PUT YOUR SKILLS TO WORK: EXPLORATION VIA RANDOM WALKS IN SKILL SPACE

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-supervised pre-training in reinforcement learning aims to discover a diverse set of behaviours without extrinsic reward, which can then accelerate a wide range of downstream tasks. Recent skill discovery methods such as METRA and CSF learn skills as *directions* in a latent space and produce impressive behaviours in locomotion domains. However, the exploration of existing skill learning methods degrades in complex environments, and existing remedies add machinery on top of skill learning, such as an explicit skill tree or a learned guide policy. Leveraging the latent space that METRA/CSF already learn, we instead change only how skills are executed during training. Rather than committing to one skill per episode, we chain skills of a fixed horizon, resampling the latent direction at each boundary, so that a rollout performs a random walk whose steps are skill executions rather than primitive actions. Applying this to CSF gives Chained-CSF (CCSF), which leaves the base algorithm, its losses and its networks untouched and adds only the skill horizon as a hyperparameter. On a bottleneck maze, skill chaining increases state coverage from 51.6% to 97.8%. On Craftax-Classic, a Minecraft-like domain with a deep crafting tree, CCSF unlocks 8.9 of the 22 achievements per episode without task rewards, the highest we are aware of for an unsupervised RL method. Removing the survival pressures that end episodes early lets it climb further, obtaining an iron pickaxe (depth 6; 97.0% of episodes) and diamonds (depth 7; 11.9%), while a baseline without skill switching does not reliably collect stone (depth 3). Without any reward or additional learning, the skills learned during pre-training can be used to solve downstream tasks: goals specified as a change in the raw observation are converted into latent directions and commanded from the frozen policy, reaching every stage up to iron in over 90% of episodes and doing so several times faster than random skill selection.

1 INTRODUCTION

Unsupervised RL promises agents that learn a repertoire of behaviours from their own experience, before any task is specified. Skill discovery is one route, learning a set of distinguishable, temporally extended behaviours that can later be composed to solve tasks. Recent methods such as METRA (Park et al., 2024) and CSF (Zheng et al., 2025) learn skills as directions in a latent space and produce striking locomotion behaviours, but they have been evaluated almost entirely in open spaces, and their exploration deteriorates once the state space is not freely traversable (Figure 1). The irony is that a random walk, which has no learned structure at all, eventually reaches everywhere, but takes on the order of d^2 steps to travel a distance d. A skill of this kind travels far and fast, but only in the directions it was trained from. In this paper we put these skills to work as the steps of a random walk.

Most skill discovery methods are built on mutual information skill learning (MISL), which maximises the mutual information between a latent skill variable and the states it induces (Gregor et al., 2016; Eysenbach et al., 2018; Sharma et al., 2019). The objective places little pressure on exploration, since skills need only be distinguishable, not far apart (Park et al., 2024). METRA and CSF address this by tying skills to distance in a learned latent space, which is what makes their skills travel, but a skill that travels in a straight line still stops at the first wall. This limitation is not new, and prior methods address it by adding machinery on top of skill learning: EDL sidesteps it by sampling directly from an oracle state distribution (Campos et al., 2020), UPSIDE iteratively grows

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

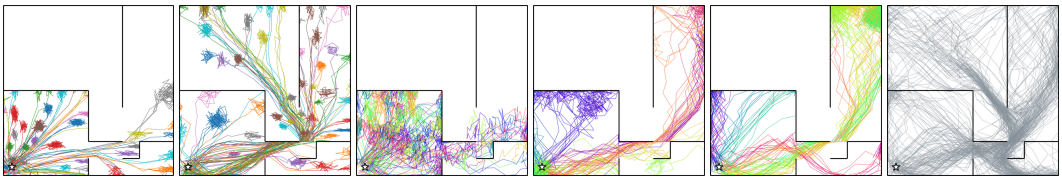


Figure 1: **A random walk on skills explores a maze.** For several skill learning methods, we visualize 100 rollouts, each starting from the bottom-left cell marked by a star, with state coverage statistics below each panel. Our approach (CCSF) achieves higher coverage than prior methods and differs from CSF by resampling skills every so often during a rollout; this is done when collecting data during training as well.

an explicit tree of skill segments (Kamienny et al., 2021), and DISCO-DANCE (Kim et al., 2023) iteratively selects a guide skill to pull other skills through the bottleneck.

In this work, we improve the ability of unsupervised skill learning algorithms to explore complex environments without adding such machinery. We build on METRA and CSF, keeping their objective and learnt latent space intact while changing only how skills are executed during pre-training. Instead of holding a single skill for the full episode and traversing the latent space in a straight line, we chain skills of a fixed horizon, resampling the skill, and hence the latent direction, at every switch. Because METRA and CSF skills spread out in diverse directions and travel far from any given state, this random walk in the latent space translates into a random walk over diverse, far-reaching behaviours in the environment. **Chaining puts these skills to work** through two mechanisms. *First*, each segment begins wherever the previous one ended, so skills are learned from states at the edge of what earlier skills reach, and these new skills carry later segments further and to more diverse places. As a side effect, the policy experiences more skills from many start states rather than only the states each skill led to, which makes the skills easier to command downstream. *Second*, because each step of the random walk is a skill held for a fixed horizon rather than a single action, the walk passes through bottlenecks that a random walk over primitive actions rarely crosses.

We study the effect of this single change in three settings:

- A didactic 2D maze with bottlenecks (Fig. 1) designed by prior work to stress-test exploration (Campos et al., 2020). Our method, Chained CSF (CCSF), raises coverage on the maze from 51.6% to 97.8% over CSF.
- Craftax-Classic (Matthews et al., 2024; Hafner, 2021), a Minecraft-like domain with a deep crafting tree, where CCSF unlocks 8.9 achievements per episode against 5.3 for the strongest prior unsupervised method. With survival pressures removed (*Peaceful* mode), it obtains iron pickaxes in 97.0% of episodes, whereas CSF does not reliably collect stone.
- Locomotion environments (Ant and HalfCheetah), following prior work (Eysenbach et al., 2018; Park et al., 2024; Zheng et al., 2025), which tests the ability to learn complex control in open-space environments. Here we show that CCSF can learn to chain skills more cleanly, at a cost of some loss in state space coverage.

In addition, the resulting skills are also easier to command downstream, reaching every stage of the Craftax-Classic technology tree up to iron in over 90% of episodes zero-shot and outperforming CSF in Ant multi-goal by improving its *staying time fraction* from 24.4% to 52.9%.

2 Related Work

Unsupervised RL includes methods that explore with intrinsic bonuses rather than learning skills (Pathak et al., 2017; Burda et al., 2018), discussed in Appendix A; here we focus on skill discovery. Executing skills in sequence has been considered before by Ji et al. (2026), though only analytically in tabular settings, and is distinct from skill chaining (Konidaris & Barto, 2009), which composes options backward from a goal.

Mutual information skill discovery. The dominant family maximises a variational bound on the mutual information between a latent skill and the states it induces. VIC (Gregor et al., 2016) and DIAYN (Eysenbach et al., 2018) train a discriminator $q_\phi(z \mid s)$ over visited states, VALOR (Achiam et al., 2018) conditions it on whole trajectories, DADS (Sharma et al., 2019) replaces it with a skill-conditioned dynamics model, and CIC (Laskin et al., 2022) uses a contrastive estimator over transitions with a particle entropy term, admitting continuous skill spaces. The recurring weakness is that the bound saturates at $\mathcal{H}(Z)$, so skills need only be distinguishable, not far apart, and static or spatially adjacent behaviours can be optimal (Park et al., 2024; Campos et al., 2020).

Metric-aware skill discovery. A second line ties discriminability to distance so the objective cannot be satisfied without moving. LSD (Park et al., 2022) and CSD (Park et al., 2023) impose Lipschitz constraints on the skill representation, and METRA (Park et al., 2024) maximises a Wasserstein dependency measure under a temporal-distance constraint, producing a latent space in which skills are directions. CSF (Zheng et al., 2025) recovers METRA's inner-product reward from successor features linear in the skill within the standard contrastive MISL framework. We build on CSF and modify only how its skills are executed.

Skill discovery in complex environments. Several methods target the failure of skill discovery where the state space is not freely traversable. EDL (Campos et al., 2020) obtains a target state distribution first, via state marginal matching or an oracle, and fits skills to it afterward. UPSIDE (Kamienny et al., 2021) grows an explicit tree of policies, each a directed segment reaching a frontier node followed by a diffusing segment that discovers new ones. DISCO-DANCE (Kim et al., 2023) nominates the skill with the greatest potential to reach unexplored regions as a guide and pulls the others toward it. Each shares our diagnosis that directed skills alone do not explore, but each adds explicit structure. We instead keep the base algorithm unchanged and alter only the execution schedule, which applies to any method whose skills are latent directions.

3 Preliminaries

3.1 Skill Discovery

We consider a Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, p, \rho_0)$ with state space $\mathcal{S}$, action space $\mathcal{A}$, transition kernel $p(s' \mid s, a)$ and initial state distribution ρ_0. No extrinsic reward is available during pre-training. A *skill* is a latent variable $z \in \mathcal{Z}$ drawn from a fixed prior $p(z)$, and the agent is a skill-conditioned policy $\pi(a \mid s, z)$. In standard training, z is sampled at the start of each episode and held fixed for the entire episode length T; we revisit this in Section 4.

Mutual information skill learning (MISL). MISL methods maximise the mutual information between the skill and the states it induces. Following CSF we take the transition form

$$\mathcal{I}(S, S'; Z) \;=\; \mathcal{H}(S, S') - \mathcal{H}(S, S' \mid Z), \tag{1}$$

which rewards skills that produce distinguishable *transitions* rather than distinguishable states. As Eq. 1 requires an intractable posterior, it is optimised through a variational or contrastive lower bound (Barber & Agakov, 2004), yielding an intrinsic reward that is large when the transition (s, s') identifies z. Methods differ in the estimator and in what is held fixed (Section 2).

METRA. METRA constrains the representation $\phi_\theta : \mathcal{S} \to \mathbb{R}^d$ so that latent distance lower-bounds temporal distance, and rewards latent displacement along the sampled skill direction:

$$\max_{\phi_\theta} \; \mathbb{E}\Big[\big(\phi_\theta(s') - \phi_\theta(s)\big)^\top z\Big] \quad \text{s.t.} \quad \big\|\phi_\theta(s) - \phi_\theta(s')\big\|_2 \le 1 \quad \forall\, (s, s') \text{ adjacent.} \tag{2}$$

CSF shows the Wasserstein machinery is unnecessary, as taking successor features linear in z and estimating Eq. 1 with contrastive learning recovers the same inner-product reward, placing METRA-style skills back inside MISL. We build on CSF throughout and describe it next.

