roberto Bondesan. Robust scheduling with GFlowNets. In *International Conference on Learning Representations*, 2023.

Qinsheng Zhang and Yongxin Chen. Path integral sampler: A stochastic control approach for sampling. In *ICLR*, 2022.

A Extended Related Work

GFlowNets, MaxEnt RL and continuous samplers. The mode collapse discussed in section 2 is specific to return maximization. *Entropy-regularized* (MaxEnt) RL instead learns a stochastic policy that samples in proportion to reward, and GFlowNet training can be recast as a MaxEnt RL problem with a particular reward and regularizer (Tiapkin et al., 2024b). In the multi-path environments typical of sequential construction, this equivalence requires correcting the reward so that the optimal policy's marginal is proportional to $R(x)$ regardless of the underlying MDP (Deleu et al., 2024). Sampling from an unnormalized density is also the aim of diffusion and stochastic-control samplers (Zhang & Chen, 2022; Vargas et al., 2023), to which continuous GFlowNets (Lahlou et al., 2023) are closely related. We use GFlowNets because they match the multi-path, discrete structure of our construction processes, not because we see them as a separate paradigm.

Parallel and population-based training in RL. Distributed actor-learner architectures such as A3C (Mnih et al., 2016), IMPALA (Espeholt et al., 2018), and Ape-X (Horgan et al., 2018) scale experience collection by spreading actors across many workers. Only IMPALA corrects for the resulting lag between actor and learner policies, with V-trace importance weights; A3C and Ape-X apply no off-policy correction. These frameworks parallelize a single policy and do not enforce specialization across subregions of the target. Population-based training (Jaderberg et al., 2017) maintains a population of agents with diverse hyperparameters, periodically replacing under-performers; diversity-aware variants additionally regularize for behavioral spread (Parker-Holder et al., 2020). Specialization there emerges only implicitly from hyperparameter perturbations or behavioral diversity bonuses, not from an explicit partition of the target distribution, and connects to a broader literature linking GFlowNet training to maximum-entropy reinforcement learning (Tiapkin et al., 2024a; Deleu et al., 2024). Mixture-of-experts architectures (Jacobs et al., 1991; Shazeer et al., 2017; Obando-Ceron et al., 2024) are closer in spirit: a learned gate routes inputs to specialized sub-networks. Our classifier-driven partition can be viewed as a mixture of experts in which the experts are full generative policies rather than feed-forward heads, and the gate's role is to define each expert's *target distribution* rather than to combine their outputs.

Parallel and distributed GFlowNets. Several recent methods parallelize GFlowNet training. Only one of them also targets mode coverage. Trajectory Balance with Asynchrony (Bartoldson et al., 2025) decouples exploration from learning by spreading off-policy searchers across nodes that populate a shared replay buffer, yielding large speedups for language-model post-training; the searchers, however, are snapshots of a single policy, so the underlying reachable set is shared and the population is asymptotically homogeneous. Embarrassingly Parallel GFlowNets (Da Silva et al., 2024) and federated contrastive variants (Silva et al., 2024) train one local GFlowNet per factor of the *reward* (or per data shard in federated Bayes) and aggregate them via balance conditions that recover a global target, so that reward evaluation can be spread across machines. Subgraph Asynchronous Learning (SAL, Silva et al., 2025) comes closest to our setting. It splits the *state graph* into subgraphs, trains one GFlowNet per subgraph in parallel, and then combines them. The authors report up to $8\times$ more modes than a single GFlowNet. In SAL the split is chosen before training and the local GFlowNets train in isolation; in MoG the partition is learned jointly with the policies, and in MoGX they also exchange trajectories. Compositional Sculpting (Garipov et al., 2023) composes pre-trained generative processes via classifier-guided routing, but operates post-hoc on fixed models rather than co-training a community of policies. *No existing method scales GFlowNet training by partitioning the state space across a community of policies that are trained jointly under a fixed environment-interaction budget*, precisely the regime that matters for AI for Science, where the dominant per-sample cost is an expensive environment query rather than gradient compute.

B Shared Distributed Infrastructure

Every method in this paper (NoComm, DDP, EGFN, MoG, MoGX) runs on the same distributed stack. Isolating this stack as a single shared layer serves two purposes: it lets us hold the environment-interaction budget fixed across methods and community sizes, so that performance differences reflect the method rather than uneven sampling; and it provides a single place to define the global mode registry that produces our headline metric.

Rank topology. A run with community size K uses K *training ranks*, B *buffer ranks* (default B=1), and an optional *coordinator rank*. Training ranks own a GFlowNet replica θ_k plus a thin local replay buffer of capacity 2048; buffer ranks own a remote replay buffer plus per-rank mode-discovery state; the coordinator, when present, maintains the union mode set across the population. Training and coordinator ranks share an MPI sub-communicator used for all collectives (all-reduce of f_ψ gradients, all-to-all of exchanged trajectories). Buffer ranks are excluded from the sub-communicator so they never participate in training-time synchronization.

Replay and mode registry. Each training rank composes the loss-time batch as a fixed mixture of fresh on-policy trajectories and replay samples. After loss evaluation, the batch is shipped to the assigned buffer rank for centralized scoring; the buffer rank invokes the environment's mode-detection routine on every terminating state, hashes any newly discovered modes, and updates a per-rank counter $n_k^{(t)}$. The coordinator unions these counters and exposes the global metric `n_modes_found` reported in section 4. For MoG and MoGX, the mode registry is read-only; the same infrastructure supports discovery-rate-driven replacement variants of the framework that are out of scope for this paper. Section G repeats the on-policy sweep with SubTB and DB.

C Method Implementation Details

Loss-agnostic instantiation. The framework is orthogonal to the choice of GFlowNet objective. We instantiate with trajectory balance (TB) (Malkin et al., 2022) throughout, but the same code path applies to detailed balance, sub-trajectory balance (Madan et al., 2023), log-partition variance, and modified DB. The only requirement is that the loss be off-policy-friendly when consumed through a replay buffer, a property shared by all four objectives since each is computed from (s, a, s') tuples or trajectories without an importance-weighted likelihood correction tied to the sampling policy.

Classifier coherence. f_ψ consumes the same K-hot preprocessed state $\phi(x) \in \mathbb{R}^{\text{ndim}\cdot\text{height}}$ used by the policies. Its parameters are broadcast from rank 0 at construction time, so every replica starts from identical weights; combined with the gradient all-reduce, this gives true data-parallel training of f_ψ regardless of how the policies are synchronized. Because every rank starts from the same $\psi^{(0)}$ and applies the same averaged gradient, all replicas of f_ψ remain identical throughout training, with no broadcast beyond initialization. This is essential: the shaped reward in eq. (1) would be incoherent across ranks if their classifiers drifted apart.

Reward-shaping safety net. We clip the shaping term in eq. (1) to $\text{clip}\big(\log f_\psi(x)_k,\, -20,\, 0\big)$, a numerical safety net: deep into training, $\log f_\psi(x)_k$ for an out-of-subregion state can otherwise saturate the loss for that sample. Shaping is applied *after* the replay-buffer sampler, so the multiplier $f_\psi(x)_k$ always reflects the current classifier weights even when x was buffered many iterations ago, which is what makes joint training stable, and it is computed inside a `torch.no_grad()` block so policy gradients never propagate back into the classifier.

Self-consistency. The all-reduced cross-entropy in eq. (3) is equivalent to supervised classification on the union dataset $\bigcup_k \{(x, k) : x \in \mathcal{B}^{(k)}\}$. The Bayes-optimal classifier under that distribution is

$$f_\psi^\star(x)_k \;=\; \Pr(\text{component}{=}k \mid x) \;=\; \frac{\pi_{\theta_k}(x)\,\mathcal{P}_\Pi(\pi_k)}{\sum_j \pi_{\theta_j}(x)\,\mathcal{P}_\Pi(\pi_j)}.$$

As each π_{θ_k} specializes onto a subregion of $\mathcal{X}$, the classifier sharpens its routing of states to whichever component covers them most, and eq. (1) re-amplifies that specialization on the next iteration. The two halves close the loop, and eq. (2) is the fixed point.

Symmetry breaking and initialization. At iteration 0, f_ψ outputs a near-uniform softmax, so $\log f_\psi(x)_k \approx -\log K$ pointwise and eq. (1) reduces to a global constant offset on every component's log-reward; this constant is absorbed into $\log Z_k$ within a handful of gradient steps and the policy distribution $\pi_{\theta_k}(x) \propto \tilde{R}_k(x)$ is unaffected. Which subregion ends up assigned to which component is supplied by the independence of on-policy sampling early in training. Empirically the partition stabilizes within the first few thousand iterations and stays put.

