GFLOWNETS FOR STRATEGIC PLANNING IN HIERARCHICAL AND CYCLIC ENVIRONMENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Generative Flow Networks (GFN) sample terminal states in proportion to reward and have shown promising results in molecular and biological sequence design. Their capacity to explore diverse high-reward solutions may extend naturally to strategic decision-making in tasks with hierarchies, prerequisite chains, and alternative routes, yet they have been underexplored due to departure from standard formulations such as cyclicity in transitions. We designed and evaluated a sub-trajectory-balance (SubTB) GFN with replay on HierarchyCraft (Fédérico et al., 2026), which is a discrete hierarchical environment whose cyclic state space and recursive tool dependencies allow the freedom of diverse valid solution routes. We handle the cyclic state space using the theory of Morozov et al. (2025) and tested it empirically with the SubTB GFN. Compared against an entropy-regularized SoftDQN agent, which shares the optimal policy of the GFN, the GFN attains higher success rates on four of the five deep tasks and more strategic modes on most of them. At similar success rate, the GFN finds about two to four times as many strategic modes as SoftDQN and PPO. We further note that the gap persists across nearly all setups we tried for SoftDQN, suggesting that objectives with a shared optimum can learn at different rates and that flow-based objectives may retain an advantage, especially in long-horizon and sparse-signal tasks.

1 INTRODUCTION

Generative Flow Networks (GFNs) learn to sample terminal outcomes in proportion to their rewards through a sequence of actions (Bengio et al., 2021). This emphasis on covering rewarding outcomes makes them a promising candidate for strategic planning, where multiple useful plans may achieve the same goal through different resources or prerequisite chains. Access to these alternatives can offer flexibility as requirements change or unexpected circumstances arise, motivating agents that combine reliable goal completion with a variety of strategies.

The standard GFN formulation uses a directed acyclic graph in which actions progressively construct an object (Bengio et al., 2023). While this structure naturally describes incremental construction, real-world planning environments often contain reversible actions and repeated visits to previous states. Accommodating cycles allows GFNs to operate within these existing dynamics and may broaden their practical use.

Additionally, hierarchical tasks often combine long prerequisite chains with sparse rewards and learning dependencies from limited successful experience remains a challenge. Cyclic environments add an additional complexity to the task. Our aim is to explore this hierarchical planning under such conditions, by comparing GFNs with entropy-regularized reinforcement learning under matched budget and conditions to isolate the exact properties needed to succeed over long horizons with delayed rewards.

1.1 CONTRIBUTIONS

This work builds a cyclic GFN implementation that approaches or exceeds the success rate of standard RL methods like PPO and SAC and finds more strategic modes than those methods, and studies whether reward-proportional sampling can be carried over to a cyclic, long-horizon crafting hierarchy, and how it performs against SoftDQN, its RL counterpart. Our contributions are as follows.

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.

- We provide the first practical instantiation of SubTB with a Retrospective Backward Synthesis (RBS) inspired replay mechanism (He et al., 2025) in a non-acyclic GFN which extends from Morozov et al. (2025), and train it at a fixed exploration budget of 100 to 250 steps with one demonstration only.
- We extend the instantiation to HierarchyCraft, a hierarchical crafting environment with deep prerequisite chains and substitutable routes, and validate the results. To our knowledge this is the first study of GFNs in an environment of this kind.
- We build a functional GFN in a cyclic environment, and show that it achieves similar or higher success rates on most tasks and higher diversities in long-horizon and sparse-signal tasks than standard RL methods and its RL counterpart SoftDQN (Tiapkin et al., 2024), with about two to four times diversity on top of similar or higher success rates.
- We provide insights into differences in GFN and SoftDQN and suggest that GFN's credit assignment mechanism contributes to its higher performance. We also observe that SubTB keeps most of its success under a shifted backward policy while one-step TD stays near zero.

2 Related Work

2.1 Generative flow networks

Bengio et al. (2021) introduced GFNs to generate objects with probability proportional to their reward, and Bengio et al. (2023) developed the mathematical foundations of this framework. Forward and backward flows can be balanced on individual transitions, complete trajectories (Malkin et al., 2022), or every consecutive segment of a trajectory through sub-trajectory balance (SubTB), which trades off bias and variance and can help training with long action sequences and sparse rewards (Madan et al., 2023). Applications such as biological sequence design (Jain et al., 2022), Bayesian structure learning (Deleu et al., 2022), and combinatorial optimization (Zhang et al., 2023) commonly construct objects through a directed acyclic graph (DAG).

Brunswic et al. (2024) extend GFNs to cyclic graphs, and show that existing losses can encourage flow to accumulate in cycles, and propose a family of losses to address this. Morozov et al. (2025) presents a framework for discrete cyclic environments that trains with a fixed backward policy. To overcome slow convergence with detailed balance in pathfinding, Morozov et al. (2026) proposes using a sum of trajectory-balance terms over segments that begin at the initial state. However, the long-horizon planning remains comparatively less explored despite theoretical advances.

2.2 Reinforcement learning

Maximum-entropy reinforcement learning (RL) maximizes the entropy in a policy to spread probability across actions, and its optimal policy takes a Boltzmann form that still favors actions with higher values (Ziebart et al., 2008; Haarnoja et al., 2017). Soft actor-critic (Haarnoja et al., 2018) and its discrete variant (Christodoulou, 2019) are widely used implementations. Tiapkin et al. (2024) connect GFN training to entropy-regularized RL through soft DQN and Munchausen DQN (Vieillard et al., 2020), and Morozov et al. (2025) extend this connection to cyclic graphs. Mohammadpour et al. (2024) establishes a corresponding result from the maximum-entropy perspective and constructs a maximum-entropy backward policy. The connection depends on the backward policy, which Gritsaev et al. (2025) optimizes by maximizing the value function of an entropy-regularized Markov decision process (MDP), thereby leading to faster convergence and a broader discovery of rewarding solutions.

Long horizons also make it harder to correlate early decisions with later rewards. One-step temporal-difference (TD) learning passes reward information backward through successive updates via individual transitions, and n-step returns let it use rewards from several steps ahead (Sutton & Barto, 2018). These returns can introduce bias when the sampled actions come from a different policy. Deep crafting tasks further combine long prerequisite chains with sparse rewards, thereby motivating hierarchical policies whose decisions span multiple actions (Sutton et al., 1999; Nachum et al., 2018), exploration strategies (Pathak et al., 2017; Ecoffet et al., 2021), and learning from demonstrations (Hester et al., 2018). While some methods encourage behavioral diversity through

an additional objective (Eysenbach et al., 2019), GFNs seek diverse solutions by learning a reward-proportional distribution. In symbolic and fully observed domains, classical and hierarchical planners can provide a route to a goal (Helmert, 2006; Erol et al., 1994), and top-k planning returns several distinct plans under a specified criterion (Katz et al., 2018). However, diversity remains mostly an auxiliary objective relative to the success rate.

