PACE: DISCOVERING OPTIONS VIA SELF-DISTILLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Canonical Reinforcement Learning (RL) policies require reading an observation before sampling an action. However, running a policy at each step for a single decision can be expensive. Should an agent be able to emit n actions at a time, this would allow reducing the number of observations it has to read and the number of inference passes it has to run during a decision process, unlocking large computational savings. In this work, we argue that many decision problems contain states where one observation carries enough information for the policy to emit more than one optimal action at a time, making intermediate state reads an avoidable cost. We propose **Per-state Adaptive Commitment (PACE)**: while Proximal Policy Optimisation (PPO) trains as usual, PACE distills it into a multi-step policy that reads one observation and emits several consecutive actions from it alone. Every time this policy reads an observation from the environment, it also predicts the number of actions to output by measuring how far that replay can be trusted, and committing exactly that far. On random tasks with deterministic transitions, PACE reduces the number of required observations by up to $8\times$, and the compression shrinks as transition stochasticity rises. Furthermore, we show that, in open-ended settings, recursively augmenting the action space of a Markov Decision Process with the options discovered by PACE drastically improves success rates in environments where both curriculum and tabula-rasa learning fail.

1 INTRODUCTION

Imagine walking towards a door you can see from where you are: one look fixes the whole route; all else being equal, looking again does not change your route. The intuition at the base of this work is that there exist states of a decision problem where the information contained in one observation is enough to make more than one optimal choice at a time. Yet, a deployed RL policy processes each observation at every step of the decision making problem, even where the next few steps are predictable from the initial one. Each of those reads costs inference time, and every forward pass is wasted when the answer was already in hand. A natural question arises: "*Can an agent autonomously discover action sequences it can trust? How far to trust them without reading observations?*".

Today, a set of works investigates how to discover these temporally-extended actions – i.e., *options*. Hierarchical RL learns the policy over options together with the policy over actions (Bacon et al., 2017; Vezhnevets et al., 2017; Nachum et al., 2018), but these methods still read an observation at every state. Action chunking and observation pricing fold the length of an option into the training objective, either by learning it from multi-step returns (Sharma et al., 2017; Lakshminarayanan et al., 2017; Biedenkapp et al., 2021; Nagy et al., 2026) or by charging the reward for every observation the agent reads (Nam et al., 2021; Koseoglu et al., 2021; Krale et al., 2023); both change the decision problem, which, in turn, changes the optimal policy for that problem. Skill discovery methods run a separate phase before the RL optimisation routine, either unsupervised (Eysenbach et al., 2019) or on demonstrations (Fox et al., 2017; Kim et al., 2019; Pertsch et al., 2020), increasing the data required for training, and reducing sample efficiency. We discuss these approaches in detail in Section 2.

In this work, we introduce a method that uses options to minimise the number of observation readings from the environment – and thus of the forward passes – without changing the decision problem and without requiring additional training data. We propose PACE, which builds on top of a PPO

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.

(Schulman et al., 2017) policy, trained as usual, but adds two heads that reuse its interaction data to discover new options while training. The first one is a multi-step policy that learns to emit more than one action per observation by imitating the PPO one-step policy. The second head predicts from that same observation how many actions to imitate from the base policy. At deployment, the agent reads an observation, commits for the predicted number of actions, and reads again on arrival. We refer to this process as *multi-step self-distillation*.

We first test the ability of PACE to compress the one-step policy into a multi-step policy in Section 6 on fixed and randomised sparse-reward navigation tasks. Results in Table 1 show that PACE reduces the number of required observations by a factor of 8 against the base policy, and by up to 2.7 against the best baseline at a similar success rate. Figure 1 shows that the savings decrease as the environment becomes less predictable, and provides experimental confirmation for our initial intuition that the first observation contains enough information to predict more than one optimal action. Then, we investigate the power of the discovered options to facilitate control in complex tasks in open-ended settings: "*Does using the new options make the control problem easier in tasks where curriculum or tabula-rasa learning fail?*". Tables 2 and 3 show that, by recursively augmenting the action space of a Markov Decision Process (MDP) or a Partially-observable Markov Decision Process (POMDP) with the options discovered in the earlier tasks, a base PPO policy can solve problems where tabula-rasa and curriculum learning fail.