Exchange transport. In MoGX, selected trajectories are grouped by destination rank, serialized into a CPU-tensor payload (states, actions, terminating indices, log-rewards, log-probabilities, estimator outputs), and exchanged via a single `MPI_Alltoall` on the training sub-communicator. Every rank participates on every iteration; empty send slots are no-ops. Received payloads are concatenated with the locally retained slice of the source batch, and enter the replay buffer as ordinary samples. The exchange step sits between on-policy sampling and the optimization step, so the receiving rank attributes any mode discoveries to itself and the reward shaping of eq. (1) applies uniformly to local and received trajectories.

D Implementation Details and Hyperparameters

This appendix consolidates the architecture, optimizer, and training-schedule settings shared across all four population configurations (NoComm, DDP, MoG, MoGX). All methods use an identical per-agent architecture and run each agent for the same number of trajectories. Section 4 compares them at a shared *total* budget, so agents in larger communities take fewer gradient steps. The methods differ only in their inter-agent coordination, as described in section 3. table 2 lists the full configuration. All experiments are implemented on top of the `torchgfn` library (Viviano et al., 2023).

Hardware. All runs are executed on the CPU partition of the Trillium cluster (SciNet, University of Toronto; part of the Digital Research Alliance of Canada). Each CPU node comprises two 96-core AMD EPYC "Zen5" (Turin) processors for 192 cores and 768 GB of RAM, connected by an NDR InfiniBand (Mellanox Dragonfly+) fabric. We map one MPI rank to one physical CPU core, so a community of size K uses K cores for the training ranks plus one core for the buffer rank (section C). Every configuration in the sweep, up to the largest (K=64, i.e. 65 ranks), therefore fits within a single node, and all cross-rank communication reported in section H (gradient/classifier all-reduce and the MoGX trajectory all-to-all) is intra-node shared-memory MPI rather than inter-node network traffic.

D.1 HyperGrid environments

All environments instantiate the d-dimensional HyperGrid of Bengio et al. (2021): a grid of height H whose states are vectors $s \in \{0, \dots, H-1\}^d$, with all trajectories starting at $\mathbf{0}_d$ and permitted to terminate in any state, giving $|\mathcal{X}| = H^d$ terminal states. Where convenient we write the per-dimension normalized coordinate $a_i(s) = \left| s_i/(H-1) - \frac{1}{2} \right| \in [0, \frac{1}{2}]$. Unless noted, we use $d = 10$. The three compositional rewards share a *cumulative tiered* form

$$R(s) = R_0 + \sum_t w_t \, \mathbf{1}[s \text{ satisfies all constraints of tiers } 0, \dots, t], \tag{6}$$

with strictly increasing weights w_t, so that a state attains the reward of tier t only if it also satisfies every lower tier. Because the tier indicators are cumulative, summing equation 6 over $\mathcal{X}$ gives a closed-form partition function

$$Z \;=\; \sum_s R(s) \;=\; R_0 \, H^d \;+\; \sum_t w_t \, m_t, \qquad m_t := \big|\{s : s \text{ passes tiers } 0..t\}\big|, \tag{7}$$

where m_t is the tier-t mode count given in closed form for each environment below. For $H = 16$, $d = 10$ we have $H^d \approx 1.10 \times 10^{12}$, and the $R_0 H^d$ term dominates Z (e.g. R_0=0.1 gives $R_0 H^d \approx 1.10 \times 10^{11}$); since the highest tier occupies a vanishing fraction of $\mathcal{X}$, on-policy sampling almost never lands on a top-tier mode, which is precisely the mode-bias regime of section 2.

Original (Bengio et al., 2021).

$$R(s) = R_0 + R_1 \prod_{i=1}^{d} \mathbf{1}[a_i(s) > 0.25] + R_2 \prod_{i=1}^{d} \mathbf{1}[0.3 < a_i(s) < 0.4]. \tag{8}$$

We use R_0=0.01, R_1=0.5, R_2=2.0 at H=10. Modes are the states lying in the inner band $\{0.3 < a_i < 0.4\}$ in *every* dimension simultaneously (reward $R_0{+}R_1{+}R_2$); with b admissible band indices per axis the mode set has size b^d (e.g. b=2 at H=10, i.e. 2^{10} modes concentrated near the 2^d corners).

Component / hyperparameter	Value
Policy network	
Architecture	MLP, 2 hidden layers, width 256
Forward/backward policy	shared parameters (tied P_F/P_B)
GFlowNet loss	trajectory balance (TB)
Sampling temperature / ϵ	T=1, ϵ=0 (on-policy)
Optimization	
Optimizer	AdamW
Learning rate	10^{-3}
Batch size	1024 trajectories
LR warmup	linear over first 500 iterations
Gradient clipping	$\|\nabla_\theta \mathcal{L}\|_2 \leq 10$
Per-agent budget	1.31×10^8 trajectories
Partition classifier f_ψ (MoG, MoGX)	
Architecture	MLP, 2 hidden layers, K logits
Input	K-hot preprocessed state $\phi(x)$
Initialization	broadcast from rank 0 (identical)
Gradient synchronization	all-reduce (sum-then-divide), AdamW step
Entropy weight β	0 (default)
Shaping clip	$\log f_\psi(x)_k$ clipped to $[-20, 0]$
Replay and infrastructure	
Local replay buffer capacity	2048
Remote replay buffer capacity	8192
Buffer ranks B	1 (default)
Replay mix ratio ρ	0.5 (replay) / 0 (pure on-policy)
Trajectory exchange (MoGX only)	
Exchange count N	128 of 1024 (12.5%)
Routing rule	$\arg\max_j \log f_\psi(x)_j$
EGFN only (our adaptation, see section 4)	
Learner / population	rank 0 / ranks $1, \ldots, K-1$
Generations	one per gradient step
Elite fraction / tournament size	0.2 / 4
Crossover probability	0.1
Mutation	$\mathcal{N}(0, 1)$ per tensor, prob. 0.9
Prioritized replay buffer	top 1000 by reward
Replay sampling	50% from top 10%, 50% from rest
Trajectories sent per member	top 64 of 1024
Learner batch	1024 on-policy + 1024 replay
Sweep	
Environment	HyperGrid, ndim=10
Community sizes K	$\{1, 2, 4, 8, 16, 32, 64\}$

Table 2: Shared per-agent hyperparameters across all four population configurations. The four methods differ only in inter-agent coordination (section 3); all other settings are identical, which is what makes the iso-budget comparisons in section 4 meaningful. Reward-specific grid heights H are given in section 4. The EGFN rows apply to that baseline only.

Cosine. With $f(a) = \big(\cos(50a) + 1\big) \cdot \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(5a)^2}$,

$$R(s) = R_0 + R_1 \prod_{i=1}^{d} f\big(a_i(s)\big), \tag{9}$$

with R_0=0.1, R_1=0.5 at H=16. The oscillatory factor produces a regular lattice of high-reward regions; a state is counted as a mode when the product exceeds a γ-scaled per-dimension peak, $R \geq R_0 + (\gamma\, f_{\max})^d R_1$ (we use γ=0.8).

Sparse (Pan et al., 2023). Let the target set $\mathcal{T}$ contain, for each $k \in \{0, \ldots, d-1\}$, every distinct permutation of the vector with k entries equal to 1 and $d-k$ entries equal to $H-2$. The reward places an isolated spike on each target,

$$R(s) = \varepsilon + \sum_{t \in \mathcal{T}} \mathbf{1}[s = t], \qquad \varepsilon > 0. \tag{10}$$

There are $|\mathcal{T}| = \sum_{k=0}^{d-1} \binom{d}{k} = 2^d - 1$ such spikes (1023 at d=10; the all-ones vector is excluded), each surrounded by reward ε, making the modes extremely small and maximally isolated.

Bitwise XOR (compositional; rule reuse). Let $\mathrm{bit}_b(\cdot)$ denote the b-th bit and restrict attention to a subset $D \subseteq \{1, \ldots, d\}$ of $|D| = m$ constrained dimensions. For tier t with bit window $[\ell_t, h_t]$, the constraint requires, at *every* bit-plane $b \in [\ell_t, h_t]$, a system of $\mathrm{GF}(2)$ (XOR) parity checks $A_t\, u^{(b)} \equiv c_t \pmod 2$ on the bit vector $u^{(b)} \in \{0,1\}^m$ across constrained dimensions (defaulting to even parity $\sum_{i \in D} \mathrm{bit}_b(s_i) \equiv 0$ when unspecified). The reward is equation 6 with these cumulative bit-plane constraints. Modes are therefore *non-local*: they are placed by an algebraic law rather than by proximity, and carry no spatial gradient. For a single even-parity check across m dimensions at all bit-planes, one coordinate is fully determined by the XOR of the others, and (since H is a power of two the determined value is always in range) the mode count is $H^{m-1} \cdot H^{d-m} = H^{d-1}$; e.g. m=d=10, H=16 gives $16^9 \approx 6.87 \times 10^{10}$ modes at the strictest tier. The intended challenge is *rule reuse*: the same parity computation, learned once, is reused at each tier under increasing strictness, so that identifying the rule from a handful of modes predicts the location of all the others.