3 PRELIMINARIES

3.1 SUB-TRAJECTORY BALANCE

We follow the sub-trajectory balance (SubTB) formulation of Madan et al. (2023). A trajectory $\tau = (s_0, \dots, s_T)$ runs from the initial state s_0 to a terminal state x, which connects to a sink state s_f and receives a reward $R(x) > 0$, with $Z = \sum_x R(x)$. SubTB asks a forward policy $P_F(s' \mid s)$, a backward policy $P_B(s \mid s')$ over the parents $\mathrm{in}(s')$, and a state flow $F(s)$ to satisfy, for every segment $0 \le m < n \le T$,

$$F(s_m) \prod_{t=m}^{n-1} P_F(s_{t+1} \mid s_t) \;=\; F(s_n) \prod_{t=m}^{n-1} P_B(s_t \mid s_{t+1}), \tag{1}$$

which reduces to detailed balance (DB) when $n = m + 1$ and to trajectory balance (Malkin et al., 2022) on a full trajectory with $F(s_0) = Z$ and $F(x) = R(x)$. Training minimizes the squared log residual

$$D_{m,n} \;=\; \log F_\theta(s_m) - \log F_\theta(s_n) + \sum_{k=m}^{n-1} \big[\log P_F(s_{k+1} \mid s_k) - \log P_B(s_k \mid s_{k+1})\big], \tag{2}$$

averaged over segments with weights set by λ_{sub}, where $\lambda_{\mathrm{sub}} = 1$ weights all segments equally.

3.2 ENTROPY-REGULARIZED RL

Tiapkin et al. (2024) express GFN training as an entropy-regularized RL problem by constructing, for a fixed P_B, an MDP on the same graph with

$$r(s \to s') = \log P_B(s \mid s') \text{ for } s' \neq s_f, \qquad r(x \to s_f) = \log R(x), \qquad \gamma = 1, \qquad \lambda = 1, \tag{3}$$

where γ is the discount factor and λ is the entropy coefficient. With the policy $\pi = \mathrm{Softmax}_\lambda(Q)$, the soft Bellman optimality equations

$$Q^\star(s, s') = r(s \to s') + V^\star(s'), \qquad V^\star(s) = \mathrm{LogSumExp}_\lambda\big(Q^\star(s, \cdot)\big) \tag{4}$$

are solved by $Q^\star(s, s') = \log F(s \to s')$, $V^\star(s) = \log F(s)$, and $\pi^\star = P_F$, so solving this MDP recovers the GFN forward policy. Substituting $V = \mathrm{LogSumExp}(Q)$ into Eq. equation 4 makes the TD error equal to the one-step residual $D_{m,m+1}$ of Eq. equation 2, equivalent to SoftDQN on the right-hand side over a trajectory. The equivalence fixes the optimum of both methods but how each method arrives at it remains open. Therefore, a gap under a fixed budget likely reflects how each objective learns.

3.3 GFNS ON CYCLIC GRAPHS

Cycles in the graph allow a trajectory to revisit states, so flow may get stuck in loops, and the expected trajectory length can become arbitrarily large (Brunswic et al., 2024; Morozov et al., 2025). Nonetheless, Morozov et al. (2025) proposed tackling the repeated visits by redefining edge flow as an expected visit count,

$$F(s \to s') \;=\; F(s_f) \cdot \mathbb{E}_{\tau \sim P_F}\Big[\textstyle\sum_t \mathbf{1}\{s_t = s,\ s_{t+1} = s'\}\Big], \tag{5}$$

where $F(s_f)$ is the total flow. Under their assumptions, which include a finite graph where every state lies on a path from s_0 to s_f, a fixed backward policy with $P_B(s \mid s') > 0$ on every edge guarantees a finite expected length of the trajectory, and any loss from the acyclic setting learns the corresponding forward policy with a unique solution. However, their experiments train with detailed balance and its stable variant, so how SubTB behaves in this setting remains open. Further, their benchmarks, namely hypergrids and permutation groups, are synthetic, and how the framework fares in tasks with long prerequisite chains remains untested.

4 METHOD

4.1 PROBLEM SETTING

We apply GFN to a discrete environment over a finite state space, where a finite set of transformations carries one state to another until a goal state is reached. Prerequisite chains run deep, so a goal state often sits many transformations above s_0 and states can traverse in loops. The transformation set contains invertible pairs, so a trajectory that leaves a state can return to it. Breaking the acyclicity assumption creates significant challenges for the vanilla GFN formulation. Additionally, we target a task environment where several distinct transformation sequences reach the same goal state, so that diversity is a desirable trait. A fitting environment that satisfies all properties is the HierarchyCraft proposed by Fédérico et al. (2026), which will be detailed in Section 5.

4.2 TRAINING OBJECTIVE

We adopt the expected-visit flow of Eq. equation 5 throughout, and hold P_B fixed at uniform over the reconstructed parent set, $\log P_B(s_k \mid s_{k+1}) = -\log|\mathrm{in}(s_{k+1})|$. The uniform P_B is strictly positive on every edge, so following Morozov et al. (2025), the expected trajectory length is finite. With that, every objective from the acyclic literature applies unchanged, including our SubTB instantiation. We take the detailed-balance identity as noted by Morozov et al. (2025) and multiply it edge by edge along a segment $s_m \to \cdots \to s_n$. The log-flow head regresses a log expected visit count, so a state on a repeatedly traversed cycle can legitimately carry $F(s) > Z$ and we leave the head unnormalized.

The goal-conditioned network carries the forward policy and the flow, taking the goal as a learned embedding alongside the observation. We train Eq. equation 2 at $\lambda_{\mathrm{sub}} = 1$ as written, writing $x_k = \log P_F(s_{k+1} \mid s_k) - \log P_B(s_k \mid s_{k+1})$ for the per-step term, and the whole matrix of segment discrepancies follows from one $O(T)$ forward pass and one prefix-sum.

For a trajectory $\tau = (s_0, a_0, \ldots, s_T)$, we follow the standard GFN boundary condition and substitute $F(x) = R(x)$ at terminal states, using the observed $\log R(s_T)$ as the flow head's prediction at the last state. A trajectory that reaches its goal takes the task's $\log R$ from Table 1. A trajectory that exhausts its budget is treated as terminal at its last state with a fixed $\log R = -10$ (Appendix A). With this substitution, each of the T segments ending at s_T, from $D_{0,T}$ to $D_{T-1,T}$, ties the flow at an earlier state directly to the observed reward, so a single trajectory can carry its reward to every state along it in one update. SoftDQN applies the same condition by adding $\log R(x)$ to the reward of its final transition (Appendix B).