2 Related Work

Three lines of work stand out for aiming to discover options from interacting with an environment.

Return-based action chunking. The first line of work includes methods where one update spans several steps without reading the environment in between. Fine Grained Action Repetition (FiGAR) and Dynamic Action Repetition (DAR) add repetition counts to the action space (Sharma et al., 2017; Lakshminarayanan et al., 2017). TempoRL extends them by learning the count (Biedenkapp et al., 2021). Sample Efficient Action Chunking Reinforcement Learning (SEAR) (Nagy et al., 2026), Q-chunking (Li et al., 2025), and Adaptive Chunk Size Actor-Critic with Causal Transformer Q-network (ACSAC) (Chen et al., 2026) predict a whole action sequence at once. All of them seek to solve the task and save observations at the same time, and they learn the multi-step policy directly from returns. In all these methods, learning gets harder as the action chunk becomes longer (Nagy et al., 2026), which makes training difficulty and deployment efficiency hard to balance.

Observation pricing. Observation costs cut trajectories into options the same way: each paid read starts a new segment, and the segment between two reads is what gets executed without looking. These methods price observations directly (Nam et al., 2021) and learn which observations are worth taking (Koseoglu & Ozcelikkale, 2020; Koseoglu et al., 2021; Krale et al., 2023). The drawback is twofold. Pricing the reads changes the original problem, so the optimal policy shifts with the price. And a cut made to save reads is not made to be reused: nothing guarantees the segment between two reads will show up again.

Skills pre-training. Skill discovery methods learn where an option begins and ends before the RL optimisation starts: options are cut by hand (McGovern & Sutton, 1998), counted from frequent action patterns (Hosu et al., 2025), or read off subtask and subgoal structure (Konidaris & Barto, 2009; Tessler et al., 2017; Fox et al., 2017). All of them increase the data required for training and reduce sample efficiency. Furthermore, for any human intervention, our intuition is the opposite: an option or subtask manually defined by hand does not always hold for the agent. On a randomly generated *DoorKey* map, the subgoal of reaching the key lands the key in a different cell every episode, which means that reusable segments do not necessarily fall on the boundaries of task structure, and the cuts are therefore not necessarily optimal.

3 Preliminaries

Canon. We consider discounted MDP settings with discrete action spaces. An MDP $\mathcal{M}$ is described by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, with state space $\mathcal{S}$, action space $\mathcal{A}$, state-transition probability

$s_{t+1} \sim P(\cdot|s_t, a_t)$, deterministic-Boolean reward function $r_t = R(s_t, a_t) \in \{0, 1\}$ and discount factor $\gamma \in [0, 1]$. We denote returns with $G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$. An episode *succeeds* when it collects $r_t = 1$ before it is truncated at T; the *success rate* of a policy is the fraction of its episodes that succeed. We use $a_t \sim \pi(\cdot|s_t)$ for the policy over actions, and we refer to it as a *base policy*. In POMDPs, for the sake of readability, we abuse notation and denote a history of observations of length c with $s_t = \{o_k \colon t - c \leq k \leq t\}$, where o_k is the observation at time k. We recall Sutton et al. (1999), where an option ω is a tuple $(\mathcal{I}, \pi_\omega, \beta)$, with $\mathcal{I} \subseteq \mathcal{S}$ the initiation set, $\pi_\omega(a \mid s)$ the intra-option policy, and $\beta : \mathcal{S} \to [0, 1]$ the termination function, the probability that the option stops in a state. $\pi_\Omega(\omega \mid s_t)$ describes the policy over options: a mapping from state to a distribution over options. Acting over options turns the MDP into a semi-Markov Decision Process (SMDP). An option initiated at s_t runs until β stops it; if it stops after h primitive steps, that run is one SMDP transition: it lands in s_{t+h} and collects $r_\omega = \sum_{j=0}^{h-1} \gamma^j \, r_{t+j}$, and the value of s_{t+h} enters the update discounted by γ^h.