Multiplicative Coprime (compositional; knowledge composition). Coordinates are internally shifted by $+1$ (state value $0 \mapsto 1$) so the origin is valid and short trajectories immediately encounter small prime-factorable numbers, providing early signal. Over a small prime set P and active dimensions $A \subseteq \{1, \ldots, d\}$, successive tiers impose *qualitatively different* constraints:

- **tier** 0 **(prime support):** each s_i, $i \in A$, factors entirely over P;
- **tier** ≥ 1 **(exponent caps):** every prime exponent is bounded, $e_{i,p} \leq c_t$, with c_t tightening per tier;
- **coprimality:** designated pairs (i, j) become coprime, $\gcd(s_i, s_j) = 1$;
- **LCM targets:** an optional global $\mathrm{lcm}_{i \in A} s_i$ target.

The reward is equation 6 with these cumulative constraints. Ignoring the cross-dimensional coprimality/LCM couplings, the prime-support-plus-caps mode count factorizes as $N^{|A|} \cdot H^{d-|A|}$, where N is the number of admissible "smooth" values in $\{1, \ldots, H\}$; for $P = \{2, 3, 5\}$, $c = 2$, $H = 16$ we have $N = 10$ (namely $\{1, 2, 3, 4, 5, 6, 9, 10, 12, 15\}$), giving 10^{10} modes when all dimensions are active. Coprimality couples dimensions and further reduces this count (it no longer factorizes). The intended challenge is *knowledge composition*: each tier is a different kind of number-theoretic reasoning, and the factorization structure learnable near the origin is a prerequisite for the coprimality and LCM reasoning that gates the rarer high-tier modes.

Conditional Multi-Scale (compositional; conditional hierarchy). Decompose each coordinate in base B into $L = \log_B H$ digits, $s_i = \sum_{k=0}^{L-1} d_k(i)\, B^k$. Ordering tiers coarse-to-fine, tier t constrains digit $L{-}1{-}t$ (the $(t{+}1)$-th most significant) through a *shifted* filter whose offset depends on all coarser digits already constrained:

$$\big(d_{L-1-t}(i) + \sigma_t(i)\big) \bmod B < f, \qquad \sigma_t(i) = \sum_{k=0}^{t-1} a_{t,k}\, d_{L-1-k}(i) \bmod B, \tag{11}$$

with seed-derived coefficients $a_{t,k}$ (tier 0 has $\sigma_0{=}0$), and an optional cross-dimensional term $\sum_i d_{L-1-t}(i) \equiv 0 \pmod{m_t}$. The reward is equation 6. Because coarse-scale digits are small near the origin (high digits equal 0 and pass trivially), difficulty is distance-correlated. The mode count admits a closed form,

$$m_T = \left(f^T\right)^d B^{(L-T)d} \Big/ \prod_t m_t, \tag{12}$$

(with the $\prod_t m_t$ factor present only under cross-dimensional constraints), and Z follows from equation 7 with no enumeration. As a reference instantiation, $B{=}2$, $L{=}4$, $f{=}1$, $d{=}10$ yields $m_1 \approx 1.07 \times 10^9$, $m_2 \approx 1.05 \times 10^6$, and $m_3{=}1024$ as tiers accumulate. The intended challenge is a *conditional hierarchy*: the constraint at each scale is a function of the scales below it, so exploration must proceed coarse-to-fine, with the lower tiers acting as foothills that bootstrap discovery of the higher ones.

E Mixture-Correctness Validation: Setup and Full Results

Setup. We take a $K{=}4$ MoGX checkpoint on HyperGrid *original* ($\mathsf{ndim}{=}10$, $H{=}7$, $R_0{=}10^{-8}$, "challenging" preset; $2^{10}{=}1024$ corner modes; analytical reward-proportional optimum uniform across modes at 7.26×10^{-6} each with 99.26% off-mode mass). For each rank k we draw 10^6 on-policy trajectories and bin terminating states by mode index. We then build the Z-weighted mixture marginal $p_\Pi(x) = \sum_k w_k \, \pi_{\theta_k}(x)$ with $w_k = \mathrm{softmax}(\log Z_k)_k$, reading $\log Z_k$ directly from the trained TB scalar exactly as prescribed by eq. (2). The reference optimum is computed in closed form by enumerating only the 2^{10} mode-band states and using *original*'s outer-ring decomposition for the partition function Z (no H^{ndim} enumeration). KL divergences are reported over the full n_{modes}-category support.

Full results. Figure 4 reports the per-rank and mixture marginals. Each individual rank is far from the optimum on its own, by design: each π_{θ_k} targets its shaped reward $\tilde{R}_k = R \cdot f_\psi(\cdot)_k$, not R. All four ranks place mass on distinct, sparse subsets of the 1024 modes (off-mode mass 99.1–99.6%, forward KL 0.81–$1.11{\times}10^{-2}$). The Z-weighted mixture, in contrast, recovers the analytical optimum: $\mathrm{KL}(p_\Pi \,\|\, \pi^\star) = 1.51{\times}10^{-4}$, lower than every individual rank by a factor of 50–$75\times$, with off-mode mass 99.31% matching the analytic 99.26% to within sampling noise and on-mode mass distributed near-uniformly across the 1024 modes. No single rank covers the modes on its own; the mixture does, because the union of the four ranks' coverage, weighted by $\{Z_k\}$, reassembles into the full target. This is the operational content of MoG's mixture-correctness proposition.

F Full Training Curves

For completeness, we give the full per-trajectory mode-discovery curves underlying the iso-budget endpoints reported in section 4. figs. 6 to 11 cover the replay regime ($\rho{=}0.5$) and figs. 12 to 17 the pure on-policy regime ($\rho{=}0$), with one figure per reward. In general, the curves show that MoG and MoGX are more sample efficient than NoComm and DDP and continue benefiting from larger K. In each figure, columns subdivide by community size K and shaded bands show ± 1 s.d. across seeds. Figures 18 and 19 give the same view for the ChromaX ideotype environment of section 4.1, in the replay and pure on-policy regimes respectively.

G Other Training Objectives

We reran the pure on-policy sweep ($\rho{=}0$) with SubTB and DB on *original* (easy preset, $R_0{=}0.01$), *bitwise_xor* and *conditional_multiscale*. SubTB uses $\lambda{=}0.9$ with geometric weighting within each sub-trajectory, and for both losses the $\log F$ head shares the policy trunk. Everything else follows table 2. Endpoints are read at the largest budget every run with $K{\geq}2$ reached, over 4 seeds.

SubTB. SubTB's baselines barely suffer from a larger community. On *bitwise_xor*, NoComm finds 9.1×10^7 modes at $K{=}2$ and still 8.2×10^7 at $K{=}64$, where all four methods sit within 7% of each other (fig. 20). There is no collapse for MoG to prevent. The one clear gain is on *original* at intermediate K: at $K{=}16$ MoG finds 100 modes, DDP 34 and NoComm 17. So the TB gains of fig. 3 depend on how badly TB's baselines degrade with K.

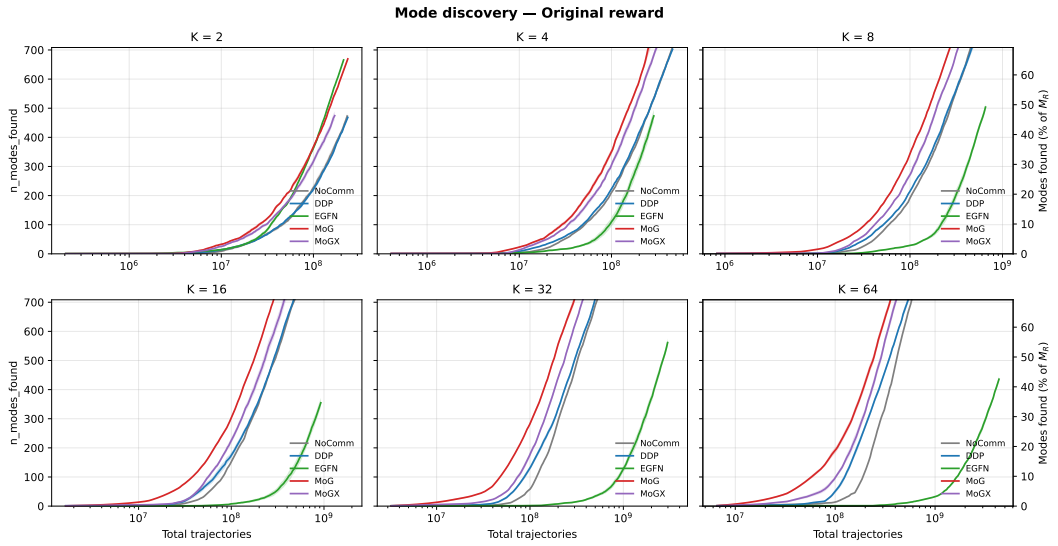


Figure 6: Modes discovered vs total trajectories on the *original* reward, **replay regime** (ρ=0.5). MoGX (purple) and MoG (red) discover modes faster than DDP (blue) and NoComm (grey).