The trained objective adds one term at a fixed weight,

$$\mathcal{L} \;=\; \mathcal{L}_{\mathrm{SubTB}} + w_{\mathrm{TB}}\mathcal{L}_{\mathrm{TB}}, \tag{6}$$

where $\mathcal{L}_{\mathrm{TB}} = \mathbf{1}\{\tau \text{ succeeds}\}(\log Z_\theta - \log R(s_T) + \sum_k x_k)^2$ is the only place a global scale is tied to $\log R$.

4.3 DEMONSTRATIONS AND REPLAY

Following work that bootstraps learning from a single demonstration (Resnick et al., 2018; Salimans & Chen, 2018), we assume one successful trajectory per goal is available. A randomly initialized policy reaches a deep goal with extremely low chance in long horizon tasks, so some supply of successes is necessary for the agent to begin. We obtain that first supply from a walker that terminates at the goal state. Inspired by the shared-goal structure and joint replay of He et al. (2025), our implementation samples a batch of goals per iteration and passes it to the current policy. During initialization, the walker uses the same goals. Goal batching and separate replay rings are design choices of our implementation.

The on-policy candidates that the agent later finds are checked against the existing buffer content. Each entry has a set of distinct actions it uses, and a new trajectory is only admitted when the key is different, so a reordering of an existing route is rejected. Each ring holds 32 slots and overwrites its oldest entry once it fills.

Together, these choices kick-start the training while keeping the demonstrations minimal. In our experiments, one sample for each buffer is enough to start the training, after which buffer entries

are dominated by the model's own on-policy successes. The same configuration is applied to both the GFN and SoftDQN, and it enables training on long-horizon tasks that would otherwise fail. SoftDQN calls the same walker on the same schedule and reads its per-transition $\log P_B$ off the same parent mask field the SubTB loss reads, so both see the same backward policy on the same trajectory. Temporal-difference learns from transitions, so each walker trajectory is flattened for every minibatch.

4.4 BASELINES

We first compare the GFN with two widely used reward-maximizing methods, PPO (Schulman et al., 2017) and discrete SAC (Christodoulou, 2019). Both share the goal-conditioned backbone layout, action masks, walker demonstrations, and evaluation protocol of the GFN, and they maximize a return discounted at $\gamma = 0.99$ that combines the task's success bonus, a failure penalty, and a small per-step cost.

We focus our analysis on SoftDQN, which is identified in Section 3.2 as the GFN's equivalent RL counterpart. Because the two share the same optimum, differences between them may point to how each objective carries the reward during training. The equivalence holds under the properties below:

- **Fixed backward policy.** Both settings read $\log P_B$ from the same uniform policy over the same reconstructed parent set, computed by the same routine.
- **Discount and entropy coefficient.** Both are held at $\gamma = 1$ and $\lambda = 1$.
- **Rewards.** The intermediate reward is $\log P_B$ per transition and the terminal reward is $\log R(x)$.
- **Deterministic kernel with an absorbing s_f.** HierarchyCraft environment is deterministic, and the terminal edge folds into the last transition (Appendix B).

5 EXPERIMENTAL SETUP

5.1 HIERARCHYCRAFT

All experiments run in HierarchyCraft (Fédérico et al., 2026), which represents a Minecraft-inspired tech tree through 201 transformations over 51 items and 8 zones. Both GFN and SoftDQN use the same backbone and shared configurations, with details in Appendix A. Crafter provides a benchmark with a predefined progression of crafting dependencies (Hafner, 2022), while our study uses HierarchyCraft's symbolic transformation graph to examine alternative routes. For Minecraft, the MineRL competition emphasizes sample efficiency under explicit interaction and training budgets (Guss et al., 2021). These considerations motivate our choice of a compact symbolic environment for experiments across several methods and seeds. HierarchyCraft abstracts the low-level visual details into symbolic observations, allowing both objectives to use the same representations and helping clarify differences in their learning behaviour.

The tech tree of HierarchyCraft environment provides the testbed needed. Dependencies are recursive (e.g. an iron ingot requires a stone pickaxe, which requires cobblestone and a wooden pickaxe) and prerequisites are only partially ordered, so independent sub-goals may be interleaved in many valid orders. And routes are genuinely substitutable (e.g. a higher-tier pickaxe serves wherever a lower-tier one is required). Meanwhile, those same features make the transition graph cyclic (e.g. location traversal, multiple recipes).

5.2 TASKS

As mentioned in the previous section, a successful demo is required to warmup the training for long horizon tasks. We use eight HierarchyCraft tasks, listed in Table 1. Seven are single-goal, and "multi" task has two terminal goals from the single-goals. The first three are shallow tasks, easy for both methods, while the five deep tasks are more challenging. Plan lengths run from 28 to 89 steps, so training horizons are set to 100 to 250 steps respectively. The signal is sparse as it is only provided at the goal state. With the walker disabled, every setting we ran scores 0.0% on iron, gold, diamond and ender-eye (Section 6.2). A detailed visualization of the tasks can be found here link.

Table 1: The eight HierarchyCraft tasks. $|\pi^\star|$ is the lower bound of the length of the environment's own plan, $T_{\max}$ the training horizon, and evaluation uses $1.5\,T_{\max}$ of headroom so that detoured yet valid plans score as successes.

Task	Goal item(s)	Goals	$\lvert\pi^\star\rvert$	$T_{\max}$	$\log R$
wood-pickaxe	wooden pickaxe	1	10	30	50
stone-pickaxe	stone pickaxe	1	14	35	50
cobblestone	cobblestone	1	11	40	50
iron	iron ingot	1	28	100	100
diamond	diamond	1	34	200	175
gold	gold ingot	1	47	150	150
ender-eye	ender eye	1	89	250	200
multi	iron / gold ingot	2	28 / 47	250	200

5.3 Matched settings

The same implementation of the buffer is used by both GFN and SoftDQN with the same capacity, exponents, goal sampler, and initial demo, on the same environment instance and seed, so environment interactions are matched exactly. Parameter counts match exactly between the GFN and SoftDQN, with surplus on SoftDQN's side during training. All results are inferred from the best recorded checkpoints of each method, which includes mid-training peak. The best checkpoint is chosen by the periodic evaluation during training, and then inferenced for a final evaluation for every reported metric. Other related RL methods are included in the Appendix H.

5.4 Mode counting

A naive counting of distinct plans encourages lengthy routes, because a longer plan carries more opportunities to pad a procedure with strategically irrelevant steps. We therefore replay each successful plan τ and greedily remove actions for as long as the replay still reaches the goal, which leaves an essential plan $\tilde{\tau}$ behind. We then reduce it to a fingerprint over its strategic decision points alone. Writing $\mathcal{G}$ for the items on the goal's dependency closure that admit more than one recipe, $g \in \mathcal{G}$ for the recipes of one such item with location variants of a single recipe gathered into one $r \in g$, and $A(\cdot)$ for the set of action indices a plan uses,

$$\phi_g(\tau) \;=\; \{\, r \in g \mid r \cap A(\tilde{\tau}) \neq \emptyset \,\}, \qquad \text{Modes} \;=\; \big|\{\, \phi(\tau) \mid \tau \in \mathcal{S} \,\}\big|, \tag{7}$$

where $\phi(\tau)$ gathers $\phi_g(\tau)$ over the groups the plan touches and $\mathcal{S}$ is the pooled set of successful rollouts from the final evaluation of one seed, which runs 640 to 1280 stochastic rollouts. The Modes column of Table 2 is computed through this equation and averaged over seeds. The walker provides only one demo, so the additional modes all come from trajectories that the agent discovers.