Options as action sequences. Here, we consider options with $\mathcal{I} = \mathcal{S}$, where ω is itself an ordered sequence of primitive actions, $\omega = (\omega_0, \ldots, \omega_{h-1})$ with $\omega_j := a_{t+j}$, and where β is a function only of the initiating state s_t, rather than at every state visited during execution. β determines the length of ω directly, $h \sim \beta(\cdot \mid s_t) \in \{1, \ldots, h_{\max}\}$, and we call h the *commitment length*: the number of primitive actions the option emits. Since we represent options as action sequences, option execution becomes trivial. We omit π_ω and have $\pi_\Omega : \mathcal{S} \to \Omega$, with Ω the set of action sequences of length at most $h_{\max}$, such that $\omega = (\omega_0, \ldots, \omega_{h-1}) \sim \pi_\Omega(\cdot \mid s_t, h)$. Because h and every ω_j are fixed by s_t alone, executing ω requires no further evaluation of β and π_Ω after time t: the h actions run open-loop, i.e., without reading the states they pass through, where a closed-loop policy would read each one. Each evaluation reads one observation from the environment and runs one forward pass, so the number of options an episode uses is also the number of observations it reads.

4 METHODS

To operationalise our intuitions, we propose PACE. PACE trains three disjoint parameter groups, each with its own optimisation objective. We use θ for a parameter group and use subscripts to distinguish them: the base policy $a_t \sim \pi(\cdot \mid s_t, \theta_a)$; the option policy $\omega = (\omega_0, \ldots, \omega_{h-1}) = (a_t, \ldots, a_{t+h-1}) \sim \pi_\Omega(\cdot \mid s_t, h; \theta_\omega)$; and the termination function $h \sim \beta(\cdot \mid s_t, \theta_h)$, $h \in \{1, \ldots, h_{\max}\}$. We proceed next to describe the training objectives and protocols for each in order.

4.1 THE BASE POLICY

We optimise θ_a using the standard PPO objective:

$$\mathcal{L}_{\text{PPO}}(\theta_a) = \mathbb{E}_t\Big[\min\big(\rho_t \, \hat{A}_t,\ \text{clip}(\rho_t,\, 1 - \epsilon_{\text{clip}},\, 1 + \epsilon_{\text{clip}})\, \hat{A}_t\big)\Big], \qquad \rho_t = \frac{\pi(a_t \mid s_t, \theta_a)}{\pi(a_t \mid s_t, \theta_a^{\text{old}})}, \quad (1)$$

where ρ_t is the probability ratio against the parameters that collected the rollout, ϵ_{clip} the clipping radius, and $\hat{A}_t = \sum_{k=0}^{T-t-1} (\gamma\lambda)^k \, \delta_{t+k}$ with $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ is the generalised advantage (Schulman et al., 2016), computed from a learned value function V. Recall that π is a canonical, one-step policy. While our experiments only test PACE using PPO as a base policy, there is no theoretical restriction to PPO.

4.2 THE OPTION POLICY

The option policy – i.e., the multi-step policy that imitates the base policy, but outputs more than one step at a time – is composed of two elements: a *codebook bottleneck* and a *decoder head*.

Codebook bottleneck. Our intuition is that an option is only worth learning if the situation that calls for it comes back. Therefore, we force the model to learn a discrete codebook that associates the representation of a state with the most useful option. At each step, PACE encodes the state s_t, compares it to each of the latent codes in the codebook, and selects the one that is closest to that

representation of that state. Bottlenecking the model with a codebook of a size way smaller than the number of states in the MDP allows the model to learn to group states that require to act alike: "*What skill in my library is the most useful in this situation?*", and prevents the model from memorising one fixed action string per observation.