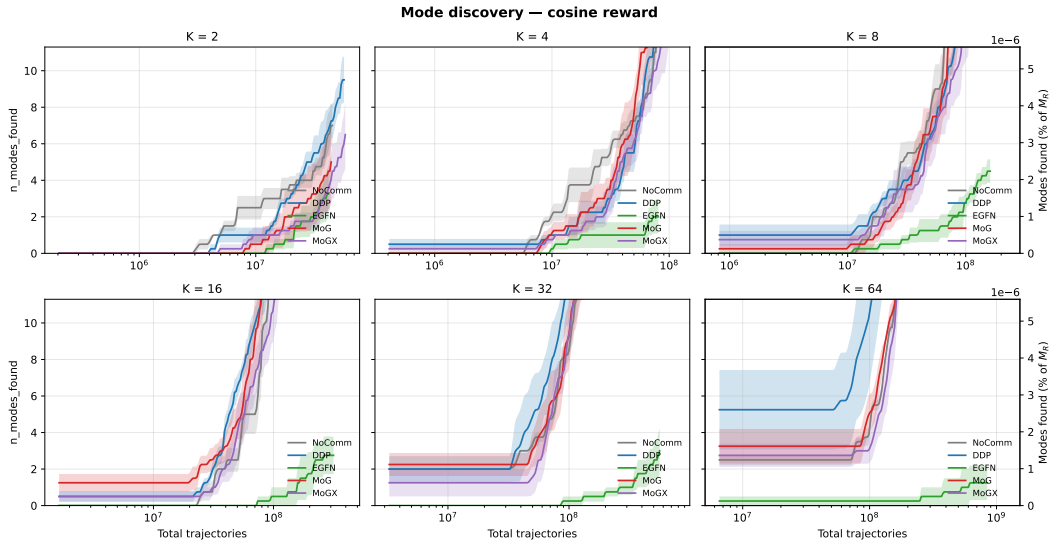


Figure 7: Modes discovered vs total trajectories on the *cosine* reward, **replay regime** (ρ=0.5).

DB. Under DB, MoG and MoGX carry the tilt as a flow potential. Rank k uses

$$\log F(s) \;=\; \log F_\theta(s) + u_k(s), \qquad u_k(s) \;=\; \max\big(\log f_\psi(s)_k,\, -5\big),$$

and shapes terminal rewards with the same u_k. The optimum is that of DB with boundary $\log \tilde{R}_k$, but every interior residual now carries $u_k(s) - u_k(s')$ from the first update, and $\log F_\theta$ only fits the untilted flow.

MoG tracks the baselines up to K=16 and pulls ahead on *bitwise_xor* beyond it, with 6.5×10^7 modes at K=64 against 6.2×10^7 for DDP and 6.0×10^7 for NoComm (fig. 21). On the other two rewards, it shows no consistent gain. MoGX is worse: it trails every method at every K on *bitwise_xor*, and at $K \leq 16$ on *original* it finds 17–51 modes where the others find 95–229.

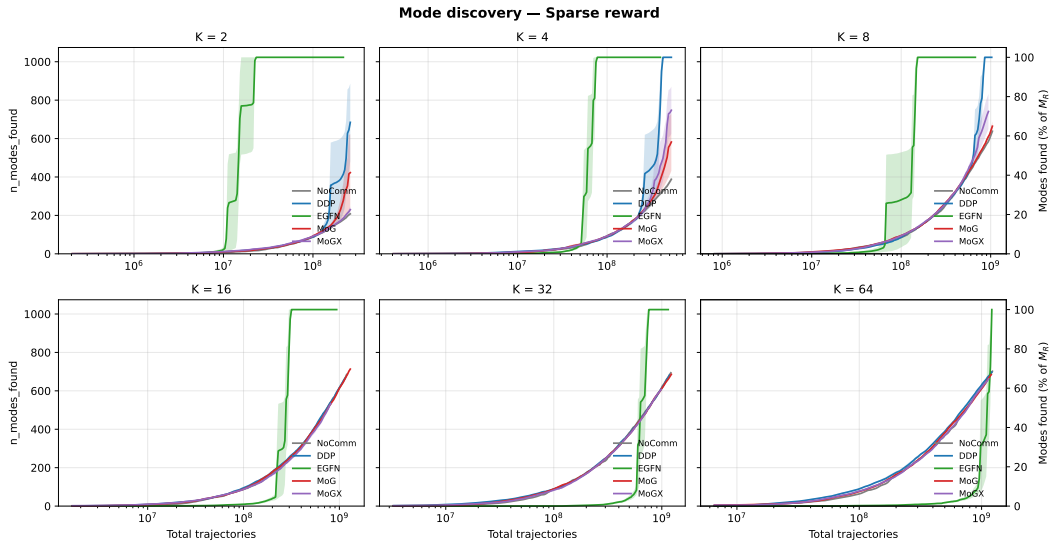


Figure 8: Modes discovered vs total trajectories on the *sparse* reward, **replay regime** (ρ=0.5).

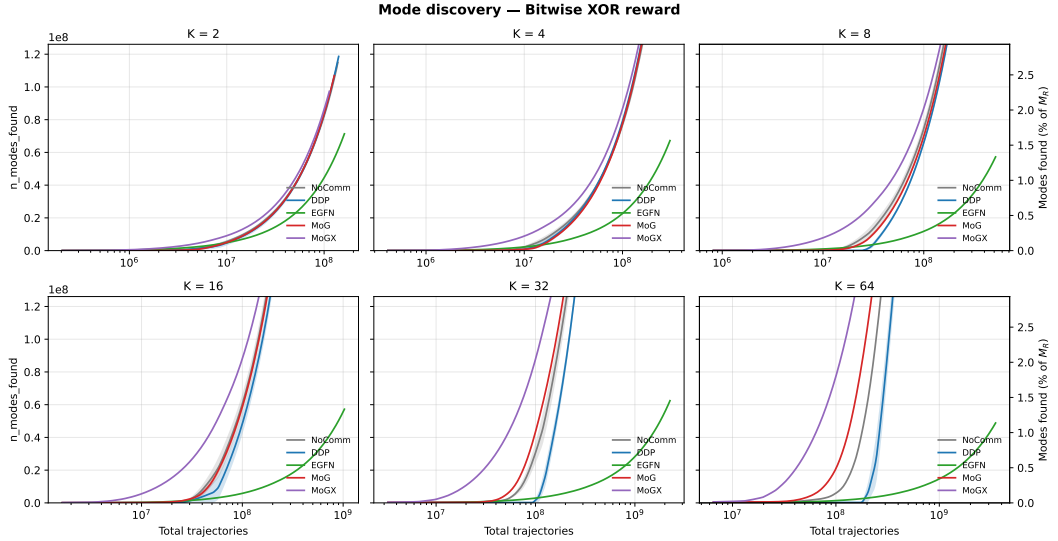


Figure 9: Modes discovered vs total trajectories on the *bitwise_xor* reward, **replay regime** (ρ=0.5). The MoGX–MoG gap widens with structural complexity.

H TIMINGS

Per-reward totals. Table 1 of the main paper reports the per-method per-iteration total for the *original* reward. Tables 3 to 8 give the same total for each reward separately, with table 3 reproducing the main-paper table. The per-method scaling pattern is identical for every reward: NoComm stays flat in K; DDP grows with the ∇_θ all-reduce; MoG grows much more slowly, paying only the much cheaper classifier all-reduce; and MoGX adds the trajectory all-to-all on top. The absolute level shifts with the per-reward shaping cost (*sparse* is the cheapest, *cosine* the most expensive). At K=64 the ordering NoComm $<$ MoG $<$ DDP $<$ MoGX holds on every reward except *sparse*, where MoG and NoComm are within noise. At small K the methods are within run-to-run noise of each other and the ordering varies. NoComm, whose per-rank work does not depend on K, varies by up to 23% across K on a single reward.