6 Results

In our experiments with tuned configurations, the GFN achieves the highest success rate among the methods we ran on the two longest single-goal tasks, namely gold and ender-eye, and stays within five points of the best reward-maximizing baseline on iron and diamond, and trails SoftDQN and PPO on multi. It also finds the most strategic modes on every deep task (Appendix H). In particular, the GFN and SoftDQN both perform well on the shallow tasks, while their performance diverges on most of the long-horizon ones.

Table 2 reports both experiments on all eight tasks. Every metric is drawn from the same inference-only evaluation of the trained checkpoints, where success rate, mean length and the mode count of Eq. equation 7 are averaged over seeds. SoftDQN additionally keeps its pinned demonstration to prevent collapses, which results in surplus exposure on its side.

6.1 GFN versus SoftDQN

Both settings saturate on the three shallow tasks and on diamond. On the remaining four deeper tasks, the GFN leads throughout by a considerable margin.

Table 2: SubTB GFN compared with SoftDQN at equal configuration, ten seeds run averaged for stochastic inference. Deep tasks are ordered by reference-plan length. Modes and length average over the seeds with at least one successful rollout, and a dash marks a cell with none.

	GFN (SubTB)			SoftDQN			
Task	Succ. %	Modes	Length	Succ. %	Modes	Length	Limit
wood-pickaxe	99.9	1.0	17.8	99.9	1.0	16.1	45
stone-pickaxe	99.0	1.0	25.1	96.0	1.1	26.6	52
cobblestone	100.0	1.0	20.0	99.2	1.0	23.7	60
iron	**99.4**	**4.4**	52.7	87.4	3.7	85.3	150
diamond	**99.9**	**6.0**	68.7	99.1	2.5	118.0	300
gold	**90.6**	**16.0**	92.6	4.6	4.5	192.7	225
ender-eye	**7.9**	**6.7**	141.7	0.0	–	–	375
multi	**87.2**	5.2	59.5	36.5	**7.4**	188.4	375

We hypothesize that the difference arises from how far one update carries the observed reward. Both settings store the same replays and read the same backward policy, so what separates them is the set of balance constraints each objective imposes on one pass over a stored trajectory. Section 3.2 identifies the SoftDQN residual with the detailed-balance case of a segment of a single step. Writing x_k for the per-step term of Section 4 and carrying the terminal substitution into each, the residual through which an objective sees the reward is

$$\begin{aligned} \text{SubTB} \qquad D_{0,T} &= \log F_\theta(s_0) - \log R(s_T) + \sum_{k=0}^{T-1} x_k, \\ \text{SoftDQN} \qquad \delta_{T-1} &= \log F_\theta(s_{T-1}) - \log R(s_T) + x_{T-1}. \end{aligned} \tag{8}$$

Each residual starts at its segment's first node and closes on the same observed $\log R(s_T)$, so the contrast sits in the partial sum, which SubTB carries over the whole trajectory.

Under Eq. 8, the reward reaches SoftDQN at the final transition alone, whereas SubTB writes it into every node in one pass. Each earlier node in SoftDQN learns from the flow of the node after it, propagating along the trajectory one node at a time because it needs to wait for its successor node to absorb the reward first. This lag hurts learning efficiency as shown by the slower convergence than GFN in the results. It also makes the learning signal sparser because most nodes on the trajectory hold little information about the final reward. The two effects combined appear to worsen the performance of SoftDQN on long-horizon tasks despite the same setup. On the other hand, GFN's single trajectory update transmits the reward signal in one update, making it both efficient and fast in learning from the successes found. Under realistic budget, the GFN formulation can achieve the goal faster without losing diversity exploration.

A natural way to compensate for the SubTB partial sum is to lengthen the SoftDQN update steps so that its target spans several steps. In a deterministic environment, the n-step soft target with its entropy correction also stays consistent along any stored route at the optimum, which is the path consistency of Nachum et al. (2017). In our runs, lengthening the step speeds early learning and later loses stability (Section 6.2), which suggests that the SubTB sum trains more stably than a direct n-step extension of SoftDQN.

6.2 Ablations

To ensure that our configurations are reasonable, we have conducted ablation studies to examine each.

Walker. The walker appears to be what makes the deep single-goal tasks learnable. With it disabled the buffer starts empty, and every configuration we ran, covering the GFN of Table 2 and the five methods of Appendix H, averages 0.0% on iron, gold, diamond and ender-eye (Appendix D). Multi keeps partial success, with two GFN seeds near 48% and one at 0.0%. The successful rollouts we inspected all end by smelting an iron ingot, which suggests that the 250-step multi horizon lets exploration reach the shallower of its two goals while gold stays out of reach. Among the shallow tasks, wood-pickaxe and cobblestone stay at or above 98% for both the GFN and SoftDQN, while stone-pickaxe falls to 0.2% or lower for all five methods.

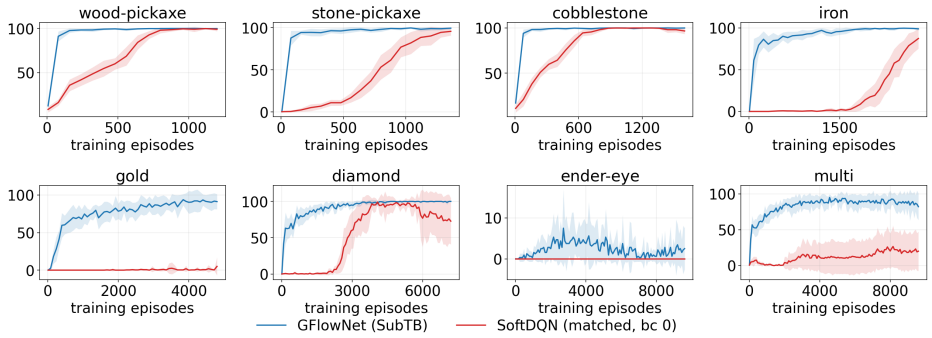

Figure 1: Evaluation of success rate against training episodes, one panel per task, mean over ten seeds with a ± 1 SD band. The shallow tasks and diamond saturate for both experiments, while elsewhere the gap opens early and stays open. Reported results use the best checkpoints for both.