3.2 Contrastive Successor Features (CSF)

CSF (Zheng et al., 2025) learns a skill-conditioned policy $\pi(a \mid s, z)$ and a state encoder $\phi_\theta : \mathcal{S} \to \mathbb{R}^d$ with skills drawn uniformly from $z \sim p(z) = \mathrm{Unif}(\mathbb{S}^{d-1})$, a uniform distribution over a

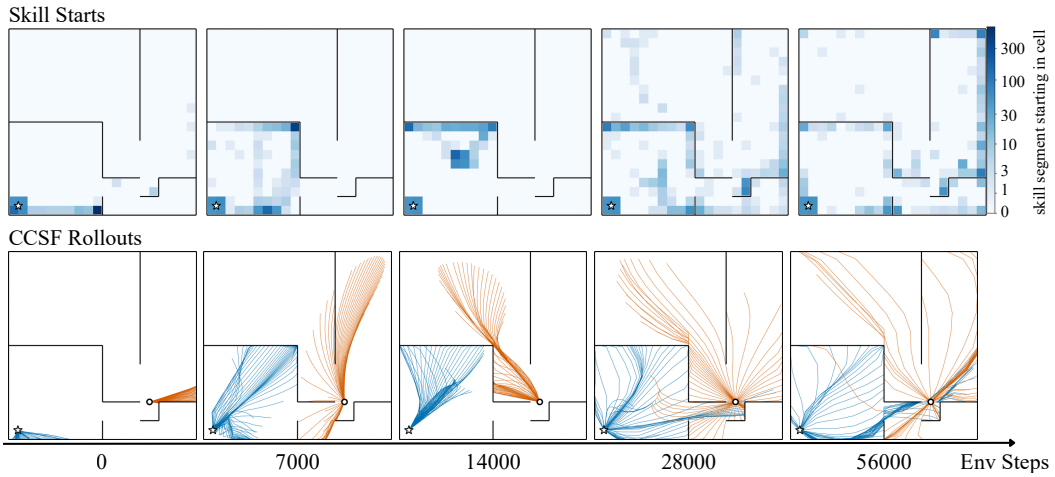


Figure 2: **Skills seed the states where the next skills begin.** Skill segment starts per cell (top) and rollouts of 60 evenly spaced skills from two probe states, over CCSF training on the bottleneck maze. A skill that spreads out from the start cell ends somewhere new, that endpoint becomes the start of the next segment, and the skills learned there spread out in turn. Segment starts therefore disperse across the maze over training, while CSF only ever begins a skill at the reset state.

d-dimensional unit hypersphere, so a skill is a direction. The intrinsic reward is the bilinear form

$$r_z(s, s') \;=\; \big(\phi_\theta(s') - \phi_\theta(s)\big)^\top z \;=\; \Delta\phi_\theta(s, s')^\top z, \tag{3}$$

so a skill z is rewarded for moving the latent state in the direction z. The encoder is trained with a contrastive lower bound on the mutual information in Eq. 1 (Barber & Agakov, 2004; Poole et al., 2019), a variant of InfoNCE (Oord et al., 2018) in which the positive is the skill that actually generated the transition and the negatives are skills drawn independently from the prior. With the negative term weighted by λ, the loss is

$$\mathcal{L}_\phi \;=\; -\,\mathbb{E}_{p(s,s',z)}\big[\Delta\phi_\theta^\top z\big] \;+\; \lambda\,\mathbb{E}_{p(s,s')}\Big[\log \mathbb{E}_{p(z')}\big[\exp\big(\Delta\phi_\theta^\top z'\big)\big]\Big], \tag{4}$$

which replaces the $\|\Delta\phi\| \le 1$ constraint of METRA with a soft penalty on the latent step size: the first term aligns $\Delta\phi$ with the executed skill, and the second term keeps $\|\Delta\phi\|$ bounded, so no dual variable or normalisation of ϕ is needed. The coefficient λ sets how tightly $\Delta\phi$ is bounded. The inner expectation over $p(z')$ can be estimated by sampling skills from the prior, or computed in closed form as the normalising constant of a von Mises–Fisher distribution, since $p(z)$ is uniform on the hypersphere (Mardia & Jupp, 2009).

Because Eq. 3 is linear in ϕ, the critic is a *successor feature* (Schaul et al., 2015; Barreto et al., 2017; Borsa et al., 2018) $\psi(s, a; z) \in \mathbb{R}^d$ predicting the discounted sum of future $\Delta\phi$, and the scalar action value is recovered by projection, $Q_z(s, a) = \psi(s, a; z)^\top z$.

4 Random Walks over Learned Skills

4.1 Chaining skills during training

To induce a random walk over skills, we modify CSF into CCSF by (1) resampling skills within an episode and (2) treating each segment boundary as a time limit for its skill. Nothing we modified from CSF depends on how π and ψ are optimised, and like CSF, our proposed CCSF could use any off-the-shelf RL algorithm. Refer to Appendix B for the algorithm.

Segmentation and sampling. Unmodified CSF holds a single z for an entire episode, so what a skill can express is limited to what is reachable from the initial-state distribution under one latent direction. We instead *chain* skills within an episode by resampling the skill every H timesteps, where H is the *skill horizon*. Each segment i holds a single skill $z_i \sim p(z)$, drawn i.i.d., for H

timesteps, and at the boundary a fresh skill is drawn and executed from wherever the previous one stopped. If the episode terminates within a segment, that segment is cut short, so no segment spans a reset. A rollout is therefore a chain $z_1 \to z_2 \to \cdots$ in which segment i begins wherever segment $i-1$ ended, and later skills are trained from states that earlier skills had to reach.

Per-segment bootstrapped value targets. A step t in segment i is rewarded under its own skill only, $r_t = \Delta\phi_\theta(s_t, s_{t+1})^\top z_i$, and the same pairing $(\Delta\phi_t, z_i)$ is the positive in the contrastive loss Eq. 4. A segment boundary ends the skill's time budget, so we truncate there instead of terminating. The reward is not used during training and we instead use a successor feature as mentioned in Section 3.2. Writing t_i for the step at which segment i begins, γ for the discount factor and $\bar{\psi}$ for a slowly updated target copy of ψ, an actor–critic learner (Haarnoja et al., 2018) forms the target

$$y_t \;=\; \Delta\phi_\theta(s_t, s_{t+1}) \;+\; \gamma\,(1-d_t)\,\bar{\psi}(s_{t+1}, a'; z_i), \qquad a' \sim \pi(\cdot \mid s_{t+1}, z_i), \tag{5}$$

where $d_t = \mathbf{1}_{s_{t+1} \text{ is terminal}}$. At the final step $t = t_i + H - 1$ the target bootstraps into the first state of segment $i+1$ under z_i rather than the newly drawn z_{i+1}, so the critic never has to model the resampling of z. Each segment therefore optimises the same objective as unmodified CSF and differs only in its initial-state distribution, and setting H to the episode length T recovers CSF exactly.

4.2 Boosting exploration in skill learning with random walks

Chaining changes nothing about what a skill optimises, and its benefit instead comes from using the skills to explore new states and learn further skills there, forming a virtuous cycle. We illustrate this on the bottleneck maze of Figure 1 (Section 5.1).

4.2.1 A virtuous cycle between exploration and skill learning

Under unmodified CSF every segment starts at $s_0 \sim \rho_0$, so skills are only ever learned from states that one latent direction reaches from the reset distribution. Once skills saturate that set, further training refines behaviour in a region the agent has already covered, since ϕ_θ is fit to transitions from that region alone. Chaining breaks this. A segment ends partway along what would otherwise be one long directed push, and the next sampled skill sets off from there in a different direction, so the chain reaches states no single held skill would visit and the initial-state distribution for skill learning becomes the chain's own occupancy rather than ρ_0. Skills are then learned *at* those states, so newly reached regions become training data instead of remaining unvisited, and those skills in turn carry later segments further out again. Exploration supplies the states at which skills are learned, and the skills learned there supply the reach for further exploration. Figure 2 shows this cycle on the bottleneck maze (details in Section 5.1).

A side effect for downstream use. Under METRA and CSF, each skill starts from ρ_0 and runs for the whole episode, so the policy only ever sees a skill at the states that same skill leads to. However, during downstream control a high-level controller or a goal command issues arbitrary skills from arbitrary states, many of which the policy never trained on. Chaining draws each segment's skill independently of where the segment starts, so the policy is trained on far more state–skill pairs, with every skill executed from many start states. This helps the skills generalise to the states where downstream control issues them, as we show in Section 5.3.

4.2.2 Chained skills traverse an easier graph

A random walk covers an environment eventually, but a narrow passage makes that take prohibitively long, since a walk on one side of it crosses only on the rare steps that happen to fall within the gap (Aleliunas et al., 1979). A chained rollout does not walk over primitive actions. Each of its steps is a skill held for H steps, and the skills CSF learns travel far and spread out in different directions from wherever they start, following whatever route the local geometry allows. The walk therefore moves on a coarser graph whose edges connect states an entire skill apart, and edges of this kind cross the gaps that a primitive walk rarely finds. Appendix D builds this graph on the bottleneck maze and compares it against the one induced by a persistent random walk over primitive actions at the same horizon.

5 EXPERIMENTS

We evaluate chaining in three settings. The bottleneck maze (Section 5.1) tests the idea directly, measuring the coverage chaining gains over unmodified CSF and verifying the two mechanisms of Section 4.2, the virtuous cycle and the easier graph. Craftax-Classic (Section 5.2) then asks how far chaining can go, in a high-dimensional environment whose technology tree demands long compositional dependencies. Finally, Ant (Section 5.3) checks that chaining retains the performance of CSF in the open-space locomotion setting and the effectiveness of skills for downstream control.

5.1 BOTTLENECK MAZE

Setup. The bottleneck maze of Figure 1 is a 10×10 continuous point maze that has rooms separated by walls with narrow gaps, and is used to evaluate the exploration capability of skill discovery methods (Campos et al., 2020; Kim et al., 2023). Every episode starts in the bottom-left cell and lasts $T = 50$ steps, and the action is a bounded 2-D displacement. We train every method for 3×10^6 environment steps on 3 seeds and measure *state coverage*, the percentage of a 20×20 grid of 400 buckets entered by at least one of 100 evaluation rollouts. Each method is rolled out under the protocol it was trained with, so CCSF resamples its skill every $H = 10$ steps while every baseline holds one skill for the whole episode, and the environment-step budget is identical across methods.

Baselines. We compare against five skill discovery methods in two groups. The MI-based methods are DIAYN (Eysenbach et al., 2018), with 70 discrete skills, DISCO-DANCE (Kim et al., 2023), which guides skills toward the states reached by the most exploratory skill and is designed specifically for bottlenecks, with 10 discrete skills grown to 100, and CIC (Laskin et al., 2022), with 64-dimensional continuous skills. The metric-based methods are METRA (Park et al., 2024) and CSF (Zheng et al., 2025), which together with CCSF use 2-dimensional skills on the unit circle. CSF is the method we modify, so the CSF/CCSF pair isolates chaining, as the two share every network, hyperparameter and update rule. Full details are in Appendix C.

Coverage. Figure 1 report the result. CCSF reaches $97.8 \pm 1.4\%$ coverage, the highest of all six methods and well clear of DISCO-DANCE ($63.2 \pm 14.5\%$), which is designed specifically to push skills through bottlenecks. The comparison that isolates our contribution is against unmodified CSF, where chaining raises coverage from $51.6 \pm 5.1\%$ to $97.8 \pm 1.4\%$ at the same environment-step budget, with no additional networks, losses or hyperparameters beyond H.

The virtuous cycle. To verify the mechanism of Section 4.2.1, we probe the skills at two states of our choosing, the start cell $(0.0, 0.0)$ and a point in the bottleneck corridor $(7.0, 1.5)$. At snapshots throughout CCSF training we execute 60 skills evenly spaced on the unit circle for one skill segment, $H = 10$ steps. As seen in Figure 2, over training the rollouts at both probes fan out and travel further, causing the start cells of skill segments to spread out with them. The two feed each other, since a skill from the start cell can end near the corridor, and the skills learned at the corridor then carry the following segment into rooms the start cell alone does not reach.

The graph traversed by chained skills. We also build the graph the agent traverses with skills and compare it against a persistent random walk at the same horizon. The skill graph spans the maze and links rooms across the narrow gaps, lifting coverage to $92.2 \pm 2.7\%$ against $67.4 \pm 6.1\%$ at the same budget, with the construction and figures in Appendix D.