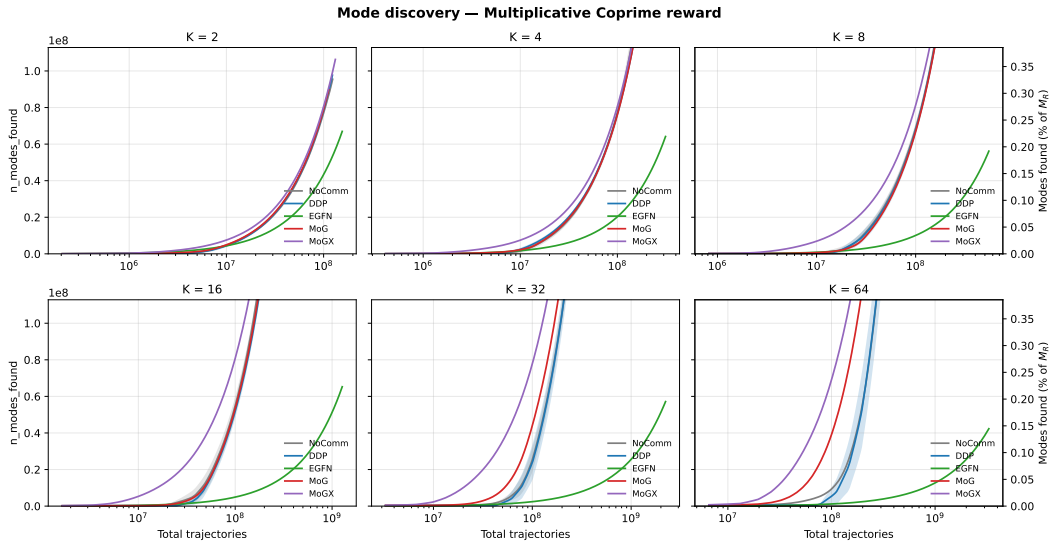


Figure 10: Modes discovered vs total trajectories on the *multiplicative_coprime* reward, **replay regime** (ρ=0.5). The MoGX–MoG gap widens with structural complexity.

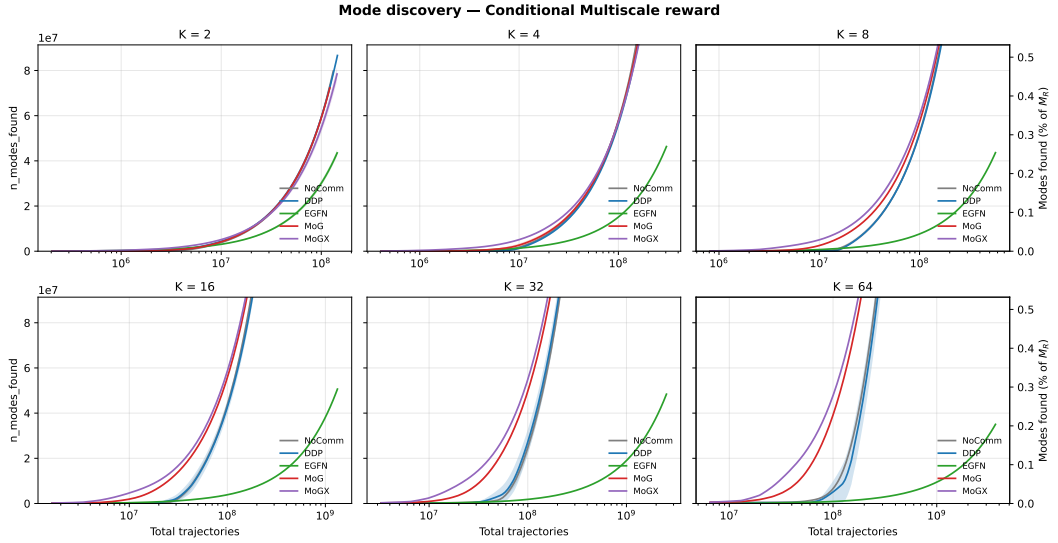


Figure 11: Modes discovered vs total trajectories on the *conditional_multiscale* reward, **replay regime** (ρ=0.5). The MoGX–MoG gap widens with structural complexity.

Per-reward phase breakdown. We also time the different training phases while scaling the community size. Tables 9 to 14 split the measurement by reward. *Sample* = on-policy rollouts; *Loss* = reward shaping + GFN loss forward; *Bwd* = backward pass. The rank-local phases are pooled over all methods. The cross-rank phases are averaged over the methods that use them: *Exch.*† = MoGX trajectory all-to-all; $\nabla_\theta^\ddagger$ = DDP policy-gradient all-reduce; $f_\psi^\star$ = classifier loss + blocking gradient all-reduce, averaged over MoG and MoGX. The two differ: on *original*, MoG's f_ψ phase takes 23 ms at K=2 and 284 ms at K=64, while MoGX's stays below 12 ms. The rank-local phases (Sample, Loss, Bwd) track the per-reward shaping cost: *sparse* is the cheapest, while *cosine* is the most expensive. The cross-rank phases (Exch., ∇_θ, f_ψ) follow the same qualitative scaling in K as in the aggregate, independent of the reward.

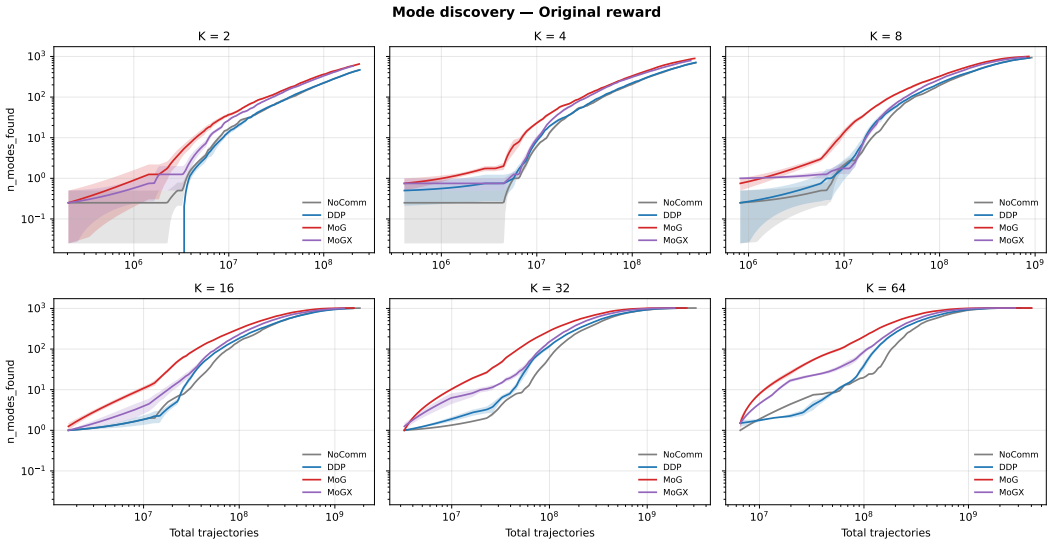


Figure 12: Modes discovered vs total trajectories on the *original* reward, **pure on-policy regime** (ρ=0, replay buffer record-only).

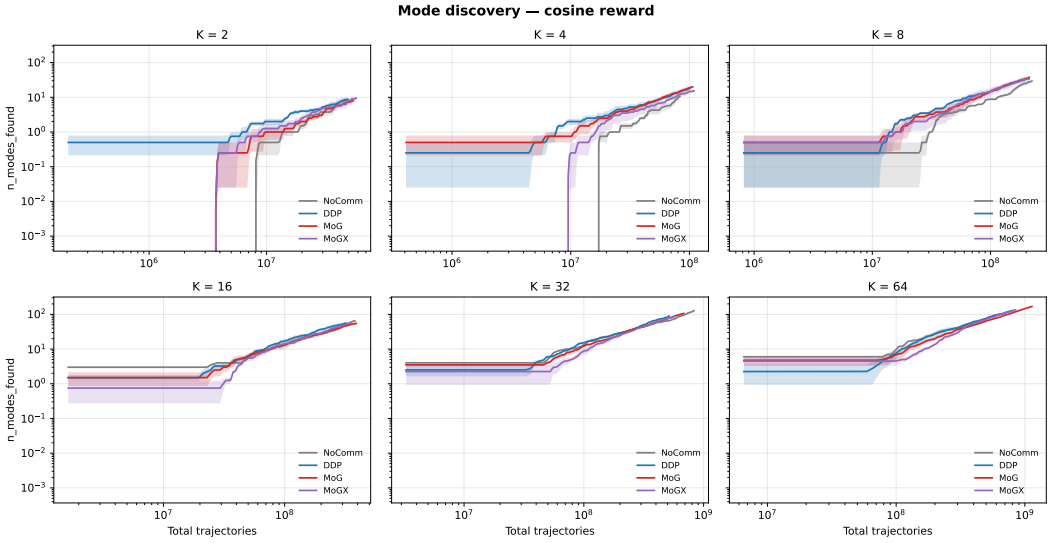


Figure 13: Modes discovered vs total trajectories on the *cosine* reward, **pure on-policy regime** (ρ=0, replay buffer record-only).