Demonstration exposure. This study tests how much SoftDQN relies on the demo. We vary the fraction of the gradient signal of the demo trajectory to examine its effect. SoftDQN learns from single transitions, so the walker's trajectory is cut into its steps, and in Table 2 each walker step appears about one to four times in every update, depending on the plan length. For comparison, the demo trajectory carries about 8% of a GFN update. We gradually decay the proportion during training. Each run starts at 1.0, where every update comes from the demo, and lowers the share linearly to the GFN-equivalent 0.08 or to 0 over either the full run or its first half (Appendix E). Every schedule scores below the fixed share of Table 2 on iron and diamond, and gold and ender-eye stay at zero throughout. The GFN stays ahead of SoftDQN under every configuration.

Backward policy and backup length. We further test SoftDQN's backward policy and step length in an attempt to approximate GFN in Appendix F. Crossing the objective with the backward policy keeps SubTB at or ahead of one-step TD under either specification on all five deep tasks, and SoftDQN appears far more sensitive to error in P_B. Under the shifted backward policy (Appendix F.1), SubTB stays above 90% on iron, diamond and gold while one-step TD stays near zero on all four single-goal deep tasks. While lengthening the SoftDQN step from $n = 1$ to $n = 4$ speeds learning on iron by about $3.4\times$, every run above $n = 1$ peaks early and then falls to zero, suggesting low stability.

6.3 Sample efficiency

Figures 1 and 2 show episode-wise success and mode curves for the two settings of Table 2, and on-policy training curves are in Appendix G. The separation is one of sample efficiency as much as of final score, which is consistent with the update mechanism described earlier. A flow objective revises a whole trajectory on a single success, so one rare successful episode leaves the forward policy usable along that entire route, which is the regime these tasks present early in training while successes stay scarce.

7 Discussion and Limitations

Our SoftDQN implementation mimics a GFN as closely as we could. Two closely related methods with the same optimum might be expected to behave similarly, yet they still differ in performance. Credit assignment is the explanation that our experiments on HierarchyCraft directly suggest. While one SubTB update feeds a whole trajectory at once, a SoftDQN update has to traverse the step-by-step transitions toward the reward.

Three limitations bound our results. First, we abstract the low-level worker away, since a HierarchyCraft action always succeeds once its preconditions hold. Real deployments fail at that layer, so our success rates could be much worse in real applications. However, the strength of GFN is that a flow objective already models a distribution over routes, so future work can exploit that flexibility. Second, every task names a terminal item or state, which restricts the generative capabilities of the

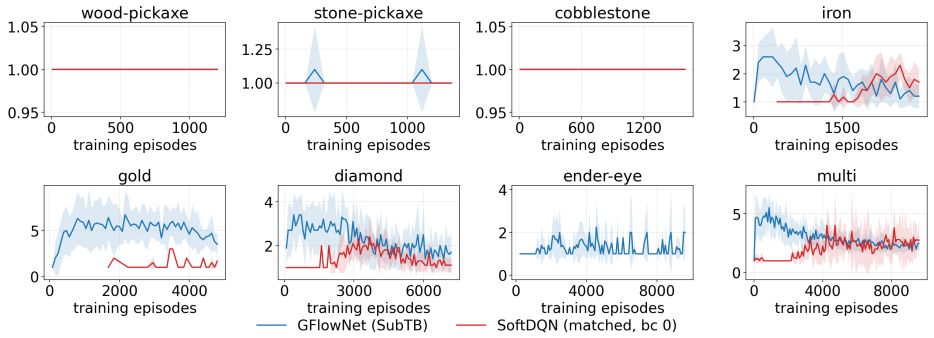

Figure 2: Strategic modes against training episodes, same layout as Figure 1. These curves are computed on each evaluation cycle's own small rollout sample, whereas the pooled end-of-run counts in Table 2 use far more rollouts, and since mode counts grow with sample size the curves sit well below the final values. SoftDQN contributes no curve on ender-eye, where it produces no successful rollout in any of the ten seeds.

models. While the suite is representative, a study one hierarchy up, with the goal stated as a purpose and the object left to the agent, would exploit a GFN better. Third, despite our best effort to deflate the mode counts, we can still observe some repetitions in the strategies by manually examining them. The mode should be considered for general trend between the methods compared in this work.

8 Conclusion

We applied the non-acyclic GFN framework of Morozov et al. (2025) with sub-trajectory balance and trained it on HierarchyCraft, a cyclic crafting hierarchy with deep prerequisite chains and substitutable routes. Against SoftDQN, which shares its optimal policy through the reduction of Tiapkin et al. (2024), the GFN matches on the shallow tasks and leads on most deep tasks in both success and strategic diversity. The gap appears to arise from how far one update carries the reward. These results suggest that the choice of loss can matter even when two methods share the same optimum, and combining trajectory-level flow objectives with soft RL appears to be a promising direction.

AI use statement

In this work, we used generative AI tools for literature reviews, manuscript editing, algorithm implementation and visualization. The ideation, theoretical framework, mathematical claims, hypotheses, experimental design and interpretation of results are the authors' own work. Author 1 reviewed and verified all generated code. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

References

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

Leo Brunswic, Yinchuan Li, Yushun Xu, Yijun Feng, Shangling Jui, and Lizhuang Ma. A theory of non-acyclic generative flow networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 11124–11131, 2024.

Petros Christodoulou. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207*, 2019.

Tristan Deleu, António Góis, Chris Emezue, Mansi Rankawat, Simon Lacoste-Julien, Stefan Bauer, and Yoshua Bengio. Bayesian structure learning with generative flow networks. In *Proceedings of the 38th Conference on Uncertainty in Artificial Intelligence*, 2022.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. First return, then explore. *Nature*, 590(7847):580–586, 2021.

Kutluhan Erol, James Hendler, and Dana S. Nau. HTN planning: Complexity and expressivity. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pp. 1123–1128, 1994.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2019.

Mathis Fédérico, Shang Wang, Yuxuan Li, and Matthew E. Taylor. HierarchyCraft: A benchmark builder for hierarchical reasoning. *Journal of Open Source Software*, 11(119):6468, 2026. doi: 10.21105/joss.06468.

Timofei Gritsaev, Nikita Morozov, Sergey Samsonov, and Daniil Tiapkin. Optimizing backward policies in GFlowNets via trajectory likelihood maximization. In *International Conference on Learning Representations*, 2025.

William H. Guss, Mario Ynocente Castro, Sam Devlin, Brandon Houghton, Noboru Sean Kuno, Crissman Loomis, Stephanie Milani, Sharada Mohanty, Keisuke Nakata, Ruslan Salakhutdinov, John Schulman, Shinya Shiroshita, Nicholay Topin, Avinash Ummadisingu, and Oriol Vinyals. The MineRL 2020 competition on sample efficient reinforcement learning using human priors. *arXiv preprint arXiv:2101.11071*, 2021.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *PMLR*, pp. 1352–1361, 2017.

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*, volume 80 of *PMLR*, pp. 1861–1870, 2018.

Danijar Hafner. Benchmarking the spectrum of agent capabilities. In *International Conference on Learning Representations*, 2022.