Formally, an encoder $f^{\text{enc}} : \mathcal{S} \to \mathbb{R}^{d_z}$ with parameters θ_ω maps a state s_t to its latent representation $z_t \in \mathbb{R}^{d_z}$, and the codebook is a set of M learned vectors $\mathcal{E} = \{e_1, \dots, e_M\} \subset \mathbb{R}^{d_z}$. To select the skill most appropriate in a certain state s_t, we find the latent code $e_t^* \in \mathcal{E}$ that is the closest to z_t in the ℓ_2 distance,

$$e_t^* = \arg\min_{e_i \in \mathcal{E}} \|z_t - e_i\|_2, \tag{2}$$

with straight-through gradients from e_t^* to z_t. We train the codebook with the quantisation and commitment losses (van den Oord et al., 2017): the former moves each entry toward the encoder outputs assigned to it, the latter, weighted by λ_{c}, keeps the encoder output close to the codebook entry so that assignments stay stable:

$$\mathcal{L}_{\text{VQ}}(\theta_\omega) = \left\|\text{sg}[z_t] - e_t^*\right\|_2^2 + \lambda_{\text{c}} \left\|z_t - \text{sg}[e_t^*]\right\|_2^2, \tag{3}$$

where $\text{sg}[\cdot]$ stops the gradient at its argument.

Option policy self-distillation. A decoder turns the latent code into an action sequence one position at a time, conditioning each step on what the option has already committed to. Formally, given the latent code of an option e_t^* as described earlier, it gives the action distribution at position $j \geq 1$ for an option $\omega = (\omega_0, \dots, \omega_{h-1}) = (a_t, \dots, a_{t+h-1})$. Besides the code, it reads the actions already in the option through a learned embedding e^{act} of each action: their sum $\bar{e}(\omega_{0:j-1}) = j^{-1/2} \sum_{i=0}^{j-1} e^{\text{act}}(\omega_i)$ and the embedding of the last action, $e^{\text{act}}(\omega_{j-1})$, together with a learned embedding $e^{\text{pos}}(j)$ of the position itself. We refer to its output as the *factorised distribution*:

$$\pi_\Omega^j(\omega_j \mid s_t, \omega_{0:j-1}; \theta_\omega) = f^{\text{dec}}\big(e_t^*,\ \bar{e}(\omega_{0:j-1}),\ e^{\text{act}}(\omega_{j-1}),\ e^{\text{pos}}(j)\big). \tag{4}$$

The *option policy* is the joint distribution over these factors,

$$\pi_\Omega\big(\omega \mid s_t, h;\ \theta_\omega\big) = \prod_{j=0}^{h-1} \pi_\Omega^j\big(\omega_j \mid s_t,\ \omega_{0:j-1};\ \theta_\omega\big). \tag{5}$$

We train the option policy to imitate the base policy by distilling π – which reads every intermediate state $\{s_t, \dots, s_{t+h_{\max}}\}$ – into π_Ω, which reads only the starting state s_t. At each update we reuse the batch of trajectories collected by PPO using π. From these trajectories, we construct windows of length $h_{\max}$, starting at s_t. The option policy then receives the actions $a_{t:t+j-1}$ as $\omega_{0:j-1}$, and the action distribution at position j: $\pi(\cdot \mid s_{t+j})$. Finally, we pull the option policy towards the base policy's distribution at s_{t+j} using the Kullback-Leibler (KL) divergence between the two:

$$\mathcal{L}_{\text{distill}}(\theta_\omega) = \mathbb{E}_j\Big[\mu_j \,\text{KL}\Big(\text{sg}\big[\pi(\cdot \mid s_{t+j}, \theta_a)\big] \,\Big\|\, \pi_\Omega^j(\cdot \mid s_t, \omega_{0:j-1}; \theta_\omega)\Big)\Big], \tag{6}$$

where $\mu_j \in \{0, 1\}$ masks positions whose window leaves the trajectory or crosses an episode boundary.

4.3 The termination function

The termination function $\beta(\cdot \mid s_t, \theta_h)$ predicts, from the initiating state alone, how far the option can be trusted. We define the agreement event at depth j as the event in which the option's greedy action matches the base policy's greedy action at s_{t+j}: with $a_j^* = \arg\max_a \pi(a \mid s_{t+j}, \theta_a)$,

$$\alpha_j = \mathbf{1}\big[\arg\max_a \pi_\Omega^j(a \mid s_t, \omega_{0:j-1}; \theta_\omega) = a_j^*\big]. \tag{7}$$