5.2 CRAFTAX-CLASSIC

To further test the exploration capabilities of randomly chaining skills, we turn to a far more demanding environment. Craftax-Classic (Matthews et al., 2024) is a JAX reimplementation of Crafter (Hafner, 2021), an open world environment inspired by Minecraft in which an agent gathers resources, crafts tools, places objects and fights mobs. Its technology tree, as shown in Figure 3, is strictly ordered and heavily dependent. Depth in this tree is therefore a demanding measure of exploration, and on top of it the agent must stay alive, managing health, hunger, thirst and energy while hostile mobs attack. More details about Craftax-Classic are in Appendix E.

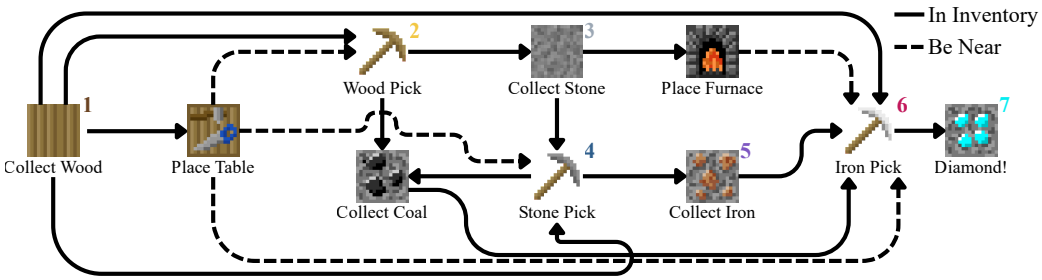


Figure 3: **Technology tree of Craftax-Classic.** Arrows show the dependencies of each item. The number at the top right of an item is its *depth*, its position along the longest chain of dependencies, which ends at the diamond.

Setup. All runs use the symbolic observation and the 17-action discrete space, with 256 parallel environments and 1×10^9 environment steps of training. We impose no episode-length cap beyond the environment's default of 10,000 steps, since agents die well before it. CCSF uses a skill horizon of $H = 32$. We report two metrics. The *achievement rate* of an achievement is the percentage of the parallel environments that unlock it at least once in an episode, and *achievements per episode* is the sum of these rates over the 22 achievements the environment defines, which together span every way of interacting with it, from gathering resources and crafting tools to eating, drinking, sleeping and defeating mobs. Further details are in Appendix E.1.

Baselines. We compare against four exploration methods that do not learn skills: ICM (Pathak et al., 2017), RND (Burda et al., 2018), E3B (Henaff et al., 2022) and C-TeC (Mohamed et al., 2026), with further discussion in Appendix A. *Random* is a random policy, establishing the floor of easier-to-obtain achievements. As Craftax-Classic is a standard benchmark, we take the baseline results from Mohamed et al. (2026).

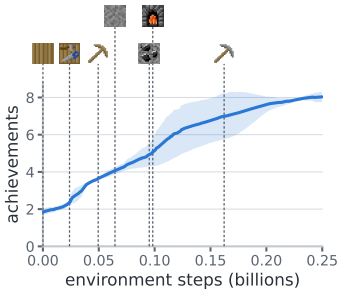


Figure 4: **Training curve of CCSF on Craftax-Classic.** Icons mark selected achievements when their rate first passes 10%.

Table 1: Achievements per episode after 1×10^9 steps.

Method	Achievements
ICM	1.3
RND	1.5
Random	2.3
E3B	3.7
C-TeC	5.3
CCSF (ours)	**8.9**

Results. Figure 4 shows part of CCSF's training curve of achievements per episode, annotating selected achievements at the point their rate first passes 10%, and Table 1 reports the final count after 1×10^9 steps. CCSF explores deep into the technology tree, reaching 8.9 achievements per episode, the highest we are aware of for an unsupervised RL method on Craftax-Classic, and a stone pickaxe (depth 4) in 37.7% of episodes without anything in its objective that rewards staying alive.

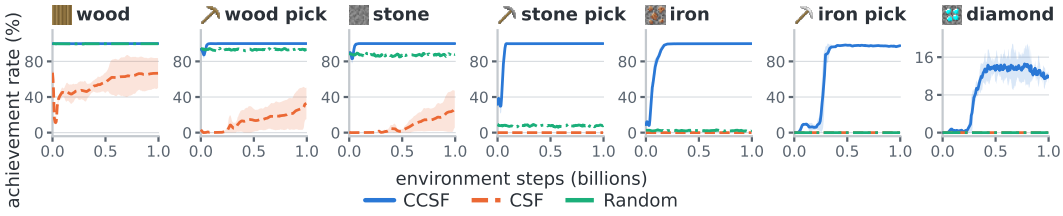


Figure 5: **CCSF reliably unlocks achievements during training.** Selected achievement rates for Craftax-Classic Peaceful during pre-training. CCSF is able to obtain diamonds and vastly outperforms CSF in exploring the deep technology tree.

5.2.1 Removing survival pressure

Without extrinsic rewards, nothing tells the agent that dying is undesirable, which results in deaths and episodes ending prematurely before any meaningful exploration of the crafting mechanic. To isolate exploration we construct a *Peaceful* variant that pins health, hunger, thirst and energy at their maximum and removes hostile mobs, leaving the technology tree untouched and the timeout ($T = 8192$ steps) as the only way an episode ends. As Figure 5 shows, CCSF climbs the tree in order: it is the only method to reach the iron pickaxe (depth 6), obtaining it in 97.0% of episodes, and it collects a diamond (depth 7) in 11.9%. A uniform-random policy reaches stone, since 8192 steps give a random walk time to stumble onto the shallow stages. CSF does not learn at this episode length in any of our runs, and only learns at all once episodes are shortened to $T = 256$, where it still does not reliably reach a stone pickaxe, likely for the same reason it stalls in the maze, as holding one latent direction for a long episode spends most of the episode pushing against what that direction cannot pass. Environment details and the full sweep over T are in Appendix F.4.

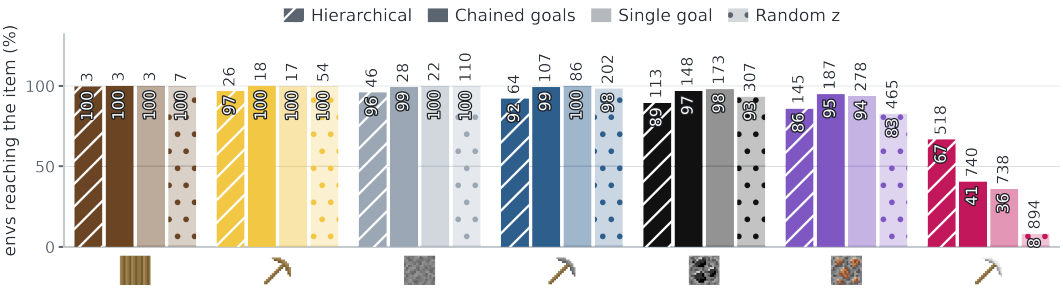


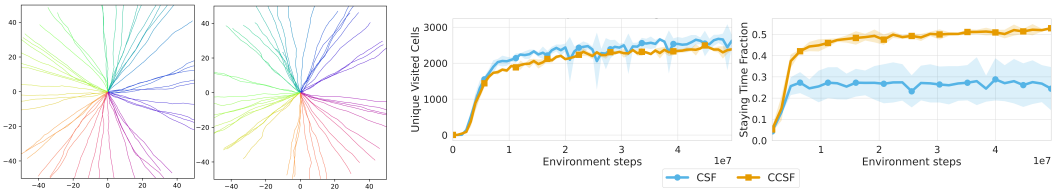
Figure 6: **Skills learnt are effective in solving downstream tasks.** After learning skills with CCSF, we compare four approaches to using those skills: a learned hierarchical policy, zero-shot chained goals, a zero-shot single goal, and random skill selection as a control. The number inside a bar is the percentage of the 256 environments that obtain the item within a 1024-step episode and the number above it is the median environment step at which it was first obtained. At the deeper stages all three directed approaches beat random skill selection in both reach and speed.

5.2.2 Downstream control with learnt skills

Strong exploration does not by itself mean the skills are useful afterwards, so we test the frozen skills of one seed downstream in three ways, over 256 environments and 1024-step episodes. Zero-shot control is possible because z is a *direction* in latent space (Park et al., 2024; Zheng et al., 2025), so for a goal state s_g, the current observation with the desired item added to the inventory, we set $z = (\phi_\theta(s_g) - \phi_\theta(s))/\|\phi_\theta(s_g) - \phi_\theta(s)\|$ and hand it to the frozen policy, recomputing it periodically and wandering on a random skill when a goal is not met. *Chained goals* commands the tech tree in order, advancing whenever a stage is met, while *single goal* spends a whole episode on one item with nothing upstream to build on. We also train a PPO high level over the same frozen skills, rewarded once per stage the first time it is unlocked in an episode, and compare all three against *random* z, which runs the same machinery with uninformative skills. As Figure 6 shows, the zero-shot chain reaches iron in 94.9% of episodes and an iron pickaxe in 40.6% with no learning at all, against 82.8% and 8.2% for random z, and the hierarchical policy, trained in under five minutes, raises the iron pickaxe to 66.8% and reaches it faster. Diamonds remain out of reach for every approach, at roughly 1%. The skills are therefore usable downstream, beating an undirected walk over the same skills in both reach and speed. The full protocol and further results are in Appendix F.5.

5.3 Locomotion Tasks

METRA, CSF and much of the unsupervised RL literature are evaluated on open-space locomotion, where nothing blocks a skill held for a whole episode. We follow the setup of METRA and CSF on Ant and HalfCheetah, with 2-dimensional skills as in CSF, and CCSF holding each skill for $H = 25$ steps, with further details in Appendix G.



CSF CCSF State Space Coverage AntMultiGoal

Figure 7: CCSF maintains the performance of CSF with slightly smaller state space coverage. The left pair shows CSF and CCSF rollouts under a fixed skill, while the two plots on the right give state space coverage over training and *staying time fraction* for zero-shot Ant MultiGoals. The shaded area for the two plots on the right indicate 1 std across 3 seeds.

Results. Figure 7 reports state space coverage over training and zero-shot *staying time fraction* (proportion of the episode horizon T that the agent spends near assigned goals) on Ant MultiGoals. CCSF covers slightly less of the state space than CSF, which is the expected cost of turning every H steps, but achieves a substantially stronger zero-shot *staying time fraction* of $52.9\% \pm 0.6\%$ compared to $24.4\% \pm 10.2\%$. This supports the claim of Section 4.2.1, that chaining improves generalisation and with it downstream performance. This improvement can likely be attributed to cleaner skill chaining, as shown qualitatively in Appendix G. Further results and analysis, as well as results for HalfCheetah can be found there.

5.4 Ablations

We ablate the two components of chaining on the bottleneck maze, resampling the skill within an episode and treating each segment boundary as a time limit, with details in Appendix H.

Chaining at evaluation versus in training. Re-evaluating every trained baseline on the maze with its skill resampled every $H = 10$ steps raises coverage for every method, by $+15.9$ to $+30.6$ points, so part of what chaining gives is available to any skill policy at deployment. The best of them still reaches only $80.1 \pm 15.3\%$ against CCSF's $97.8 \pm 1.4\%$, so chaining at deployment does not substitute for training under it.

Truncating versus terminating at segment boundaries. Truncating at a boundary, so the critic bootstraps through the switch, rather than terminating and zeroing the value there, is better in two different ways (Table 12). Chained, it reaches $97.8 \pm 1.4\%$ against $94.7 \pm 0.7\%$. Held for a whole episode, where chaining plays no part, it reaches $71.6 \pm 6.5\%$ against $55.8 \pm 2.2\%$, so bootstrapping produces better individual skills while terminating is only competitive when the skills are chained. Its one cost is speed, as it trails terminating for the first 5×10^4 frames of training before passing it.