Haoran He, Can Chang, Huazhe Xu, and Ling Pan. Looking backward: Retrospective backward synthesis for goal-conditioned GFlowNets. In *International Conference on Learning Representations*, 2025.

Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246, 2006.

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, Gabriel Dulac-Arnold, John Agapiou, Joel Z. Leibo, and Audrunas Gruslys. Deep Q-learning from demonstrations. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pp. 3223–3230, 2018.

Moksh Jain, Emmanuel Bengio, Alex Hernandez-Garcia, Jarrid Rector-Brooks, Bonaventure F. P. Dossou, Chanakya Ajit Ekbote, Jie Fu, Tianyu Zhang, Michael Kilgour, Dinghuai Zhang, Lena Simine, Payel Das, and Yoshua Bengio. Biological sequence design with GFlowNets. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *PMLR*, pp. 9786–9801, 2022.

Michael Katz, Shirin Sohrabi, Octavian Udrea, and Dominik Winterer. A novel iterative approach to top-k planning. *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 28:132–140, 2018.

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 *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *PMLR*, pp. 23467–23483, 2023.

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*, volume 35, pp. 5955–5967, 2022.

Sobhan Mohammadpour, Emmanuel Bengio, Emma Frejinger, and Pierre-Luc Bacon. Maximum entropy GFlowNets with soft Q-learning. In *Proceedings of the 27th International Conference on Artificial Intelligence and Statistics*, volume 238 of *PMLR*, pp. 2593–2601, 2024.

Nikita Morozov, Ian Maksimov, Daniil Tiapkin, and Sergey Samsonov. Revisiting non-acyclic GFlowNets in discrete environments. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *PMLR*, pp. 44887–44910, 2025.

Nikita Morozov, Ian Maksimov, Daniil Tiapkin, and Sergey Samsonov. Learning shortest paths with generative flow networks. *arXiv preprint arXiv:2603.01786*, 2026.

Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the gap between value and policy based reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 31, pp. 3303–3313, 2018.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *PMLR*, pp. 2778–2787, 2017.

Cinjon Resnick, Roberta Raileanu, Sanyam Kapoor, Alexander Peysakhovich, Kyunghyun Cho, and Joan Bruna. Backplay: "Man muss immer umkehren". In *AAAI Workshop on Reinforcement Learning in Games*, 2018.

Tim Salimans and Richard Chen. Learning Montezuma's revenge from a single demonstration. In *NeurIPS Deep Reinforcement Learning Workshop*, 2018.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *International Conference on Learning Representations*, 2016.

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

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211, 1999.

Daniil Tiapkin, Nikita Morozov, Alexey Naumov, and Dmitry P. Vetrov. Generative flow networks as entropy-regularized RL. In *Proceedings of the 27th International Conference on Artificial Intelligence and Statistics*, volume 238 of *PMLR*, pp. 4213–4221, 2024.

Nino Vieillard, Olivier Pietquin, and Matthieu Geist. Munchausen reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 4235–4246, 2020.

Dinghuai Zhang, Hanjun Dai, Nikolay Malkin, Aaron Courville, Yoshua Bengio, and Ling Pan. Let the flows tell: Solving graph combinatorial optimization problems with GFlowNets. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Brian D. Ziebart, Andrew L. Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, pp. 1433–1438, 2008.

Table 3: Hyperparameters shared across the eight tasks for the main comparison.

Setting	Value
Observation width	66
Goal embedding width	32
Trunk	2×128, ReLU
Action space	201 transformations
Optimizer	AdamW, weight decay 0.01
Learning rate, trunk and heads	3×10^{-4}
Learning rate, $\log Z_\theta$	1×10^{-2}
Learning-rate schedule	constant
Goals sampled per iteration	8
Gradient updates per epoch	8
SubTB weighting λ_{sub}	1
Trajectory-balance weight w_{TB}	1.0
Backward policy P_B	uniform over reconstructed parents
Exploration ϵ	0.30 to 0.05 over the first half of training
Buffer slots per ring	32
Buffer admission	one entry per goal and action set
Walker epochs	1
SoftDQN demo share	0.4 of each minibatch from the walker pool
Step penalty	0.0
Failure $\log R$	-10
Seeds	10

A Backbone Network and Hyperparameters

Every setting shares one backbone. The 66-dimensional symbolic observation is concatenated with a learned 32-dimensional goal embedding and passed through two fully connected layers of width 128 with ReLU activations. The GFN reads a forward head of 201 logits, masked to the actionable set and shifted by a per-goal action bias, a flow head for $\log F_\theta(s, g)$, and a scalar $\log Z_\theta$. Table 3 lists the shared settings. The training horizon and success bonus of each task follow Table 1, and the epoch budget scales with the horizon.

B SoftDQN Implementation Details

We implement SoftDQN from the equations of Tiapkin et al. (2024) within our own harness, so that it shares the buffer, walker and backbone of the GFN. HierarchyCraft ends an episode at the goal state, so the last transition carries $r = \log P_B + \log R(x)$ and is marked done, as in the authors' implementation. The value head's output bias is initialized at $\log R$, which leaves the fixed point in place since $\mathrm{Softmax}_\lambda(Q)$ is invariant to a constant shift. SoftDQN uses a dueling Q head, a Polyak-averaged target network with $\tau = 0.05$, a Huber loss, and minibatches of 256 transitions. The Munchausen coefficient (Vieillard et al., 2020) is zero in every reported run, and the n-step backups of Appendix F.2 appear only in that study.

C Demonstration Synthesis and Buffer Details

Our replay procedure follows the shared-goal structure of He et al. (2025), with a symbolic walker in place of their learned backward synthesizer. It samples a batch of goals and rolls the current policy on them. At the first epoch the walker produces a successful trajectory for the goal, stored in the same format as an on-policy success. It rolls the environment's own solution once and permutes it by swapping adjacent commuting actions, and the permuted sequence is replayed from a reset and kept when every action is legal and the goal arrives on the final step.

The buffer holds three rings, for on-policy successes, walker trajectories, and failed trajectories at the floor reward. The success ring is prioritized (Schaul et al., 2016). Each GFN update draws up to eight successes, one trajectory per walker slot, and four failures. SoftDQN flattens each stored trajectory into transitions and reads the per-transition $\log P_B$ from the same stored parent mask as the SubTB loss. Its on-policy transitions enter a replay ring, and the demo transitions enter a separate pool that supplies 40% of every minibatch.

D Disabling the Walker

Table 4 reports success rate with the walker disabled (i.e. no demo) and every other setting unchanged. The first column is the GFN of Table 2 on the five deep tasks, and the remaining columns are the presets of Appendix H. Every configuration scores 0.0% on the four single-goal deep tasks.

Table 4: Success rate with the walker disabled, mean $\pm$ SD over three seeds, two for the GFN preset on ender-eye. The first column covers the five deep tasks. A bare entry varies by at most 0.1 across seeds.