Past the first disagreement the option leaves the executed path, so $m_j = \mu_j \prod_{i<j} \alpha_i$ marks the positions reached with agreement so far. Formally, β outputs one number per position: $b_j = \beta^j(s_t; \theta_h)$ is the Bernoulli probability that position j is the first disagreement, given agreement at every earlier position. Multiplying them gives $c_j = \prod_{i \leq j}(1 - b_i)$, the probability that the option still agrees at position j, which is non-increasing in j, and implements the distribution over commitment lengths

of Section 3, $\beta(h > j \mid s_t, \theta_h) = c_j$. Each b_j is trained as a binary classifier of the disagreement event $1 - \alpha_j$, on the positions reached with agreement so far:

$$\mathcal{L}_{\text{term}}(\theta_h) = -\mathbb{E}_j\Big[(1-\alpha_j)\log b_j + \alpha_j \log(1-b_j)\,\Big|\, m_j{=}1\Big]. \tag{8}$$

$1 + \sum_j c_j$ estimates the expected commitment length $\mathbb{E}[h \mid s_t]$.

Final objective and deployment. The three groups are trained together on:

$$\mathcal{L} = \mathcal{L}_{\text{PPO}}(\theta_a) + \mathcal{L}_{\text{VQ}}(\theta_\omega) + \lambda_{\text{d}}\,\mathcal{L}_{\text{distill}}(\theta_\omega) + \mathcal{L}_{\text{term}}(\theta_h), \tag{9}$$

where λ_{d} anneals linearly from 0 to its target value early in training (Appendix D.5), and no gradient crosses parameter groups.

At inference time, for a confidence threshold $\tau \in (0, 1)$, the agent reads s_t, takes the base policy's greedy action at position 0, and continues with π_Ω greedily for a total of $h = 1 + |\{j : \; c_j \geq \tau\}|$ actions. τ is fixed for all experiments, and as $\tau \to 1$, $h \equiv 1$ and PACE is exactly π. Algorithm 1 in Appendix B lists one training iteration and the deployment loop.

5 Experimental Settings

5.1 Environments

All experiments are navigation tasks in NAVIX (Pignatelli et al., 2025), a JAX (Bradbury et al., 2018) reimplementation of MiniGrid (Chevalier-Boisvert et al., 2019; 2023). Reward is 1 on reaching the goal and 0 everywhere else, with no per-step penalty; an episode that hits the step limit ends unrewarded. We write *DKN* for *DoorKey* in an $N \times N$ room, *DON-NxPy* for *Dynamic-Obstacles* with x obstacles that move once every y steps (*static* in place of the period: they do not move), and *SCN-Nw* for *SimpleCrossing* with w walls. The *Room Chain* of Section 7.2 is built from these rooms.

5.2 Protocol

Every method is built on the same PPO implementation in *rejax* (Liesen et al., 2024). For each method and task we search over 24 hyperparameter configurations with 3 seeds each, using the full training budget of that task (1.0M primitive frames on *DK5*, 10.5M on *DK8* and the *DO6* series, 5.2M on *SC11-N5*). We select the configuration with the highest greedy success rate at the end of training. For each experiment, that configuration is retrained from scratch on 16 or 32 new seeds. Each of these seeds is then evaluated greedily on 256 episodes sampled from a single, held out seed shared across all runs and methods. We measure two quantities: whether a method's success rate is similar to PPO's (Appendix A.3), and how many observations per episode it reads relative to PPO, which reflects the number of inference passes a method performs at deployment. We report mean and standard deviation over seeds.

6 Observation Compression at Deployment

6.1 Minimising deployment observations

We first aim to test whether PACE can learn a multi-step policy that holds the base policy's performance while reducing the number of observation reads from the environment. We compare PACE with PPO and four baselines that train a multi-step policy directly: Fixed, SEAR, TempoRL, and DAR; we describe their implementations and deployment settings in Appendix C. Every method observes the whole grid, on layouts re-drawn each episode (Appendix D.2). We train each method *tabula-rasa* on four environments and measure success rate and compression.

Results in Table 1 show that PACE matches PPO's success rate on every task and reaches the goal in the same number of steps, but with far fewer observation reads: up to $8\times$ fewer than PPO (*DK5*, 1.18 reads per episode) and up to $2.7\times$ fewer than the best baseline at a similar success rate (*SC11-N5*). From *DK8* up, Fixed and SEAR achieve lower success rates, and their decline as the training length grows is marked. We investigate it in Appendix E.2.