6 Conclusion

We showed that the exploration of latent-directional skill discovery methods such as CSF can be substantially improved by chaining skills into a random walk, without altering the objective or losses. This simple change yields large coverage gains in environments more complex than the locomotion domains these methods are usually evaluated on, and the resulting skills remain useable zero-shot.

Limitations and future work. Our support for the graph argument is behavioural rather than formal, and characterising the induced transition structure, along with variable-horizon or Lévy-type chaining, is the most direct follow-up. CCSF does not obtain diamonds reliably, and we have not attempted unmodified Craftax-Classic, where survival pressures return.

AI USE STATEMENT

In this work, we used generative AI tools to implement parts of our method and experimental pipeline. We have not used generative AI tools to generate synthetic data sets, develop theoretical models or conceptual frameworks, formulate mathematical claims, provide ingredients for or assist in writing proofs, propose or refine hypotheses, design research methodology or experiments, or interpret results. Assisting with translation, cleaning and reformatting datasets, and qualitative or thematic data analysis are not applicable to this work. Additionally, we used generative AI tools to create and refine plotting code for the figures reporting our experiments, to edit software code and help us understand existing codebases we build on, to identify and summarize relevant literature during an initial survey, to suggest a structure for the paper, and to draft and edit parts of the paper for readability.

We have reviewed all AI-assisted work. Every line of AI-generated or AI-edited code was read by the authors and tested before use, and all reported results were produced by code the authors verified. Figures were checked against the underlying experimental logs. All literature identified with AI assistance was located and read by the authors before being cited, and all AI-edited text was revised by the authors to ensure it accurately reflects our methods and findings. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We have made efforts to ensure our results are reproducible. The full algorithm is given in Appendix B, and Appendices C, E.1, F.3 and G provide the implementation details, hyperparameters, and environment settings needed to reproduce our experiments. For the CSF experiments, we additionally report the random seeds and the hyperparameter sweeps performed in Appendix F.4. We will release our code upon publication.

REFERENCES

Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. Variational option discovery algorithms. *arXiv preprint arXiv:1807.10299*, 2018.

Romas Aleliunas, Richard M Karp, Richard J Lipton, László Lovász, and Charles Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pp. 218–223. IEEE Computer Society, 1979.

David Barber and Felix Agakov. The im algorithm: a variational approach to information maximization. *Advances in neural information processing systems*, 16(320):201, 2004.

André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado Van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Diana Borsa, André Barreto, John Quan, Daniel Mankowitz, Rémi Munos, Hado Van Hasselt, David Silver, and Tom Schaul. Universal successor features approximators. *arXiv preprint arXiv:1812.07626*, 2018.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

Víctor Campos, Alexander Trott, Caiming Xiong, Richard Socher, Xavier Giró-i Nieto, and Jordi Torres. Explore, discover and learn: Unsupervised discovery of state-covering skills. In *International conference on machine learning*, pp. 1317–1327. PMLR, 2020.

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

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control. *arXiv preprint arXiv:1611.07507*, 2016.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. Pmlr, 2018.

Danijar Hafner. Benchmarking the spectrum of agent capabilities. *arXiv preprint arXiv:2109.06780*, 2021.

Mikael Henaff, Roberta Raileanu, Minqi Jiang, and Tim Rocktäschel. Exploration via elliptical episodic bonuses. *Advances in Neural Information Processing Systems*, 35:37631–37646, 2022.

Catherine Ji, Vivek Myers, Sergey Levine, and Benjamin Eysenbach. The geometry of empowerment. August 2026. URL `vivekmyers.github.io/empowerment-geometry-site/static/pdf/geometry_empowerment.pdf`.

Pierre-Alexandre Kamienny, Jean Tarbouriech, Sylvain Lamprier, Alessandro Lazaric, and Ludovic Denoyer. Direct then diffuse: Incremental unsupervised skill discovery for state covering and goal reaching. *arXiv preprint arXiv:2110.14457*, 2021.

Hyunseung Kim, Byung Kun Lee, Hojoon Lee, Dongyoon Hwang, Sejik Park, Kyushik Min, and Jaegul Choo. Learning to discover skills through guidance. *Advances in Neural Information Processing Systems*, 36:28226–28254, 2023.

George Konidaris and Andrew Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. *Advances in neural information processing systems*, 22, 2009.

Michael Laskin, Hao Liu, Xue Bin Peng, Denis Yarats, MA NYU, Aravind Rajeswaran, and Pieter Abbeel. Contrastive intrinsic control for unsupervised reinforcement learning. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, pp. 34478–34491, 2022.

Hao Liu and Pieter Abbeel. Aps: Active pretraining with successor features. In *International Conference on Machine Learning*, pp. 6736–6747. PMLR, 2021a.

Hao Liu and Pieter Abbeel. Behavior from the void: Unsupervised active pre-training. *Advances in Neural Information Processing Systems*, 34:18459–18473, 2021b.

Kanti V Mardia and Peter E Jupp. *Directional statistics*. John Wiley & Sons, 2009.

Michael Matthews, Michael Beukman, Benjamin Ellis, Mikayel Samvelyan, Matthew Jackson, Samuel Coward, and Jakob Foerster. Craftax: A lightning-fast benchmark for open-ended reinforcement learning. *arXiv preprint arXiv:2402.16801*, 2024.

Faisal Mohamed, Catherine Ji, Benjamin Eysenbach, and Glen Berseth. Temporal representations for exploration: Learning complex exploratory behavior without extrinsic rewards. *arXiv preprint arXiv:2603.02008*, 2026.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Seohong Park, Jongwook Choi, Jaekyeom Kim, Honglak Lee, and Gunhee Kim. Lipschitz-constrained unsupervised skill discovery. In *International Conference on Learning Representations*, 2022.

Seohong Park, Kimin Lee, Youngwoon Lee, and Pieter Abbeel. Controllability-aware unsupervised skill discovery. *arXiv preprint arXiv:2302.05103*, 2023.

Seohong Park, Oleh Rybkin, and Sergey Levine. Metra: Scalable unsupervised rl with metric-aware abstraction. In *International Conference on Learning Representations*, volume 2024, pp. 18579–18603, 2024.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pp. 2778–2787. PMLR, 2017.

Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. In *International conference on machine learning*, pp. 5062–5071. PMLR, 2019.

Ben Poole, Sherjil Ozair, Aaron Van Den Oord, Alex Alemi, and George Tucker. On variational bounds of mutual information. In *International conference on machine learning*, pp. 5171–5180. PMLR, 2019.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pp. 1312–1320. PMLR, 2015.

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

Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. *arXiv preprint arXiv:1907.01657*, 2019.

Chongyi Zheng, Jens Tuyls, Joanne Peng, and Benjamin Eysenbach. Can a misl fly? analysis and ingredients for mutual information skill learning. In *International Conference on Learning Representations*, volume 2025, pp. 31876–31903, 2025.

APPENDIX

A ADDITIONAL RELATED WORK

Exploration bonuses in unsupervised RL. A parallel branch of unsupervised RL pursues coverage without learning a skill set. Curiosity-based methods reward prediction error under a learned dynamics model: ICM Pathak et al. (2017) uses an inverse-model feature space to avoid rewarding unpredictable but irrelevant transitions, Disagreement Pathak et al. (2019) uses ensemble variance, and RND Burda et al. (2018) uses the error of a network predicting a fixed random target, which is robust to stochastic dynamics. Entropy-based methods instead maximise the entropy of the state distribution directly, estimated by particle-based nearest-neighbour bounds in APT Liu & Abbeel (2021b), APS Liu & Abbeel (2021a), the last combining successor features with such a bonus to recover both coverage and skills. A third group makes the bonus episodic, which matters in procedurally generated worlds where novelty accumulated over all of training says little about the current level. E3B Henaff et al. (2022) scores a state by an elliptical bonus over the inverse-dynamics features already seen in the episode, generalising count-based exploration to continuous features. C-TeC Mohamed et al. (2026) instead learns temporal contrastive representations, whose critic scores how likely a future state is to follow a given state and action, and rewards the agent for state-actions that score low, that is, those whose futures the representations find improbable. These bonuses are complementary to our contribution: they modify the intrinsic reward, whereas we modify the execution schedule, and the two could in principle be combined.

B ALGORITHM FOR CHAINING SKILL LEARNING

Algorithm summary. Algorithm 1 summarises the steps for CCSF.

Algorithm 1 CSF with chained skill sampling (one iteration)

1: **Collect:** step $\pi(\cdot \mid s, z)$ in each parallel environment; when the segment timer reaches H (or the episode ends), reset the timer and draw $z \sim \mathrm{Unif}(\mathbb{S}^{d-1})$.
2: Store $(s_t, a_t, s_{t+1}, z_t, d_t)$ with $d_t = \mathbf{1}_{\text{s}_{\text{t+1}} \text{ is terminal}}$.
3: **for** each gradient step **do**
4: Draw a batch of transitions and N negative skills $z_j \sim p(z)$.
5: Update ϕ_θ on the contrastive loss Eq.4.
6: Recompute $\Delta\phi_\theta$ under the updated ϕ_θ.
7: Update ψ on the target (Eq.5), bootstrapping under the segment's own skill z_t.
8: Update $\pi(\cdot \mid s, z)$ against $Q_z(s, \cdot) = \psi(s, \cdot; z)^\top z$.
9: **end for**

C BOTTLENECK MAZE EXPERIMENTAL DETAILS

Environment. The maze is a 10×10 grid of unit cells on the continuous domain $[-0.5, 9.5]^2$, with every wall on the half-integer lattice. The bottom-left 5×5 block is an open room whose only exit is a one-cell gap at $x = 4.5$, $y \in [0.5, 1.5]$. The rest of the maze is a set of loosely connected rooms. The observation is the position (x, y) and the action is a displacement $(\Delta x, \Delta y) \in [-1, 1]^2$. A step that would cross a wall stops just short of it and slides the remaining perpendicular component along the wall, for up to three collisions. Episodes start uniformly in the origin cell, last $T = 50$ steps and carry no extrinsic reward.

Training. All six methods share one implementation of the training loop, the replay buffer, the SAC backbone, the evaluation protocol and the coverage metric, and differ only in the intrinsic objective. Table 2 lists the shared settings. Where the original works disagree, on learning rate, entropy coefficient and batch size, each method keeps the setting of its original work. Skill spaces also follow the original works: 70 one-hot skills for DIAYN; 10 one-hot skills for DISCO-DANCE, grown by 5 at each extension up to 100 (every seed reaches 100); $z \sim \mathrm{Unif}(\mathbb{S}^1)$ for METRA, CSF

and CCSF; and $z \sim \mathrm{Unif}(0,1)^{64}$ for CIC. Each method draws one skill per episode, except DISCO-DANCE, which redraws every 25 steps as in its release, and CCSF, which redraws every H steps. METRA uses a dual constraint with $\lambda_0 = 30$ and slack 10^{-3}. CSF and CCSF use $\xi = 5$ and 256 negative skills in the contrastive term. CCSF adds exactly one hyperparameter, the skill horizon $H = 10$. Everything else, including the random seed and the number of gradient steps, is the same as CSF.

Table 2: Training hyperparameters on the bottleneck maze. Rows that span all three columns are shared by every method; trunk and head apply to both the policy and the critics.