Method	K=1	2	4	8	16	32	64
NoComm	702 ±4	729 ±6	726 ±7	728 ±7	732 ±3	758 ±23	713 ±9
DDP	702 ±4	715 ±3	743 ±2	784 ±1	823 ±2	950 ±4	1249 ±11
EGFN	702 ±4	712 ±3	748 ±23	853 ±14	876 ±70	803 ±6	975 ±3
MoG	702 ±4	721 ±4	749 ±7	800 ±3	818 ±6	923 ±8	976 ±7
MoGX	702 ±4	762 ±30	804 ±15	841 ±12	998 ±12	1215 ±49	1477 ±12

Table 3: Same per-method per-iteration wall-clock total as table 1, reported also here for reference.

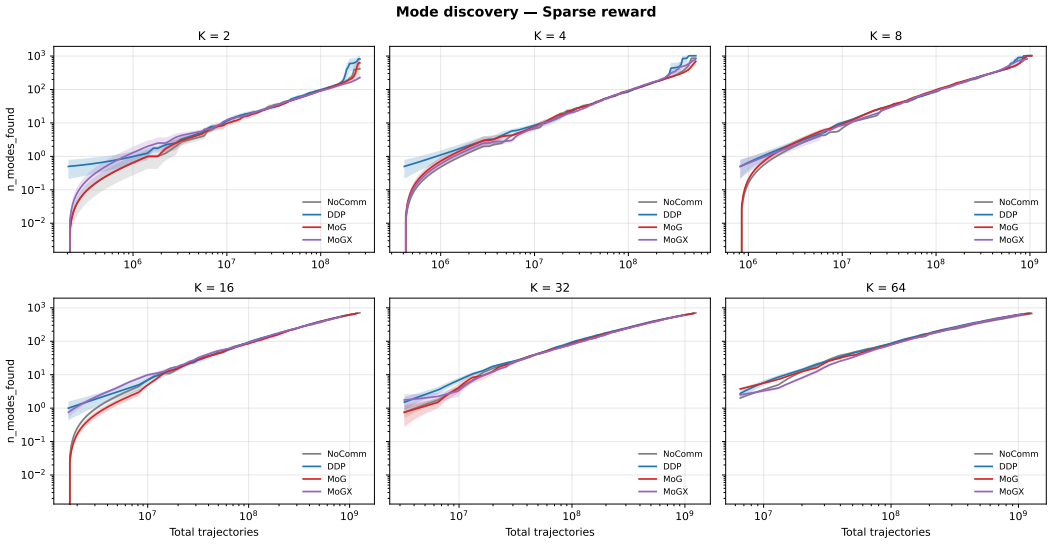


Figure 14: Modes discovered vs total trajectories on the *sparse* reward, **pure on-policy regime** (ρ=0, replay buffer record-only).

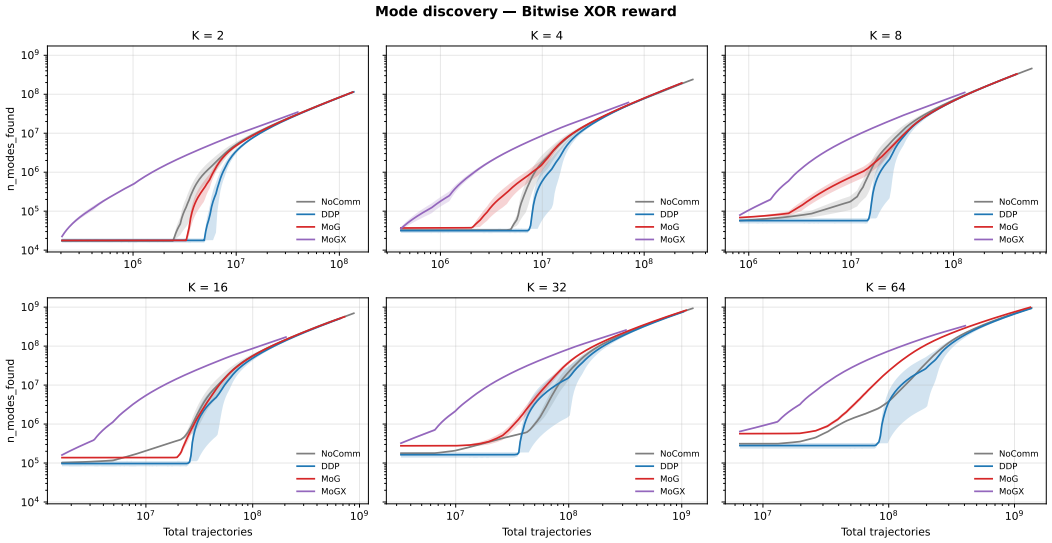


Figure 15: Modes discovered vs total trajectories on the *bitwise_xor* reward, **pure on-policy regime** (ρ=0, replay buffer record-only).

Method	K=1	2	4	8	16	32	64
NoComm	2936 ±45	3103 ±167	3278 ±278	2971 ±65	3303 ±122	2999 ±71	2969 ±73
DDP	2936 ±45	3106 ±142	3127 ±94	3181 ±123	3451 ±224	3988 ±204	4727 ±182
MoG	2936 ±45	2982 ±68	3099 ±52	3197 ±26	3274 ±37	3628 ±40	3994 ±31
MoGX	2936 ±45	2919 ±24	3092 ±35	3218 ±15	3643 ±64	4059 ±34	4840 ±17

Table 4: Per-method per-iteration wall-clock total, similar to table 1, but restricted to the *cosine* reward. Aggregation, units, and method definitions match table 1. At K=1 all four methods collapse to the same single-agent loop and share the pooled K=1 baseline.

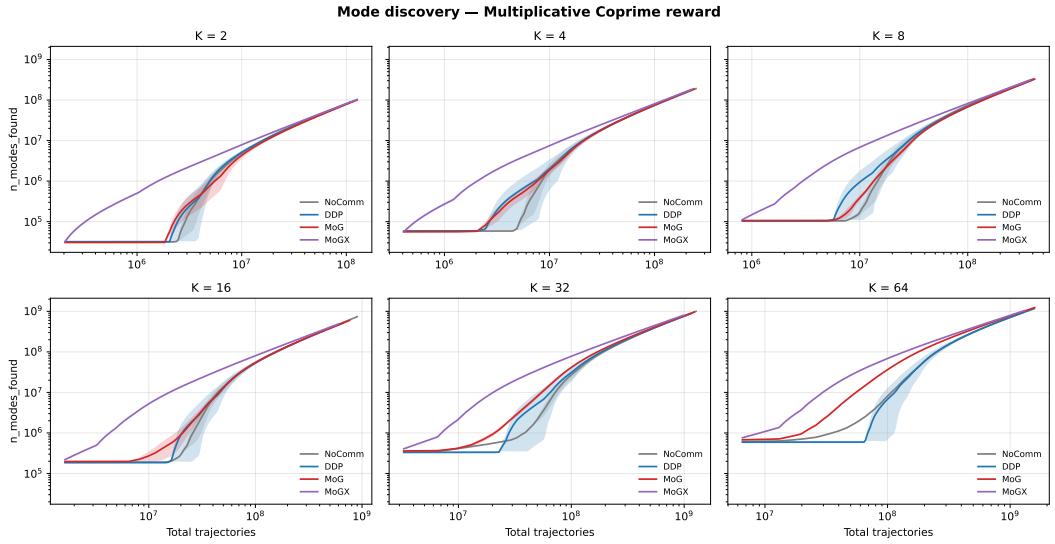


Figure 16: Modes discovered vs total trajectories on the *multiplicative_coprime* reward, **pure on-policy regime** (ρ=0, replay buffer record-only).

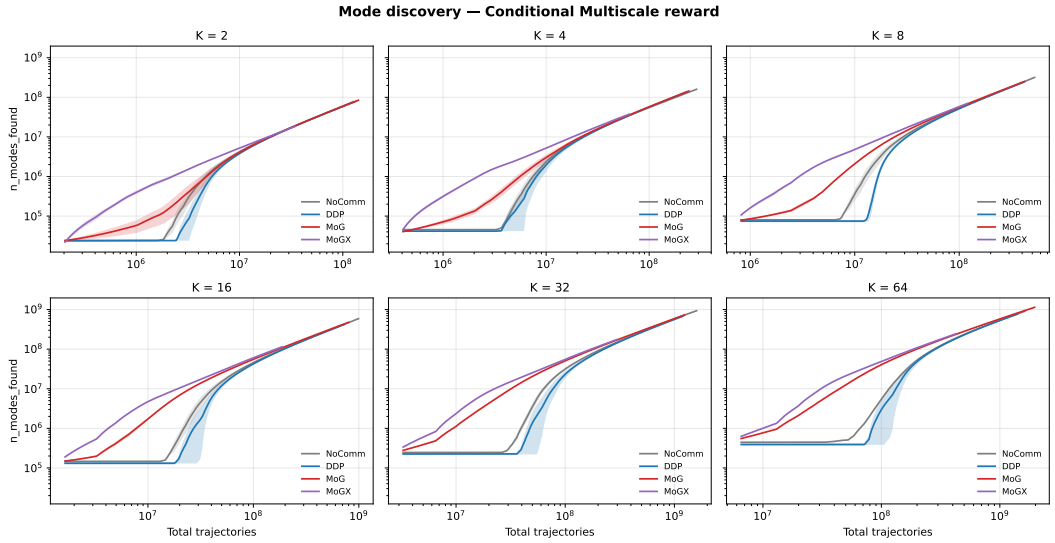


Figure 17: Modes discovered vs total trajectories on the *conditional_multiscale* reward, **pure on-policy regime** (ρ=0, replay buffer record-only).