Table 1: Observation compression rate of PACE against the set of baselines described in Section 6.1. Compression is PPO's observations per episode over the method's, per seed. Steps to goal is the number of MDP steps from start to goal. Commitment length is the number of primitive frames executed in one SMDP step. Observations are averaged over successful episodes. Bold: best compression among the methods whose success rate is similar to PPO's.

Task	Method	Success ↑	Compression ↑	Steps to goal	Observations per episode ↓	Commitment length
DK5	PPO	1.00±0.00	1.00±0.01×	9.48±0.12	9.48±0.12	1.00±0.00
	Fixed	0.93±0.13	3.47±0.49×	22.35±3.87	2.79±0.42	8.00±0.33
	SEAR	1.00±0.00	3.60±0.38×	18.79±1.46	2.66±0.26	7.08±0.29
	TempoRL	0.99±0.06	1.10±0.06×	10.56±1.02	8.62±0.46	1.23±0.14
	DAR	0.94±0.22	1.11±0.07×	10.47±2.69	8.59±0.54	1.21±0.24
	PACE	1.00±0.00	**8.01±0.00**×	9.43±0.00	**1.18±0.00**	7.97±0.00
DK8	PPO	0.99±0.03	1.00±0.01×	16.65±0.17	16.65±0.17	1.00±0.00
	Fixed	0.03±0.06	3.51±0.79×	45.13±11.14	4.97±1.01	9.03±0.57
	SEAR	0.51±0.17	2.79±0.16×	38.56±2.45	6.00±0.34	6.43±0.07
	TempoRL	0.80±0.20	1.27±0.05×	18.22±0.69	13.09±0.54	1.39±0.05
	DAR	0.84±0.10	1.20±0.09×	18.78±1.23	13.97±1.20	1.35±0.07
	PACE	0.99±0.01	**6.34±0.85**×	16.56±0.11	**2.70±0.60**	6.30±0.83
DO6-N3P2	PPO	0.79±0.02	1.00±0.03×	11.97±0.31	11.97±0.31	1.00±0.00
	Fixed	0.37±0.04	6.11±0.84×	14.98±1.75	1.99±0.23	7.53±0.11
	SEAR	0.63±0.02	1.05±0.06×	11.49±0.72	11.49±0.72	1.00±0.00
	TempoRL	0.74±0.03	1.45±0.09×	13.49±0.44	8.31±0.49	1.63±0.13
	DAR	0.77±0.02	1.19±0.04×	12.54±0.35	10.07±0.35	1.25±0.03
	PACE	0.78±0.02	**1.56±0.07**×	12.04±0.30	**7.69±0.35**	1.57±0.04
SC11-N5	PPO	0.92±0.01	1.00±0.02×	18.93±0.30	18.93±0.30	1.00±0.00
	Fixed	0.00±0.00	6.84±1.97×	26.17±7.65	3.00±0.82	8.74±0.70
	SEAR	0.22±0.10	8.40±1.16×	21.81±3.48	2.30±0.37	9.47±0.14
	TempoRL	0.92±0.03	2.64±0.16×	18.89±0.61	7.21±0.45	2.63±0.20
	DAR	0.93±0.01	2.24±0.16×	19.10±0.37	8.49±0.64	2.26±0.16
	PACE	0.92±0.01	**7.22±0.49**×	18.90±0.20	**2.63±0.19**	7.21±0.48

TempoRL and DAR, the two baselines with a variable commitment length, are given $3.2\times$–$4\times$ the training budget of the other methods to guarantee full convergence (Appendix C) but still fall short of PPO's success rate on half of the tasks; where they keep it, they commit to under three steps per observation and save at most $2.6\times$. Most likely, this is because both compose options by repeating one identical action, and in these navigation tasks one action is rarely worth repeating for long.