	DIAYN, DISCO-DANCE	CIC	METRA, CSF, CCSF
Trunk	`Linear`(128)–`LayerNorm`–`Tanh`		
Head	`Linear`(128)–`ReLU`		
Critics	twin Q, target $\tau = 5 \times 10^{-3}$		
Discount γ	0.99		
Optimiser	Adam		
Replay buffer	10^6, 1-step returns		
Update schedule	1 gradient step / 2 env. steps		
Random seed steps	4000		
Environment steps	3×10^6		
Training seeds	3 (101–103)		
LR (actor, critic)	3×10^{-4}	10^{-4}	10^{-4}
LR (representation)	3×10^{-4}	10^{-4}	10^{-4}
Entropy coef. α	fixed 0.2	auto, init 0.01	auto, init 0.01
Target entropy	—	$-\lvert\mathcal{A}\rvert$	$-\lvert\mathcal{A}\rvert/2$
Batch size	64	1024	256

Evaluation. Coverage divides $[-0.5, 9.5]^2$ into a 20×20 grid of 400 buckets of side 0.5. Every wall lies on a bucket boundary, so no bucket straddles one. A bucket counts as covered if any rollout passes through it. Each transition is linearly interpolated at 30 points, so a bucket crossed within a single step still counts. Each evaluation runs 100 rollouts of 50 steps. Each rollout starts from a skill drawn from the method's own prior and samples actions from the policy. Each method is rolled out under the protocol it was trained with, so CCSF redraws its skill every $H = 10$ steps and every other method holds one skill for the episode. We evaluate the 3×10^6-step snapshot under 3 evaluation seeds, average over evaluation seeds, and report the mean and standard deviation over the 3 training seeds.

For the probes of Figure 2 we use snapshots of training seed 101 taken during the first 5.6×10^4 environment steps. The agent is placed at the probe state and runs 60 skills evenly spaced on the unit circle for $H = 10$ steps each, acting with the policy mean. The segment-start heatmap counts, per bucket, the skill segments that begin there over 200 episodes of the same snapshot, with the policy mean and skills redrawn from the prior every H steps.

D THE GRAPH TRAVERSED BY CHAINED SKILLS

Construction. We use the CCSF snapshot of training seed 101 at 3×10^6 environment steps and act with the policy mean. We roll out 5000 environment steps from the start distribution, which is 100 episodes of $T = 50$ steps with the skill redrawn from the prior every $H = 10$ steps, and record the 500 states at which a segment begins. Segment starts that lie pairwise within one bucket width of each other, with no wall between them, are merged into a single node. The node is placed at the centre of the 20×20 bucket containing the group's mean and sized by how many segment starts it holds. From every node we then execute 60 skills evenly spaced on the unit circle for $H = 10$ steps, as in the probes of Appendix C. Each rollout adds an edge to the nearest node within one bucket width of its endpoint, provided no wall separates the two. 89% of rollouts end on a node, including the 5% that return to the node they started from and add no edge. The remaining 11% end too far from any node.

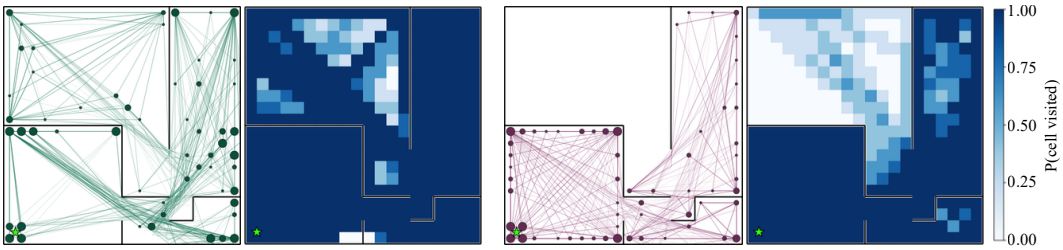


Figure 8: The graph traversed by *(Left)* chained skills compared with *(Right)* a persistent random walk over primitive actions, each beside the cells its rollouts visit. Chained skills link every room across the narrow gaps and cover 92.2% of the maze. The persistent walk covers 67.4%.

Baseline behaviour. The same construction is applied to a persistent random walk, which holds the largest admissible action in one of the same 60 directions open-loop for H steps. Its collisions are resolved by the same solver, so it slides along a wall rather than stopping at it.

Coverage. Coverage is the fraction of the 400 buckets swept within one budget of 5000 environment steps, counted as in Appendix C. Over 5 independent budgets it is $92.2 \pm 2.7\%$ for chained skills and $67.4 \pm 6.1\%$ for the persistent walk (mean $\pm$ standard deviation). The graphs are built from the first budget. The heatmaps of Figure 8 show, per bucket, the probability that it is swept within one budget.

E CRAFTAX-CLASSIC

Environment. Craftax-Classic (Matthews et al., 2024) reimplements Crafter (Hafner, 2021) in JAX on a procedurally generated 64×64 grid world. We use its symbolic observation, a vector of 1345 entries. The first 1323 encode the agent's 7×9 local view: each cell holds a one-hot over the 17 block types and four binary indicators for zombies, cows, skeletons and arrows. The remaining 22 are the inventory (12 item counts, each divided by 10), health, food, drink and energy (also divided by 10), a one-hot facing direction, the light level and a sleeping flag. The 17 discrete actions are a no-op, four moves, DO (mine, attack, eat or drink whatever the agent faces), SLEEP, four placements (stone, crafting table, furnace, sapling) and six crafting actions (wood, stone and iron pickaxe and sword).

The technology tree is strictly ordered, and an agent cannot reach a stage without having passed every earlier one in the same episode. Crafting an iron pickaxe, as shown in Figure 3, requires wood, coal and iron in the inventory while standing beside both a crafting table and a furnace, and the iron itself can only be mined with a stone pickaxe. On top of this the agent must stay alive: food, drink and energy fall over time and health drops when any of them is empty, zombies attack on grass (more often at night), skeletons shoot arrows in caves, and lava kills on contact. An episode ends when health reaches zero or at the time limit.

The environment defines 22 achievements, each counted at most once per episode: collecting wood, saplings, drink, stone, coal, iron and diamond; placing a table, furnace, plant and stone; making the wood, stone and iron pickaxe and sword; eating a cow and a plant; defeating a zombie and a skeleton; and waking up. No reward is used during pre-training. Appendix F.2 describes how achievement rates are computed.

E.1 TRAINING DETAILS

Learner. CCSF trains the CSF representation objective (Zheng et al., 2025) with a discrete-action SAC learner (Haarnoja et al., 2018; Christodoulou, 2019), and chains skills by drawing a new $z \sim \mathrm{Unif}(\mathbb{S}^{D-1})$ every $H = 32$ steps, regardless of the state. Craftax-Classic and its Peaceful variant (Appendix F) use the same learner and the same hyperparameters, listed in Table 3. The critic is a pair of successor-feature networks $\psi_{1,2}(s, \cdot, z) \in \mathbb{R}^{|\mathcal{A}| \times D}$ with $Q_i = \psi_i^\top z$. Its TD target for a transition (s, a, s', z) is $\phi(s') - \phi(s) + \gamma\, \mathbb{E}_{a' \sim \pi(\cdot|s',z)} \bar{\psi}(s', a', z)$, where $\bar{\psi}$ is whichever

Table 3: **CCSF hyperparameters**, shared by Craftax-Classic and Craftax-Classic Peaceful. $|\mathcal{A}| = 17$ on Craftax-Classic and 18 on Peaceful.

Hyperparameter	Value
Environment steps	10^9
Parallel environments	256
Rollout length	32 steps per environment
Skill dimension D	4
Skill prior	$\mathrm{Unif}(\mathbb{S}^{D-1})$
Skill horizon H	32
CSF log-partition weight ϵ	5
Representation ϕ	MLP, 3×1024, ReLU, linear output in $\mathbb{R}^D$
Actor $\pi(a \mid s, z)$	MLP on $[s, z]$, 3×1024, ReLU, categorical output
Critic	twin successor features $\psi(s, \cdot, z) \in \mathbb{R}^{\lvert\mathcal{A}\rvert \times D}$, MLP 3×1024, ReLU
Optimiser	Adam, $\epsilon = 10^{-5}$
Learning rate (ϕ, actor, critic, α)	10^{-4}
Gradient clipping	global norm 0.5, per network
Batch size	4096
Gradient steps per rollout	32
Replay buffer	32,768 steps per environment (8.4×10^6 transitions)
Learning starts	after 4096 steps per environment
Discount γ	0.99
Target network update	Polyak, $\tau = 0.005$
Initial temperature α	0.01
Target entropy	$0.3 \log \lvert\mathcal{A}\rvert$
Episode cap	8192 steps
Resets (Craftax-Classic only)	optimistic, 16 fresh worlds per step

target network has the lower $\bar{\psi}^\top z$ and the reward is recomputed with the current ϕ at every update. The actor minimises $\mathbb{E}_{a\sim\pi}[\alpha \log \pi(a \mid s, z) - \min_i Q_i(s, a, z)]$, computed exactly over the actions, and α is tuned towards a target entropy. The log-partition term of the CSF objective, $\log \mathbb{E}_{z'\sim\mathrm{Unif}(\mathbb{S}^{D-1})} \exp\big((\phi(s') - \phi(s))^\top z'\big)$, depends only on $\kappa = \|\phi(s') - \phi(s)\|$ and is computed in closed form rather than from sampled negatives: for $D = 4$ it equals $\log\big(2I_1(\kappa)/\kappa\big)$, with I_1 the modified Bessel function of the first kind.

Deaths and resets. Environments whose episode ends are reset within the same step using the optimistic resets of Matthews et al. (2024), which generate $256/16 = 16$ fresh worlds per step and hand them to the environments that finished. The reset replaces the terminal observation, so the transition on which an episode ends has no true successor. We keep it in the replay buffer, because it is the only record of what ending the episode costs, but give it zero weight in the representation loss, zero reward and no bootstrap, so the value of the action that ended the episode is learnt as zero. The end of a skill segment is not treated as terminal: its target bootstraps from s' with the same z. We cap episodes at 8,192 steps (the environment's default is 10,000). The cap is never reached, since CCSF episodes last about 250 steps at the end of training (Table 4).

Seeds and baselines. We train three seeds for 10^9 environment steps each. Table 4 reports each seed at the end of training. The baseline numbers in Table 1 are those reported by Mohamed et al. (2026). Our own uniform-random policy, run with the setup above, unlocks 2.34 achievements per episode, in agreement with their 2.3.

F Craftax-Classic Peaceful

F.1 Environment

Without extrinsic reward nothing tells the agent that dying is bad, and on Craftax-Classic a skill policy that has not learnt to eat, drink and sleep dies within a few hundred steps, long before it could walk far enough to climb the tree. Peaceful removes every way an episode can end except the time

Table 4: **CCSF on Craftax-Classic, per seed**, after 10^9 environment steps (mean over the last 5×10^7 steps). Achievements are per episode, out of 22. The other columns are achievement rates, the percentage of episodes that unlock the achievement.

Seed	Achievements	Episode length	Wood pickaxe	Place furnace	Stone pickaxe	Iron
0	8.67	243.5	94.6	71.8	36.5	0.24
1	8.67	249.2	94.3	76.2	38.0	0.59
2	9.57	253.3	95.4	76.1	38.7	0.57
Mean $\pm$ s.d.	8.97 ± 0.52	248.7 ± 4.9	94.8 ± 0.6	74.7 ± 2.5	37.7 ± 1.1	0.46 ± 0.20
Random	2.34	142.0	0.85	0.05	0.00	0.00

limit, and leaves the rest of the game unchanged. It is a subclass of the Craftax-Classic environment that overrides the step function:

- **Intrinsics.** After every step, health, food, drink and energy are reset to their maximum of 9 and the counters that drive their decay are reset to zero, so they never fall.
- **Hostile mobs.** The spawn probabilities of zombies and skeletons are set to zero, and every zombie, skeleton and arrow slot is held dead after every step.[1] Cows are unchanged.
- **Lava.** A move onto lava is undone: the agent keeps its new facing direction but not the new position.
- **A BREAK action.** Placed crafting tables, furnaces and saplings cannot be removed in the original game, so a policy that places them around itself is walled in for the rest of an episode, which now lasts thousands of steps. We add an 18th action, BREAK, which removes the table, furnace or plant the agent faces and refunds the materials placing it cost. On any other block it is a no-op. The other 17 actions are unchanged.