	Table 2	Presets of Appendix H				
Task	GFN	GFN	SoftDQN	PPO	SAC	SAC-bc2
wood-pickaxe	–	98.3 ± 0.9	98.6 ± 1.2	67.8 ± 55.6	82.4 ± 4.3	99.4 ± 0.3
stone-pickaxe	–	0.2	0.2 ± 0.1	0.1 ± 0.1	0.2 ± 0.2	0.2 ± 0.2
cobblestone	–	98.9 ± 1.1	100.0	68.7 ± 54.3	91.4 ± 10.2	99.9 ± 0.1
iron	0.0	0.0	0.0	0.0	0.0	0.0
diamond	0.0	0.0	0.0	0.0	0.0	0.0
gold	0.0	0.0	0.0	0.0	0.0	0.0
ender-eye	0.0	0.0	0.0	0.0	0.0	0.0
multi	32.0 ± 27.7	47.4 ± 1.6	24.6 ± 2.9	33.2 ± 28.7	1.5 ± 1.1	32.7 ± 26.0

E Annealing the SoftDQN Demo Share

The demo share is the fraction of each SoftDQN minibatch drawn from the walker pool, set to 0.4 in Table 2. Each schedule starts at 1.0 and lowers the share linearly to 0.08, about the walker's share of a GFN update once its success ring has filled, or to 0, over the full run or its first half. A fixed share of 0 places the walker transitions in the main replay ring. Every other setting follows Table 2. Every schedule stays below the GFN on all five deep tasks.

Table 5: SoftDQN success rate under schedules for the demo share, mean $\pm$ SD over the seeds listed. A schedule $a \to b$ lowers the share linearly from a to b over the stated span and then holds it at b. The GFN row repeats Table 2. A bare entry takes the same value on every seed.

Demo pool share	Seeds	iron	gold	diamond	ender-eye	multi
fixed 0.4	10	87.4 ± 12.3	4.6 ± 9.2	99.1 ± 1.4	0.0	36.5 ± 17.7
$1.0 \to 0.08$ (full run)	2	65.5 ± 15.1	0.0	96.9 ± 2.2	0.0	29.1 ± 28.7
$1.0 \to 0.08$ (first half)	2	80.0 ± 10.4	0.0	48.5 ± 66.6	0.0	53.4 ± 6.7
$1.0 \to 0$ (full run)	2	76.6 ± 2.1	0.0	84.5 ± 17.2	0.0	36.4 ± 5.3
$1.0 \to 0$ (first half)	2	0.6 ± 0.5	0.0	27.2 ± 37.3	0.0	29.8 ± 8.6
fixed 0	3	0.1 ± 0.1	0.0	0.0	0.0	1.7 ± 1.5
GFN (SubTB)	10	99.4 ± 0.8	90.6 ± 11.4	99.9 ± 0.1	7.9 ± 6.6	87.2 ± 10.9

F Backward Policy and Backup Length

F.1 Backward policy

Table 6 crosses the objective with two reward specifications and a behavior-cloning weight of 0.0. Spec B repeats the configuration and the ten seeds of Table 2. Spec A is the maximum-entropy backward policy of Mohammadpour et al. (2024) with a per-environment step penalty, as in the

GFN preset of Appendix H. It sets $\log P_B(s \mid s') = \log n(s) - \log n(s')$, where a learned head $\log n$ regresses toward the log-sum-exp of $\log n$ over the parents of s' and is clipped to a fixed range. The true path count is infinite on a cyclic graph, so this head is a clipped estimate that moves during training, and spec A acts as a perturbed backward policy that shifts over training. SubTB matches or exceeds one-step TD under both specifications on all five deep tasks.

Table 6: Success rate on the five deep tasks, mean $\pm$ SD over three seeds for spec A and ten seeds for spec B. Bold marks the highest value on each task. A bare entry takes the same value on every seed.

	SubTB (GFN)		One-step TD (SoftDQN)	
Task	Spec A	Spec B	Spec A	Spec B
iron	95.4 ± 1.9	$\mathbf{99.4 \pm 0.8}$	0.2 ± 0.3	87.4 ± 12.3
diamond	98.2 ± 1.6	$\mathbf{99.9 \pm 0.1}$	0.1 ± 0.1	99.1 ± 1.4
gold	90.1 ± 4.9	$\mathbf{90.6 \pm 11.4}$	0.0	4.6 ± 9.2
ender-eye	$\mathbf{53.2 \pm 6.1}$	7.9 ± 6.6	0.0	0.0
multi	66.4 ± 15.5	$\mathbf{87.2 \pm 10.9}$	13.8 ± 6.6	36.5 ± 17.7

F.2 Backup length

Table 7 sweeps the SoftDQN backup length over $n \in \{1, 4, 16, 64\}$ on iron and multi at three seeds each, with every other setting as in Table 2. The n-step target includes the soft entropy correction $-\lambda \sum_{j>i} \log \pi(a_j \mid s_j)$, where π is the softmax policy of the target network. Every $n > 1$ reaches 50% sooner than $n = 1$ and ends at zero. On iron, $n = 1$ holds its peak to the end.

G On-Policy Training Curves for the Main Comparison

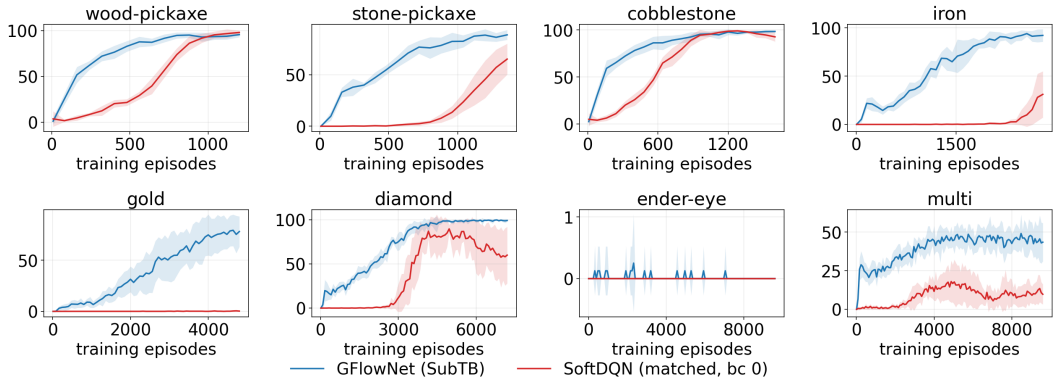

Figure 3: On-policy training success rate against training episodes, same layout as Figure 1.

H Reward-Maximizing Baselines

Table 8 reports PPO (Schulman et al., 2017), discrete SAC (Christodoulou, 2019), and SAC with a behavior-cloning term at weight 2.0 (SAC-bc2), each at its own tuned configuration over three seeds. The GFN and SoftDQN columns are their tuned presets. The GFN preset uses spec A of Appendix F.1, and the SoftDQN preset adds a behavior-cloning weight of 2.0 to the SoftDQN of Table 2. PPO reaches high success on several deep tasks, while the GFN holds the most strategic modes on every deep task.