Ablating the termination function. In the light of these results, we ask whether a fixed commitment length could replace the termination function. On all four tasks we force-deploy PACE, SEAR and Fixed at every fixed horizon $h = 1, \ldots, 10$, and compare them against PACE deployed with its termination function. No fixed h holds the success rate of the termination function (Figure 2): averaged over the four tasks it already falls behind at h=1 and declines to 0.865 at h=10, while the two fixed-length baselines stay below 0.57 throughout. Choosing the length at the initiating state commits more where the next steps are predictable and less where they are not. We further investigate where the termination function is more useful by looking at results per environment (Appendix Figure 7) and observe the termination function pays off most in stochastic environments.

6.2 How does stochasticity affect observation compression?

Since PACE learns to commit to a sequence of actions, we expect it to be more effective in environments that are predictable: those where the information in the current observation is enough to predict many steps ahead. In this section we test this hypothesis by measuring the performance of PACE on environments of rising stochasticity. We measure stochasticity as the transition entropy of the environment, and we vary it by changing the number of moving obstacles and their move-

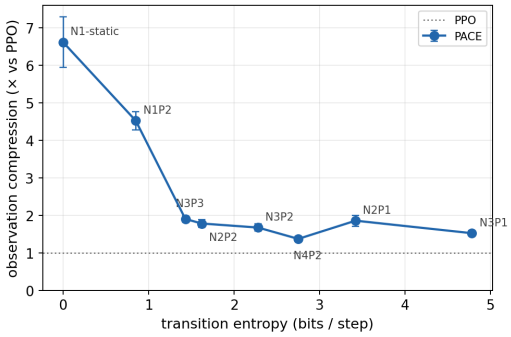

Figure 1: Stochasticity ladder on *Dynamic-Obstacles* 6×6: observation compression shrinks with transition entropy.

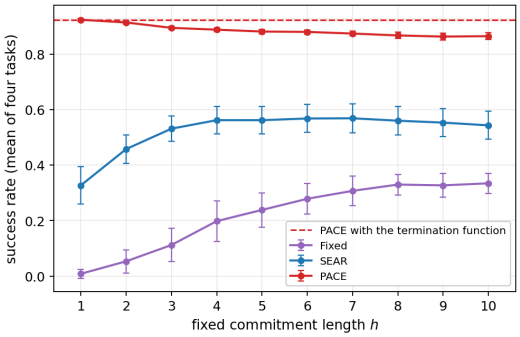

Figure 2: Success rate of our baselines and PACE ablated by its termination function at different, fixed commitment lengths. Per-task panels in Appendix Figure 7.

ment period in *Dynamic-Obstacles*. To evaluate the transition entropy, we deploy a single-step PPO policy trained on each environment and we approximate the Shannon entropy of the next-step observation distribution at every state it visits by sampling $R{=}128$ next-step observations from the environment. The entropy of *Dynamic-Obstacles* rises with the number of obstacles and with how often they move. We sweep hyperparameters using 6 configurations and 3 seeds each to select the best configuration per environment.

Figure 1 shows that as environments become less predictable, PACE commits to shorter sequences of actions per step, and therefore reads more observations from the environment: compression falls from $6.6\times$ at zero entropy to about $1.5\times$ at the top of the ladder. This is consistent with our hypothesis: when the environment is less predictable, it is not optimal for the agent to commit to long sequences of actions.

7 The Impact of the Discovered Options in Open-Ended Settings

Section 6 tested compression. Here, we investigate the impacts of augmenting an MDP's action set with the newly discovered options for open-ended learning.

We reuse the options that PACE discovers on one level as actions on the next, and measure their performance on two environments, a *DoorKey Ladder* and a *Room Chain*, which we describe in Section 7.1 and Section 7.2, respectively. Both tasks come in levels that are progressively harder to solve. The *DoorKey Ladder* stresses exploration, while the *Room Chain* stresses credit assignment. At the end of each level, we deploy the policy we have just trained and record the sequence each codebook entry commits to most often. We then take the top 8 by length-weighted frequency and add them to the action space of the next level.

7.1 Exploration Boost

Under sparse rewards, the chance of a near-random policy reaching the reward decays exponentially with the horizon (Jinnai et al., 2019). Temporally extended actions ease this by folding many steps into one option (Dabney et al., 2021; Machado et al., 2017). Here, we test whether PACE's options generalise across episodes and layouts to boost exploration.