World generation, the day–night cycle, cows, plants and the whole technology tree are unchanged, and so is the observation. The hostile-mob channels and the four intrinsics are simply constant. Every episode lasts exactly $T = 8192$ steps and starts in a newly generated world.

Achievements. We keep Craftax's 22 achievements unchanged, but three of them are unreachable in Peaceful: defeating a zombie and defeating a skeleton, because there is nothing to fight, and waking up, because the agent only falls asleep when its energy is below the maximum. At most 19 achievements can therefore be unlocked per episode. One more, collecting drink, is reachable but changes nothing the agent observes, since drinking does not consume the water block and drink is held at its maximum. A random policy unlocks it in 92% of episodes.

Resets. With deaths removed, all 256 environments end their episodes on the same step. Optimistic resets would then copy the same 16 fresh worlds across all 256 environments. We instead reset all environments together, outside the rollout, at the rollout boundary where the episode ends (T is a multiple of the rollout length), giving every environment its own world. This also keeps the true final observation of each episode.

F.2 HOW ACHIEVEMENT RATES ARE COMPUTED

During pre-training (Figures 4, 5 and 9, and Tables 1, 4, 5 and 7), an achievement's rate is the percentage of training episodes in which it was unlocked at least once. These are the episodes generated by the behaviour policy itself: actions sampled from $\pi(a \mid s, z)$, with z redrawn uniformly at random every H steps for CCSF and once per episode for CSF. The rate therefore measures what a random sequence of skills reaches within one episode. It does not ask whether *some* skill could reach

[1] Zeroing the spawn probabilities alone is not enough. In the original implementation, world generation gives *unspawned* zombie slots a positive health, and the DO action rebuilds the mob mask from health for every slot. The first DO therefore brings an unspawned zombie to life at map position $(0, 0)$. It is despawned again on the same step unless the agent is near that corner.

Table 5: **CCSF on Craftax-Classic Peaceful, per seed**, after 10^9 environment steps (mean over the last 5×10^7 steps). Achievements are per episode, of which at most 19 are reachable. The other columns are achievement rates (%). Seed 1 is the checkpoint used downstream (Appendix F.5).

Seed	Achievements	Stone pickaxe	Iron	Iron pickaxe	Diamond
0	16.65	99.97	99.89	97.49	11.87
1	16.51	99.95	99.87	96.99	13.13
2	16.74	99.95	99.87	96.37	10.77
Mean $\pm$ s.d.	16.63 ± 0.12	99.96 ± 0.01	99.88 ± 0.01	96.95 ± 0.56	11.92 ± 1.18
Random	11.62	7.99	2.31	0.00	0.00

Table 6: **CSF sweep settings** on Craftax-Classic Peaceful. Samples per transition is the number of times each collected transition is drawn for a gradient step, on average: gradient steps $\times$ batch size / (rollout length $\times$ 256 environments).

	Skill held (steps)	Rollout (steps)	Gradient steps per rollout	Replay buffer (steps per env)	Learning starts (steps per env)	Samples per transition
CSF, $T \leq 2048$	T	T	$T/4$	$16\,T$	T	4
CSF, $T = 8192$	8192	32	32	32,768	4096	16
CCSF, $T = 8192$	32	32	32	32,768	4096	16

the achievement. Achievements per episode is the sum of the 22 rates divided by 100. Each logged value averages the episodes that ended in one rollout, which on Peaceful are all 256 environments at once. Curves are smoothed with a rolling median over 2×10^7 steps, then shown as the mean $\pm$ one standard deviation over seeds. Final values are means over the last 5×10^7 steps. A rate is per episode, so it depends on the episode length: a longer episode gives more steps in which to unlock an achievement.

Downstream (Figure 6 and Tables 8 and 9), the policy is frozen, actions are greedy and there is no training. Each arm runs one 1024-step episode in each of 256 newly generated worlds. The rate of an item is the percentage of these environments that come to hold it within the episode, and the step reported above each bar is the median, over the environments that reached the item, of the first step at which they did.

F.3 Training details

CCSF. CCSF uses the hyperparameters of Table 3 with the 18-action space and $T = 8192$, for 10^9 steps and three seeds. Table 5 reports each seed.

Random. A uniform-random policy over the 18 actions, run for 10^9 steps with the same $T = 8192$ and one seed.

F.4 CSF: how long a skill is held

CSF draws one skill per episode and holds it until the episode ends, so its skill horizon is the episode length T. That makes T the parameter that decides what CSF can learn, and we sweep it over $T \in \{128, 256, 512, 1024, 2048, 8192\}$. Table 6 lists the settings that change with T. Every other hyperparameter is as in Table 3. For $T \leq 2048$ a rollout is one full episode, followed by $T/4$ gradient steps. The $T = 8192$ runs use CCSF's configuration exactly and differ from CCSF only in the skill horizon (8192 against 32). They are therefore the matched comparison. We run three seeds at $T = 256$, the best setting, and at $T = 8192$, the matched one, and one seed at every other T.

Results. Table 7 and Figure 9 show that no setting of T comes close to CCSF, and none reaches the Random baseline. The best is $T = 256$, with 3.7 ± 0.9 achievements per episode over three seeds. It makes a wood pickaxe in 29% of episodes (the seeds range from 9% to 51%) and essentially never

Table 7: **CSF against CCSF and Random on Craftax-Classic Peaceful.** *First ep.* is achievements per episode in the first logged episodes, when the policy is still close to uniform (the actor's output layer is initialised near zero, and no gradient step is taken during each environment's first $\min(T, 4096)$ steps). *Final* is the mean over the last 5×10^7 steps of training, $\pm$ s.d. over seeds where there are three. The item columns are achievement rates (%), averaged over seeds. †Crashed at 7.1×10^8 steps.

			Achievements						
Method	T	Seeds	First ep.	Final	Wood	Wood pick.	Stone	Stone pick.	Iron
CSF	128†	1	1.47	1.22	34.0	0.2	0.0	0.0	0.0
CSF	256	3	2.64	3.72 ± 0.85	66.6	29.4	24.4	0.0	0.0
CSF	512	1	3.82	2.98	65.8	35.7	18.0	0.2	0.0
CSF	1024	1	5.08	1.26	50.5	0.1	0.0	0.0	0.0
CSF	2048	1	7.17	1.12	32.3	0.8	0.4	0.0	0.0
CSF	8192	3	11.68	1.22 ± 0.61	32.0	7.4	0.6	0.0	0.0
Random	8192	1	11.55	11.62	99.8	93.3	87.1	8.0	2.3
CCSF	8192	3	11.71	16.63 ± 0.12	100.0	100.0	100.0	100.0	99.9

a stone pickaxe. This is the setting shown as CSF in Figure 5. Its rates are over 256-step episodes and are therefore not directly comparable with the 8192-step curves beside it. Every other setting unlocks fewer than 3 achievements per episode, and none makes a stone pickaxe in more than 0.2% of episodes. At the matched length $T = 8192$, the three seeds end at 1.2 ± 0.6 achievements per episode and make a wood pickaxe in 0.2% to 19.5% of episodes.

Why Random beats CSF. The first-episode column of Table 7 separates the two effects. The first is episode length. The same near-uniform initial policy unlocks 1.5 achievements in a 128-step episode and 11.7 in an 8192-step one, so Random's 32-times-longer episodes account for much of its lead over CSF at $T = 256$. The second is training itself. Only at $T = 256$ does CSF training raise the count above its first-episode value ($2.6 \to 3.7$). At every other T, training lowers it. At the matched length $T = 8192$, CSF starts where Random is (11.7 against 11.6), all three seeds fall below 3 achievements per episode within 3×10^7 steps, and they end at 1.2 after 10^9 (Figure 9a). CSF training therefore produces a policy that explores less than a uniform one. We attribute this to holding one skill for the whole episode: the objective rewards a skill only for continuing to move ϕ along its single direction z, and nothing rewards the changes of direction that each further rung of the technology tree requires. Chaining supplies those changes for free.

F.5 Downstream control with learnt skills

Every downstream arm uses the seed-1 CCSF checkpoint at 10^9 steps, with ϕ and π frozen and greedy actions ($\arg\max_a \pi(a \mid s, z)$). Each arm runs one episode of 1024 steps, without resets, in each of 256 newly generated worlds. The three zero-shot arms share the same worlds and environment random keys, so their comparison is paired.

Goal states. For a target item, the goal state s_g is the current observation s with only that item's inventory entry raised to 1, encoded exactly as the environment encodes a count of one. Every other entry of s is left unchanged. The skill is $z = (\phi(s_g) - \phi(s))/\|\phi(s_g) - \phi(s)\|$, recomputed from the current observation every 16 steps and whenever the target changes.

Zero-shot arms.

- **Chained goals** walks the technology tree in order: wood → wood pickaxe → stone → stone pickaxe → coal → iron → iron pickaxe → diamond. Each stage is a precondition, met once the agent holds at least one of its item. A stage already met on arrival is skipped without spending a step, so the budget goes to whatever is still missing. If a stage is not met within 64 steps, the agent alternates 64 steps on a random z with 64 steps on the goal z until it is. A stuck stage is often positional (no iron within reach), and wandering moves the agent before it asks again. An item obtained while wandering still counts.

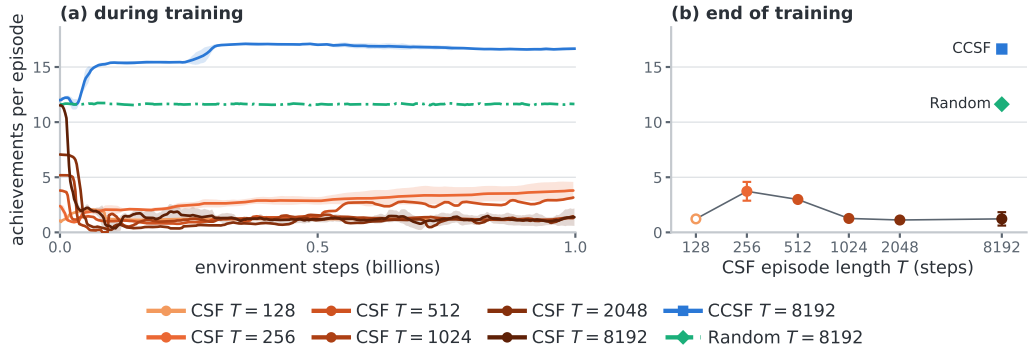


Figure 9: **No skill length rescues CSF on Craftax-Classic Peaceful. (a)** Achievements per episode during training for CSF at every episode length T, with CCSF and Random ($T = 8192$) for reference. Shaded bands are $\pm$ one s.d. over three seeds, for $T = 256$, $T = 8192$ and CCSF. **(b)** The value at the end of training against T. The hollow marker is $T = 128$, which crashed at 7.1×10^8 steps. Every CSF run starts at the level of an untrained policy for its T and, except at $T = 256$, ends below it. At $T = 8192$, the length CCSF and Random use, CSF starts level with Random and falls to 1.2.

- **Single goal** gives each item its own rollout, with that item as the only stage and the full 1024 steps for it. It therefore has nothing upstream to build on.
- **Random** z runs the chained arm's stage machinery and clock, but every z is drawn from $\mathrm{Unif}(\mathbb{S}^3)$ and carries no goal information. It is the control for both goal arms.