Table 7: SoftDQN under n-step backups, three seeds per cell. Reaching 50% counts seeds whose evaluation success crosses 50%, and the epoch column is the median over those seeds. Peak and final are medians over all three seeds of the periodic evaluation during training.

Task	n	**Reaching** 50%	**Epoch at** 50%	**Peak**	**Final**
iron	1	3/3	310	84.4	**84.4**
iron	4	3/3	**90**	68.8	0.0
iron	16	1/3	90	48.4	0.0
iron	64	1/3	90	37.5	0.0
multi	1	2/3	515	50.0	0.0
multi	4	1/3	210	12.5	0.0
multi	16	1/3	**30**	18.8	0.0
multi	64	1/3	40	34.4	0.0

Table 8: All five experiments at their own tuned configurations, three seeds. Each cell is success rate followed by the pooled strategic-mode count in parentheses. Bold marks the highest success rate on each task. A dash marks a metric left undefined by zero successful rollouts.

Task	GFN	SoftDQN	PPO	SAC	SAC-bc2
wood-pickaxe	97.3 (1.0)	**100.0** (1.0)	**100.0** (1.0)	92.8 (1.0)	99.9 (1.0)
stone-pickaxe	96.5 (1.0)	99.5 (1.0)	**100.0** (1.0)	66.6 (1.0)	93.5 (1.0)
cobblestone	99.2 (1.0)	**100.0** (1.0)	**100.0** (1.0)	88.7 (1.0)	**100.0** (1.0)
iron	94.8 (6.3)	90.1 (3.3)	**99.3** (2.0)	0.5 (2.0)	35.0 (2.0)
diamond	96.6 (8.0)	99.0 (4.0)	**99.8** (2.0)	25.3 (2.0)	95.4 (2.0)
gold	**90.2** (38.3)	74.2 (20.3)	59.0 (2.3)	0.0 (–)	20.2 (3.3)
ender-eye	**54.4** (37.7)	28.3 (7.7)	18.6 (1.7)	0.0 (–)	0.0 (–)
multi	66.4 (12.7)	**82.7** (7.0)	75.4 (9.7)	5.5 (3.7)	36.6 (3.0)

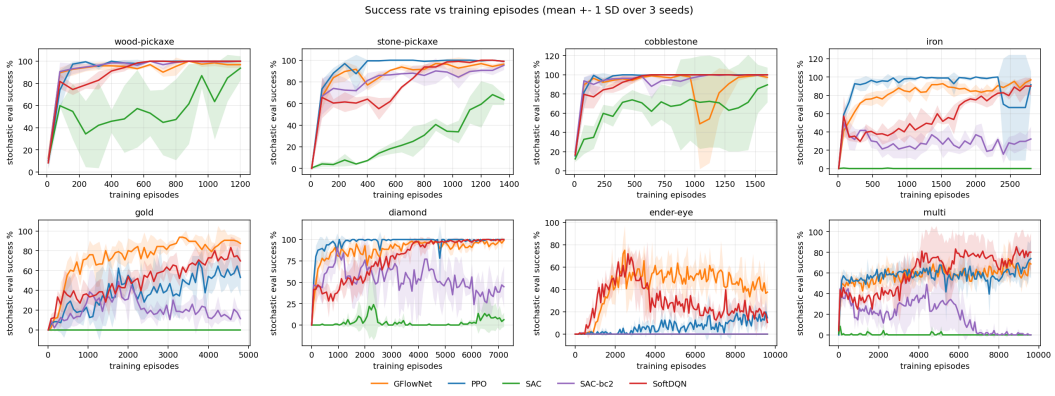

Figure 4: Evaluation success rate against training episodes for the five experiments of Table 8, mean over three seeds with a ± 1 SD band.

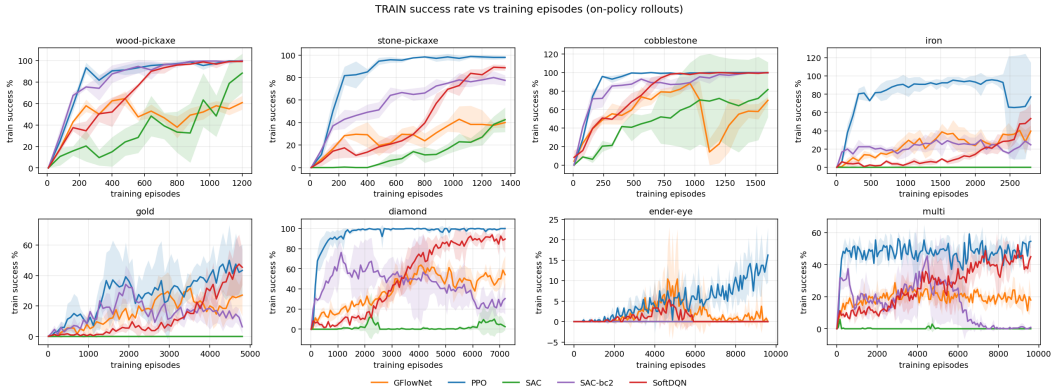

Figure 5: On-policy training success rate against training episodes, same layout as Figure 4.

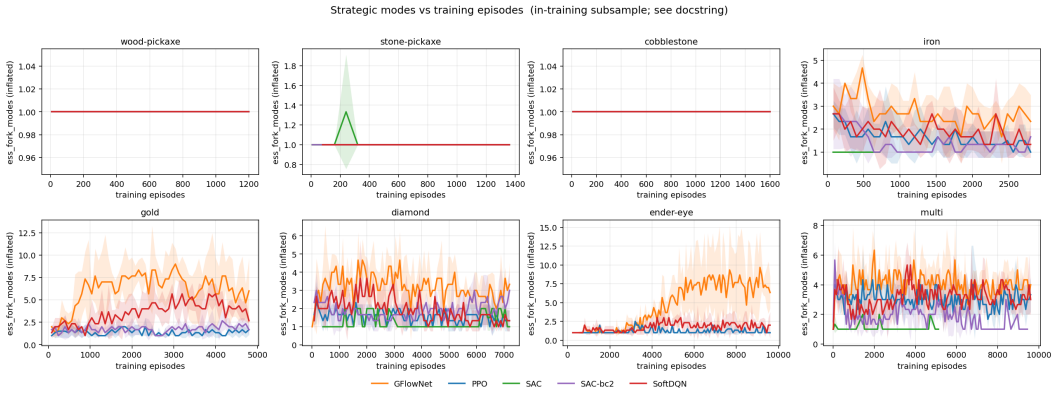

Figure 6: Strategic modes against training episodes, same layout as Figure 4 and computed as in Figure 2.