Method	K=1	2	4	8	16	32	64
NoComm	494 ±3	498 ±2	479 ±5	479 ±5	464 ±3	474 ±5	483 ±4
DDP	494 ±3	529 ±3	560 ±3	606 ±3	905 ±21	1475 ±48	2780 ±120
MoG	494 ±3	501 ±9	522 ±6	558 ±3	526 ±9	488 ±7	476 ±13
MoGX	494 ±3	556 ±8	601 ±8	701 ±4	1519 ±25	2640 ±46	4535 ±53

Table 5: Per-method per-iteration wall-clock total, similar to table 1, but restricted to the *sparse* reward. Aggregation, units, and method definitions match table 1. At K=1 all four methods collapse to the same single-agent loop and share the pooled K=1 baseline.

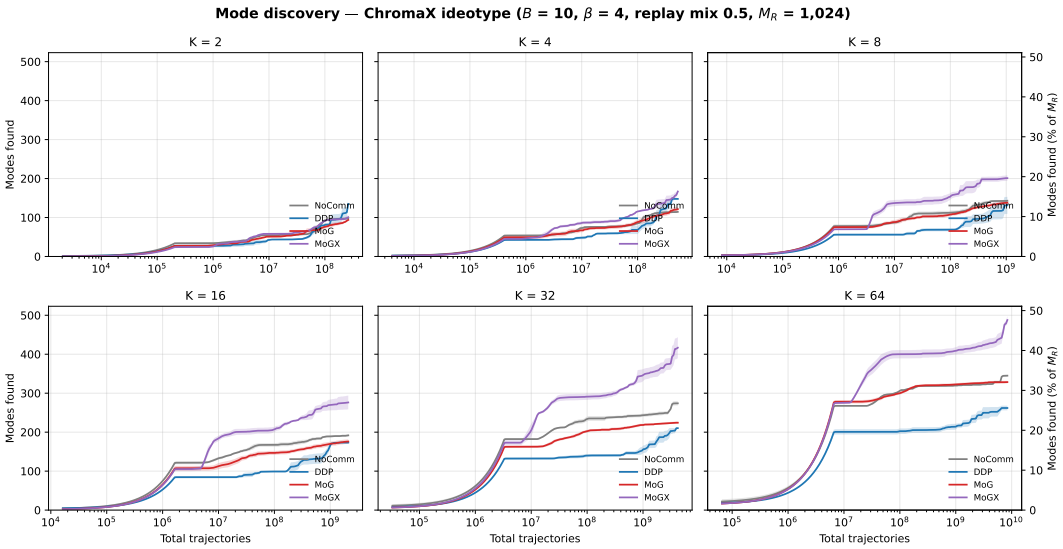


Figure 18: Modes discovered vs total trajectories on the ChromaX ideotype environment (section 4.1; B=10, β=4, M_R=1024 optima), **replay regime** (ρ=0.5), one panel per community size K, mean $\pm$ SEM over 4 seeds. MoGX (purple) pulls ahead of the other three methods from K=4 on, while MoG (red) and NoComm (grey) stay close together throughout.

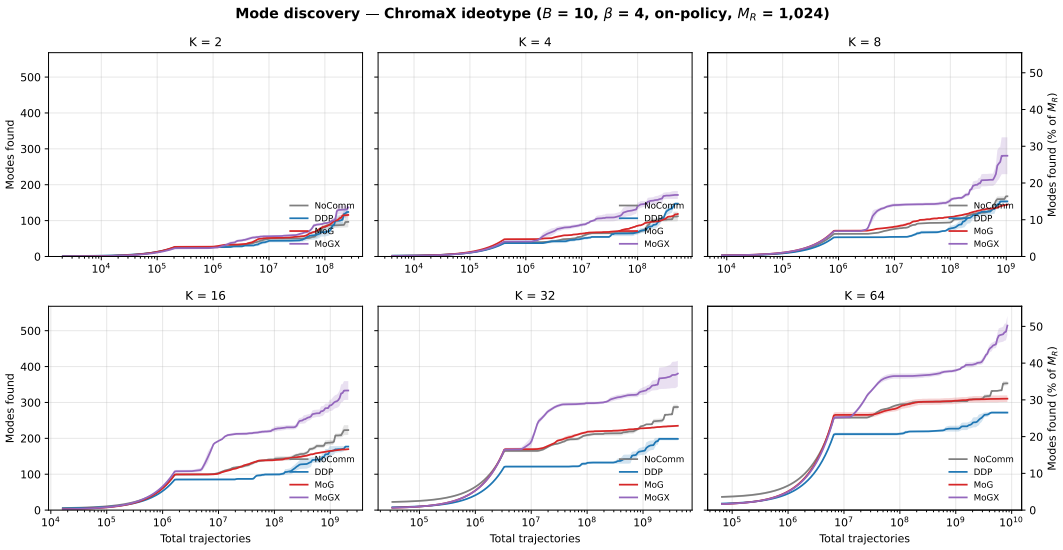


Figure 19: Modes discovered vs total trajectories on the ChromaX ideotype environment (section 4.1; B=10, β=4, M_R=1024 optima), **pure on-policy regime** (ρ=0, replay buffer record-only), one panel per community size K, mean $\pm$ SEM over 4 seeds. The picture is the one from fig. 18: MoGX (purple) separates from the other three methods from K=4 on, and MoG (red) stays on top of NoComm (grey).

Method	K=1	2	4	8	16	32	64
NoComm	1083 ±22	1151 ±48	1061 ±26	1066 ±22	1094 ±47	1067 ±9	937 ±23
DDP	1083 ±22	1206 ±19	1402 ±43	1636 ±25	1829 ±29	1904 ±41	2530 ±81
MoG	1083 ±22	1208 ±39	1363 ±40	1567 ±22	1543 ±15	1578 ±16	1470 ±44
MoGX	1083 ±22	1199 ±59	1388 ±89	1619 ±130	1901 ±147	2291 ±171	2923 ±215

Table 6: Per-method per-iteration wall-clock total, similar to table 1, but restricted to the *bitwise_xor* reward. Aggregation, units, and method definitions match table 1. At K=1 all four methods collapse to the same single-agent loop and share the pooled K=1 baseline.

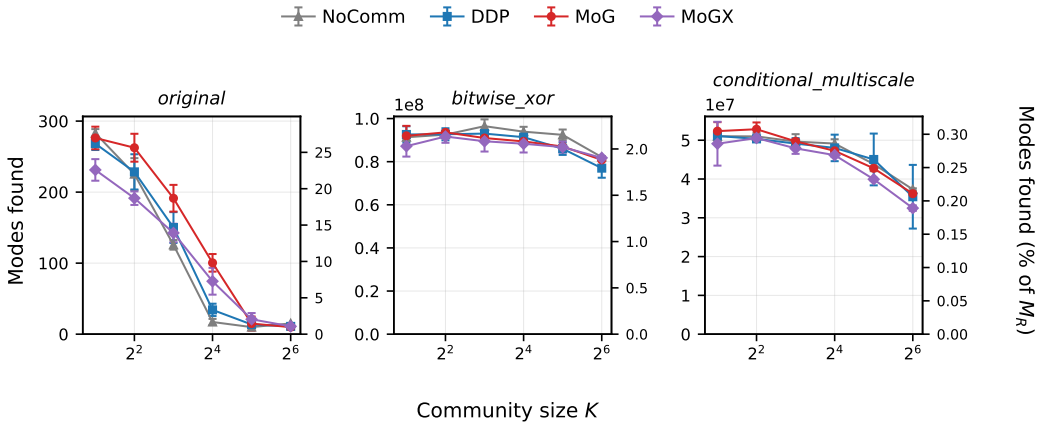


Figure 20: Modes discovered at fixed total-trajectory budget vs community size K, **SubTB**, pure on-policy regime, mean ± 1 s.d. over 4 seeds.