Hierarchical arm. Following the hierarchical experiments of Park et al. (2024) and Zheng et al. (2025), a high-level policy $\pi_{\mathrm{hi}}(k \mid s)$ picks a skill every 8 environment steps, and the frozen low level executes it. The high level is categorical over a fixed set of 24 unit skills: the 8 signed axes $\pm e_i$ and the 16 normalised corners $(\pm\frac{1}{2}, \pm\frac{1}{2}, \pm\frac{1}{2}, \pm\frac{1}{2})$. For $D = 4$ these are the vertices of the 24-cell, spaced evenly on $\mathbb{S}^3$ at 60° from their nearest neighbours. It is trained with PPO (Schulman et al., 2017) on the option-level MDP. An option's reward is the sum over its 8 steps of $+1$ the first time in the episode that each of *collect iron*, *make iron pickaxe* and *collect diamond* is unlocked. We reward only these three because random chaining over the same skills already unlocks 8 of the 10 achievements on the full path to the diamond, which would leave most of a whole-tree reward independent of the high level's choices. Training uses 256 environments, 1024-step episodes, 32 options per rollout, $\gamma = 0.99$ per option, GAE $\lambda = 0.95$, clipping 0.2, entropy coefficient 0.01, learning rate 10^{-4}, gradient clipping at norm 1, 8 epochs of 8 minibatches and a 3×1024 MLP. It runs for 5×10^7 environment steps, which takes about 4 minutes on one GPU. At evaluation the high level also acts greedily. It has no stages, so an item counts as reached when the agent first holds it.

Results. Table 8 gives the numbers behind Figure 6. Table 9 varies the hierarchical arm's reward, option length and rollout length. Rewarding all 10 achievements on the path to the diamond instead of the last three lowers the iron-pickaxe rate from 66.8% to 57.0%. Every variant still reaches the iron pickaxe in 57–67% of episodes, against 8.2% for random z.

G Experiments on Locomotion Tasks

G.1 Experimental Setup

All experiments on locomotion tasks (Ant and HalfCheetah) are identical to the implementation in CSF (Zheng et al., 2025) except for a few minor changes pertaining to CCSF. Notably, we introduce Skill horizon, H, to determine the interval at which the skills are resampled, and truncating on skill segment boundaries. For both environments, we train for approximately 50M timesteps with 3 seeds each. Other hyperparameters are listed in Table 10 and 11 for further details.

Table 8: **Downstream reach on Craftax-Classic Peaceful** (Figure 6). Each cell is the percentage of the 256 environments that obtained the item within 1024 steps, with the median first step among those that did in parentheses. All arms use the frozen seed-1 CCSF checkpoint.

	Zero-shot			Learnt
Item	Chained goals	Single goal	Random z	Hierarchical
Wood	100.0 (3)	100.0 (3)	99.6 (7)	99.6 (3)
Wood pickaxe	100.0 (18)	99.6 (17)	99.6 (54)	96.9 (26)
Stone	99.2 (28.5)	100.0 (22)	99.6 (110)	96.1 (46)
Stone pickaxe	99.2 (107)	100.0 (86)	98.4 (202.5)	92.2 (64.5)
Coal	96.9 (148)	98.0 (173)	93.4 (307)	89.5 (113)
Iron	94.9 (187)	93.8 (278)	82.8 (465)	85.9 (145)
Iron pickaxe	40.6 (740.5)	35.9 (738.5)	8.2 (894)	**66.8** (518)
Diamond	1.2 (987)	1.2 (803)	0.0 (–)	1.2 (740)

Table 9: **Hierarchical variants.** Reach (%) of the deepest items, and the median first step to the iron pickaxe, for high levels trained with different rewards, option lengths and rollout lengths. *Deep* rewards collect iron, make iron pickaxe and collect diamond. *Tech tree* rewards all 10 achievements on the path to the diamond. The first row is the arm in Figure 6.

Reward	Option length	Options per rollout	Stone pickaxe	Iron	Iron pickaxe	Diamond	Iron pickaxe median step
Deep	8	32	92.2	85.9	**66.8**	1.2	518
Tech tree	8	32	90.6	81.6	57.0	0.4	555
Tech tree	8	128	94.5	86.3	59.0	0.4	554
Tech tree	16	64	94.9	89.8	60.2	0.8	598.5
Random z (zero-shot control)			98.4	82.8	8.2	0.0	894

G.2 Ant Zero-Shot Goal Reaching

Setup. We compare CCSF with CSF for single-goal reaching and multi-goal reaching. For single-goal reaching, 50 goals are randomly sampled from $[-50, 50]$, and for multi-goal reaching, goals are randomly sampled every 50 timesteps from $[s_x - 7.5, s_x + 7.5] \times [s_y - 7.5, s_y + 7.5]$, where s_x and s_y represent the agent's coordinate. The skills are inferred every timestep from the goal by using $z = \frac{\phi(g)-\phi(s)}{\|\phi(g)-\phi(s)\|_2}$.

Results. Results can be found in Figure 10. We can see that CCSF is able to achieve a much higher *staying time fraction* for all evaluations compared to CSF. However, the difference is noticeably reduced for *success rate* in a single-goal setting, where it measures whether the agent reached the goal at any timestep. This result shows that both CSF and CCSF are capable of reaching the first goal states, but CSF is not able to continuously stay within the goal and transition to a new goal in the multi-goal setting. The reason for this is most likely due to CCSF learning to switch between different skills during training, allowing it to change directions smoothly to stay within a tight area. To test this, in Figure 11, we track the trajectory of the agent executing 48 random skills after it has been moving in one direction with the same first skill. From this, we can see that CCSF is able to quickly turn to all directions, while CSF's trajectories degrade.

G.3 HalfCheetah Environment

Setup. We evaluate zero-shot goal reaching over the pre-training phase for both CSF and CCSF. We also evaluate on both HalfCheetahGoal and HalfCheetahHurdle tasks Zheng et al. (2025) using a hierarchical policy. HalfCheetahGoal measures the ability for the agent to reach and stay in the goal while HalfCheetahHurdle measures the ability for the agent to jump over hurdles. We train the hierarchical policy for approximately 5M timesteps across 3 seeds.

Table 10: **Pre-training hyperparameters for CSF and CCSF** on Ant and HalfCheetah.

Hyperparameter	Value
Environment steps	4.96×10^7
Learning Algorithm	SAC (Haarnoja et al., 2018)
Parallel environments	8
Episode length T	200
Rollout length	200 steps per environment (1600 total)
Skill dimension d	2
Skill prior	$\mathrm{Unif}(\mathbb{S}^{d-1})$
Skill horizon H	25 (CSF: 200, i.e. $H = T$)
Segments per episode k	8 (CSF: 1)
CSF weight λ	5
Negatives per gradient step N	256
Optimiser	Adam
Learning rate (ϕ, ψ, π, α)	10^{-4}
Batch size	256
Gradient steps per rollout	50
Replay capacity	10^6
Learning starts	10^4 steps
Discount γ	0.99
Target network update	Polyak, $\tau = 0.005$
Initial temperature α	0.01
Automatic entropy tuning	yes

Table 11: **Hyperparameters for hierarchical control** on HalfCheetahGoal and HalfCheetahHurdle. Settings follow CSF (Zheng et al., 2025).

Hyperparameter	Value
Algorithm	PPO (Schulman et al., 2017)
Learning rate	10^{-4}
Skill horizon	25
Episode length	200
Parallel workers	8
Trajectories per rollout	64
Minibatch size	256
Gradient updates per rollout	10
State normalizer	yes
Parent and child policies	MLP, 2×1024, Tanh
Child policy frozen	yes

Results. During the pre-training phase of HalfCheetah environment, we notice that CCSF has a much lower state space coverage than CSF in Figure 12. This result confirms that CSF can cover more state space than CCSF by going far in one direction instead of moving in random directions. Due to this, for HalfCheetahGoal and HalfCheetahHurdle task, we can see in Figure 13 that CCSF underperforms or matches CSF in the return it receives. This is likely due to CSF learning to travel further during the pre-training phase than CCSF. However, Figure 14 also shows interesting results where its *staying rate* (the number of times the agent was within the goal at the end of the episode divided by the number of times the agent reached the goal at any timestep) is maxed out at 100% compared to CSF's value which is near 5%. This shows that while CSF can reach goals well, it hardly is able to stay within the goal, and is consistent with the Ant environment that CCSF allows better skill switching.

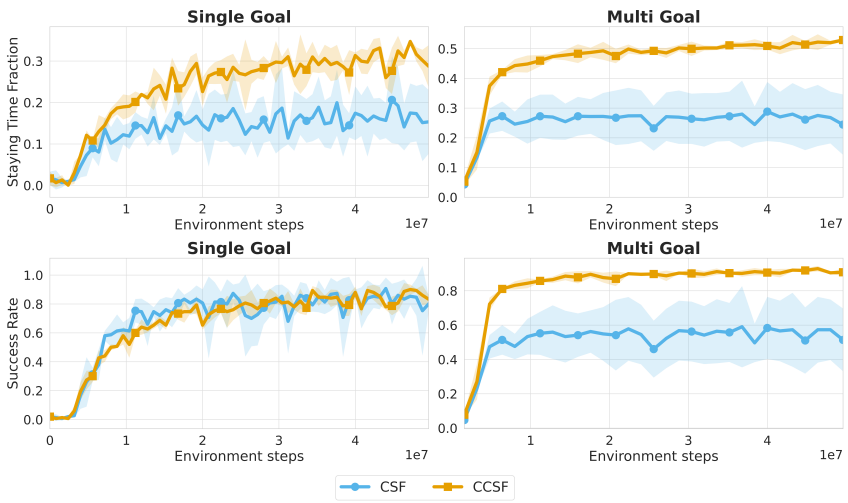


Figure 10: We show performance of CSF and CCSF in Ant single-goal and multi-goal tasks. CCSF outperforms CSF substantially, except for *Success Rate*, which indicates that CSF is incapable of staying within the goal reliably after reaching it. Multi-goal metric *(top-right)* shows *staying time fraction* instead of *return* as in prior work (Zheng et al., 2025), since it is simply *return* normalized by the episode length. The line represent the mean with the shaded area indicating 1 standard deviation across 3 seeds.

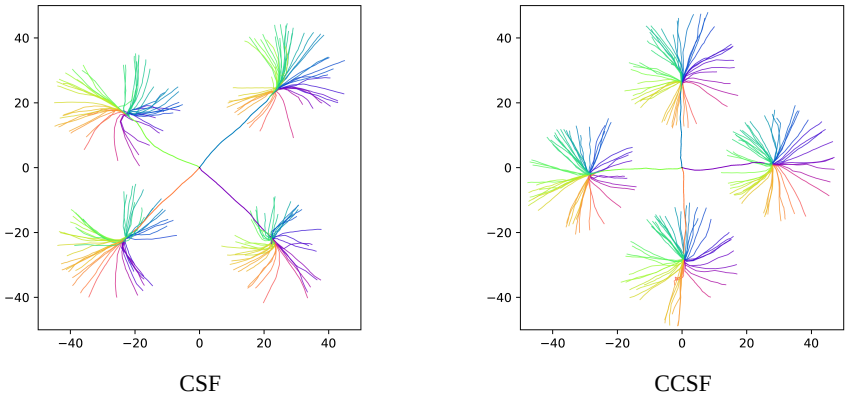


Figure 11: The plot shows the trajectories of the agent where the first skill is kept the same while the second skill is randomly sampled. This shows how well the agent is able to change direction with a different skill while moving. We can see that CCSF is able to cleanly turn to all directions while CSF struggles to change to certain directions.

H Ablation Details

H.1 Chaining at Evaluation versus in Training

Setup. Chaining is a property of the rollout protocol as well as of the objective, so it can be applied post hoc to a policy that never saw it. We take each final snapshot and re-run the evaluation of Appendix C with the skill resampled from the method's own prior every $H = 10$ steps instead of held for the full episode. Nothing is retrained and the environment-step budget is unchanged, and only the schedule on which z is drawn differs. CCSF appears twice, once as the method and once as the variant that terminates at a segment boundary instead of truncating, and both are also evaluated with one skill held for the whole episode. Each method's entry under the protocol it was trained with, held for the baselines and chained for the two CCSF rows, is computed by the same

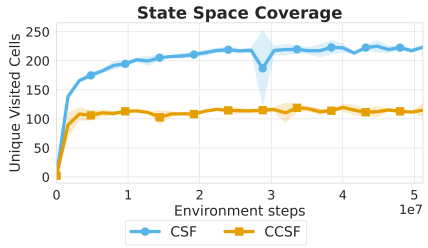


Figure 12: This plot shows the state space coverage of CSF and CCSF during the pre-training phase of HalfCheetah. CCSF underperforms CSF as it changes direction randomly instead of moving far away consistently. The line represents the mean with the shaded area indicating 1 standard deviation across 3 seeds.

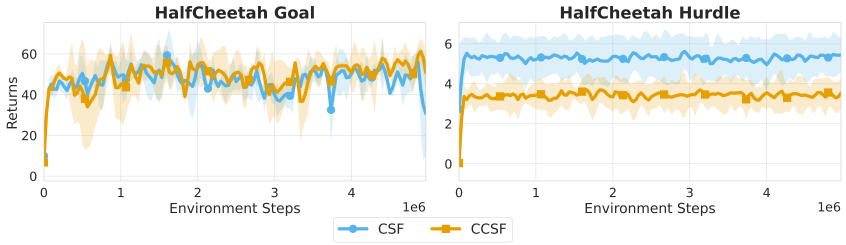


Figure 13: The plots show the return for both HalfCheetahGoal and HalfCheetahHurdle tasks. CCSF matches CSF for the HalfCheetahGoal but underperforms for HalfCheetahHurdle. The line represent the mean with the shaded area indicating 1 standard deviation across 3 seeds.

procedure as the main coverage table, so those entries reproduce the numbers reported there rather than re-estimating them.

Table 12: Coverage (%) on the bottleneck maze when the chained protocol is applied at evaluation time to methods that were not trained with it. Mean and standard deviation over 3 training seeds, each averaged over 3 evaluation seeds. Entries in the protocol each method was trained under are its main-text coverage.

Method	Held ($H = 50$)	Chained ($H = 10$)	Δ
DIAYN	40.4 ± 1.2	56.3 ± 1.3	$+15.9$
DISCO-DANCE	63.2 ± 14.5	80.1 ± 15.3	$+17.0$
CIC	39.8 ± 1.5	70.4 ± 9.3	$+30.6$
METRA	44.2 ± 8.6	71.6 ± 8.4	$+27.4$
CSF	51.6 ± 5.1	79.9 ± 5.6	$+28.4$
CCSF, terminating at H	55.8 ± 2.2	94.7 ± 0.7	$+38.9$
CCSF (ours)	71.6 ± 6.5	$\mathbf{97.8 \pm 1.4}$	$+26.3$

Chaining helps every method, and closes no gap. Evaluation-time chaining raises coverage for every method, by $+15.9$ points for DIAYN to $+30.6$ for CIC, so it is not a property of our objective but a generic change to how a skill policy is deployed. It does not make any baseline competitive, as the best of them reaches $80.1 \pm 15.3\%$ against CCSF's $97.8 \pm 1.4\%$, a gap of 17.7 points, and the gap to the best low-variance baseline, CSF at $79.9 \pm 5.6\%$, is 17.9. DISCO-DANCE, the one baseline engineered for bottlenecks, is also the least reliable of the six, with a spread across training seeds of ± 14.5 held and ± 15.3 chained, wider than any other method under either protocol.

Truncating gives better skills, chaining gives better walks. The two CCSF rows separate the two halves of the method. Terminating at a boundary reaches $94.7 \pm 0.7\%$ chained but only $55.8 \pm 2.2\%$ held, within 4.3 points of unmodified CSF, so its skills are worth little on their own and almost all of its gain comes from the chain. Truncating instead reaches $97.8 \pm 1.4\%$ chained and $71.6 \pm 6.5\%$

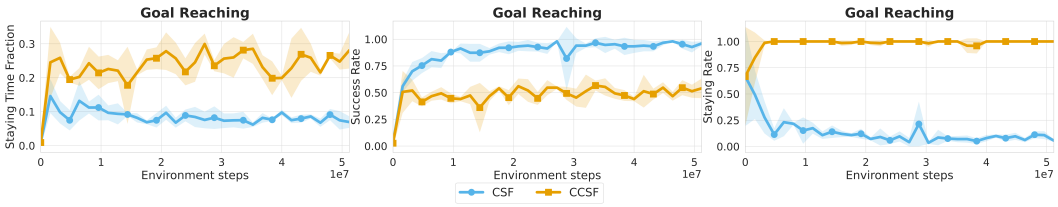


Figure 14: The plots represent the performance for zero-shot goal reaching across the pre-training phase. Left plot shows the *staying time fraction*, middle plot shows the *success rate*, and the right plot shows *staying rate* which measures how often the agent continues to stay in the goal when it reaches the goal. The results show that CSF can reach goals more often, but CCSF can stay within the goal significantly more. The line represent the mean with the shaded area indicating 1 standard deviation across 3 seeds.

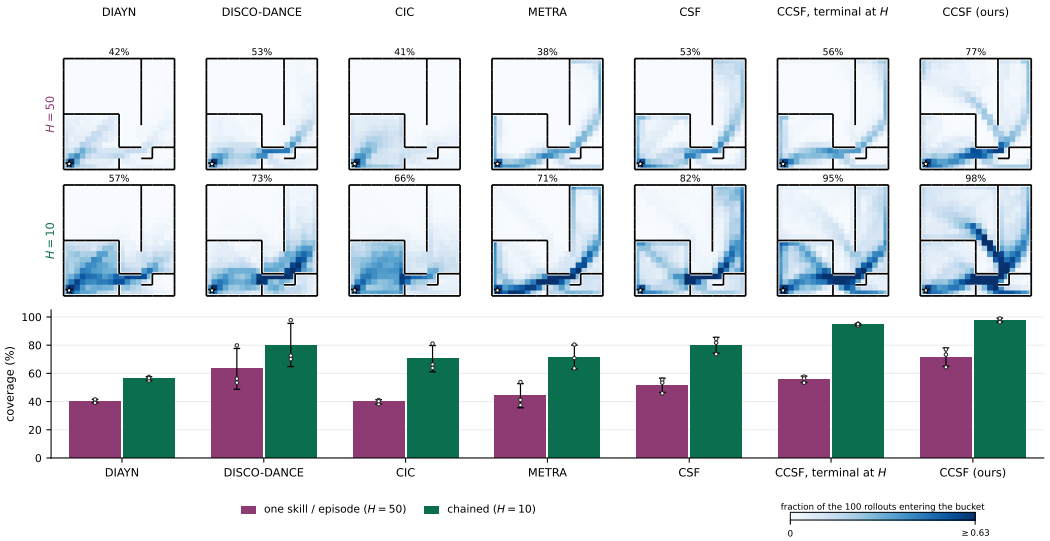


Figure 15: Chaining applied at evaluation time to every method. Each panel shades the 20×20 buckets by the fraction of 100 rollouts that enter them, for the median training seed of that method, with that seed's union coverage above. The top row holds one skill for the whole episode and the bottom row draws a fresh skill every $H = 10$ steps, at the same environment-step budget. Bars give the mean over 3 training seeds with one dot per seed.

held, exactly 20 points above CSF, so bootstrapping through the boundary produces individually better skills as well as better chains. Chaining at evaluation supplies the walk, and training under chaining is what makes the individual edges worth walking on.

H.2 Truncating versus terminating at segment boundaries

Setup. The two variants differ in one flag and nothing else. Truncating treats a segment boundary as a time limit, so the successor-feature target bootstraps through the switch and $\psi^\top z$ remains the value of *continuing* the skill. Terminating treats it as a terminal state, masking the bootstrap so that $\hat{\psi} = \phi(s') - \phi(s)$ and the value of s' is discarded, and each short skill then solves its own finite-horizon problem. Figure 16 reports both, evaluated under each protocol. The panels are rollouts of the 3×10^6-step snapshot of seed 101, and the curves re-evaluate a dense snapshot grid over the first 2.5×10^5 steps, every 2.5×10^4 frames, for all three training seeds. The curves therefore end well before the final snapshot and their levels are not directly comparable with Table 12.

Results. Truncating ends ahead under both protocols. At 2.5×10^5 frames it reaches $94.8 \pm 2.0\%$ chained against $89.9 \pm 0.6\%$, and $56.8 \pm 8.7\%$ held against $50.5 \pm 2.4\%$. The held panels make

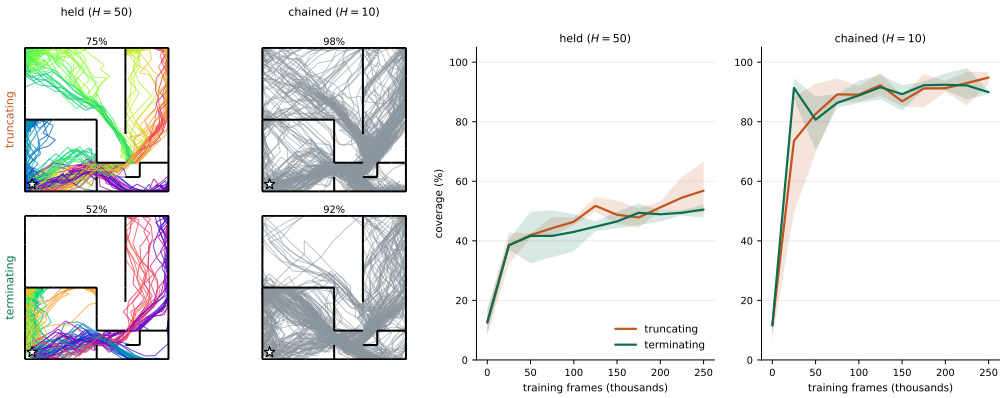


Figure 16: Truncating versus terminating at a segment boundary. Left: rollouts of the final snapshot of seed 101, with one skill held for the episode and with a fresh skill every $H = 10$ steps, union coverage above each panel. Held rollouts are coloured by their skill, and chained rollouts are drawn in one ink since each line is five skills long. Right: coverage against training frames over the dense snapshot grid, where the line is the mean over 3 training seeds and the band spans them.

the difference concrete, as with chaining switched off at evaluation truncating covers 75.3% of the maze against terminating's 52.5% on this seed, and its skills leave the start room in several directions where terminating's crowd into fewer.

The two arrive differently. Under chaining, terminating is at 90% of its own final level by 2.5×10^4 frames and then flat, while truncating passes it at around 5×10^4 and is still climbing at the end of the window. Terminating is not slower, then, but earlier to stop, since masking the bootstrap gives each short skill a well-posed finite-horizon problem that is quickly solved, and solving it is all it ever does. Bootstrapping keeps the value of continuing a skill in the target, which is a harder thing to fit and a better one to have fitted. Truncating is also the more variable of the two across seeds, ± 8.7 against ± 2.4 held, so its advantage is larger in the mean than it is reliable.