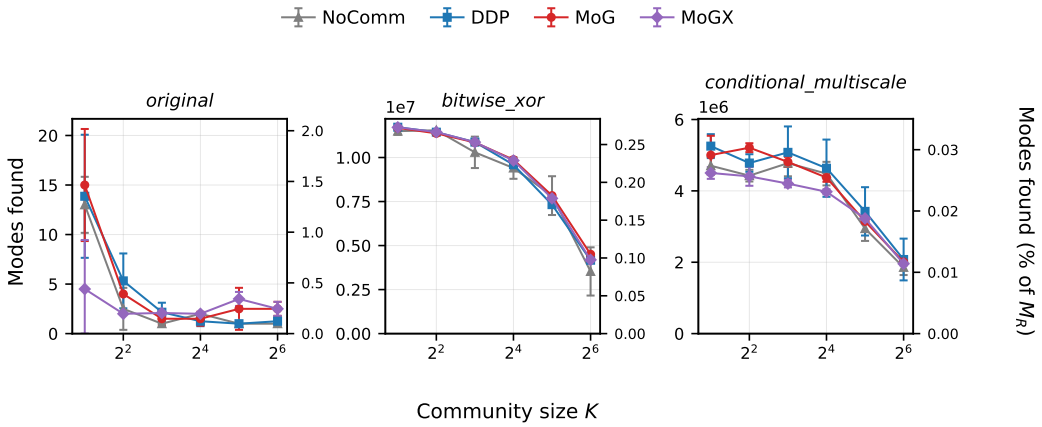


Figure 21: Same as fig. 20 for **DB**, with the tilt carried as a flow potential in MoG and MoGX.

Method	K=1	2	4	8	16	32	64
NoComm	1162 ±25	1320 ±98	1238 ±60	1328 ±106	1319 ±43	1216 ±63	1102 ±53
DDP	1162 ±25	1348 ±40	1495 ±28	1725 ±30	1906 ±33	1945 ±15	2315 ±40
MoG	1162 ±25	1244 ±41	1415 ±21	1598 ±27	1615 ±17	1789 ±30	1671 ±20
MoGX	1162 ±25	1325 ±17	1505 ±11	1774 ±20	2086 ±74	2300 ±26	2795 ±48

Table 7: Per-method per-iteration wall-clock total, similar to table 1, but restricted to the *multiplicative_coprime* reward. Aggregation, units, and method definitions match table 1. At K=1 all four methods collapse to the same single-agent loop and share the pooled K=1 baseline.

Method	K=1	2	4	8	16	32	64
NoComm	1114 ±24	1272 ±45	1110 ±26	1220 ±33	1174 ±39	1171 ±65	1087 ±45
DDP	1114 ±24	1241 ±31	1433 ±36	1597 ±30	1736 ±20	1758 ±33	2329 ±136
MoG	1114 ±24	1180 ±16	1323 ±37	1518 ±37	1580 ±12	1674 ±17	1672 ±38
MoGX	1114 ±24	1194 ±72	1364 ±99	1523 ±112	1841 ±165	2020 ±171	2697 ±265

Table 8: Per-method per-iteration wall-clock total, similar to table 1, but restricted to the *conditional_multiscale* reward. Aggregation, units, and method definitions match table 1. At K=1 all four methods collapse to the same single-agent loop and share the pooled K=1 baseline.

K	Sample	Loss	Bwd	Exch.†	$\nabla_\theta^{\ddagger}$	$f_\psi^\star$
1	246 ±1	224 ±1	232 ±2	–	–	–
2	245 ±2	224 ±2	229 ±3	84 ±45	24 ±3	14 ±4
4	245 ±3	227 ±3	233 ±4	92 ±44	55 ±2	28 ±14
8	248 ±4	230 ±4	235 ±4	101 ±29	90 ±3	54 ±22
16	244 ±3	233 ±4	238 ±4	263 ±42	129 ±4	60 ±22
32	244 ±3	232 ±6	238 ±7	514 ±102	256 ±7	108 ±37
64	245 ±2	226 ±3	233 ±3	760 ±38	549 ±11	145 ±52

Table 9: Per-iteration timing phase breakdown, restricted to the *original* reward.

K	Sample	Loss	Bwd	Exch.†	$\nabla_\theta^{\ddagger}$	$f_\psi^\star$
1	914 ±7	1051 ±26	971 ±20	–	–	–
2	915 ±11	1052 ±27	971 ±22	88 ±56	231 ±153	21 ±9
4	912 ±14	1087 ±50	993 ±36	312 ±106	210 ±56	53 ±22
8	909 ±12	1042 ±17	961 ±15	362 ±45	245 ±42	155 ±71
16	901 ±15	1095 ±39	995 ±32	773 ±217	464 ±129	233 ±98
32	895 ±13	1056 ±26	965 ±20	1220 ±60	983 ±64	403 ±161
64	913 ±16	1038 ±24	955 ±22	1981 ±122	1668 ±50	627 ±239

Table 10: Per-iteration timing phase breakdown, restricted to the *cosine* reward.

K	Sample	Loss	Bwd	Exch.†	$\nabla_\theta^{\ddagger}$	$f_\psi^\star$
1	170 ±1	167 ±1	157 ±1	–	–	–
2	172 ±2	172 ±2	161 ±2	22 ±9	21 ±3	9 ±1
4	167 ±1	170 ±2	158 ±2	79 ±26	61 ±8	23 ±7
8	167 ±2	169 ±3	157 ±3	188 ±34	103 ±7	40 ±13
16	163 ±2	165 ±3	153 ±3	1032 ±50	416 ±25	20 ±6
32	162 ±2	164 ±2	152 ±2	2164 ±33	986 ±53	6 ±0
64	166 ±3	168 ±3	156 ±3	4017 ±77	2280 ±116	10 ±2

Table 11: Per-iteration timing phase breakdown, restricted to the *sparse* reward.

K	Sample	Loss	Bwd	Exch.†	$\nabla_\theta^{\ddagger}$	$f_\psi^\star$
1	435 ±7	325 ±10	323 ±10	–	–	–
2	423 ±7	331 ±11	329 ±11	177 ±32	142 ±8	54 ±19
4	410 ±8	312 ±12	308 ±12	414 ±58	380 ±39	149 ±59
8	394 ±9	318 ±11	313 ±11	563 ±109	608 ±31	307 ±113
16	373 ±9	307 ±8	304 ±9	1006 ±134	832 ±19	295 ±109
32	346 ±11	312 ±12	308 ±11	1455 ±131	964 ±36	279 ±104
64	323 ±7	300 ±11	296 ±11	2013 ±125	1624 ±75	272 ±106

Table 12: Per-iteration timing phase breakdown, restricted to the *bitwise_xor* reward.

K	Sample	Loss	Bwd	Exch.†	$\nabla_\theta^{\ddagger}$	$f_\psi^\star$
1	463 ±5	351 ±14	349 ±14	–	–	–
2	457 ±5	366 ±17	363 ±17	165 ±5	213 ±54	58 ±23
4	433 ±9	354 ±12	351 ±13	402 ±45	380 ±40	158 ±59
8	422 ±9	365 ±18	361 ±19	706 ±23	606 ±31	260 ±95
16	402 ±8	364 ±16	361 ±17	1031 ±97	829 ±39	279 ±106
32	380 ±6	351 ±16	348 ±16	1270 ±39	816 ±49	424 ±156
64	363 ±6	352 ±14	350 ±14	1793 ±52	1291 ±24	271 ±101

Table 13: Per-iteration timing phase breakdown, restricted to the *multiplicative_coprime* reward.

K	**Sample**	**Loss**	**Bwd**	**Exch.**†	$\nabla_\theta^{\ddagger}$	$f_\psi^\star$
1	413 $\pm$7	351 $\pm$12	350 $\pm$12	–	–	–
2	416 $\pm$10	363 $\pm$15	361 $\pm$16	143 $\pm$25	123 $\pm$17	31 $\pm$13
4	404 $\pm$10	346 $\pm$14	343 $\pm$14	352 $\pm$95	306 $\pm$29	102 $\pm$47
8	386 $\pm$8	358 $\pm$11	354 $\pm$12	487 $\pm$78	532 $\pm$12	223 $\pm$83
16	379 $\pm$11	353 $\pm$10	350 $\pm$11	788 $\pm$133	697 $\pm$25	260 $\pm$100
32	362 $\pm$6	364 $\pm$14	360 $\pm$14	967 $\pm$144	710 $\pm$75	303 $\pm$115
64	347 $\pm$8	348 $\pm$9	344 $\pm$9	1687 $\pm$244	1323 $\pm$113	311 $\pm$119

Table 14: Per-iteration timing phase breakdown, restricted to the *conditional_multiscale* reward.