We measure the performance of PACE in the *DoorKey Ladder*: a set of five *DoorKey* tasks of increasing room size, fully random layouts, and a first-person view (Appendix D.2). At every level, rewards become more sparse, and exploration more difficult. We compare three methods, each trained *tabula-rasa* on every level: primitive actions only, the action space augmented by the selected options, and the action space augmented by randomly generated options of the same number and length. Every arm trains on 10.5M primitive frames at the first level and 21.0M at each later level, with 16 seeds and one shared configuration.

Table 2: Success rate of the three methods on the *DoorKey Ladder*. We report the average and standard deviation across 16 seeds. Bold: best per level.

Level	Actions	Actions and Discovered options	Actions and Random options
DK8	$0.990 \pm .002$	—	—
DK10	$0.973 \pm .003$	$\mathbf{0.978 \pm .004}$	$0.909 \pm .235$
DK12	$0.793 \pm .300$	$\mathbf{0.949 \pm .013}$	$0.247 \pm .396$
DK14	$0.554 \pm .375$	$\mathbf{0.882 \pm .019}$	$0.000 \pm .000$
DK16	$0.251 \pm .325$	$\mathbf{0.885 \pm .030}$	$0.000 \pm .000$

Training *tabula-rasa* degrades with map size and collapses at *DK16*, where it reaches 0.251, while the discovered options hold 0.885 there (Table 2). At *DK14* and *DK16* the *tabula-rasa* numbers carry a very large standard deviation, which comes from part of the seeds reaching a high success rate while the rest stay close to zero, the behaviour we expect once a larger map makes exploring to the reward much less likely. The discovered-options arm keeps a high success rate with a small deviation across seeds, so the options give a stable head start to exploration.

Where the gain comes from. An option augmenting the action space works on two axes at once: it *(a)* adds semantically different actions, and *(b)* allows selecting actions that execute more than one step in the environment. To understand the contribution of the options on each axis, we ablate the options with a random set: we select 8 options but compose them by drawing actions uniformly at random over primitives. The random-options arm in Table 2 shows that performance declines level by level and reaches 0.000 at *DK14* and *DK16*, below the arm that trains on primitive actions only. This evidence suggests that a longer commitment unit alone brings no gain, and that with the wrong content it degrades performance instead, so what helps is the content of the options rather than their length.

We also compare against two further arms. *Curriculum* inherits the parameters of the previous level instead of its options. *Manual options + Tabula-rasa* counts, by frequency, the segments committed along the previous level's deployment trajectories and takes the top 8 of them as the options that augment the action space. Appendix F reports both.

7.2 Credit-Assignment Boost

Section 7.1 kept the structure of the task fixed and scaled it up; here it changes, since every level adds a new kind of room while the reward still arrives only at the end of the chain. A room added at one level has to be solved before any reward is seen, and its solution has to be credited from an outcome several rooms away. An option executes several primitive steps as one decision, which shortens the path between the reward and the actions that earn it, and we ask whether the options discovered on the previous levels shorten it enough to keep the chain solvable.

In the *Room Chain*, rooms connect end to end and the reward comes only at the end of the chain. There are five levels (Figure 3). *L1* is a single room to navigate. *L2* to *L4* have two rooms each, one of them the type that level introduces, lava at *L2*, a door and key at *L3* and an obstruction at *L4*, and the other drawn per episode from the types already seen, in either order. *L5* introduces nothing new and presents four rooms, with their order randomised every episode. Table 6 describes the obstruction room. The agent sees an occluded first-person view (Appendix D.2). We compare four arms. They are primitive actions *tabula-rasa*, a primitive curriculum that inherits the parameters of the previous level, the options of the previous level added to the action space and trained *tabula-rasa*, and the two combined. Every arm trains on 10.5M primitive frames at *L1*, *L2* and *L3* and on 21.0M at *L4* and *L5*, with 16 seeds.

From *L2* onward training *tabula-rasa* stays at 0.000 (Table 3), so nothing is learned here without carrying something across levels. The curriculum settles between 0.30 and 0.35 from *L3* on, with a very large standard deviation, because part of its seeds never reach the end of the chain. Only the combination stays above 0.85 at every level. On the last three levels the options bring more than 0.5

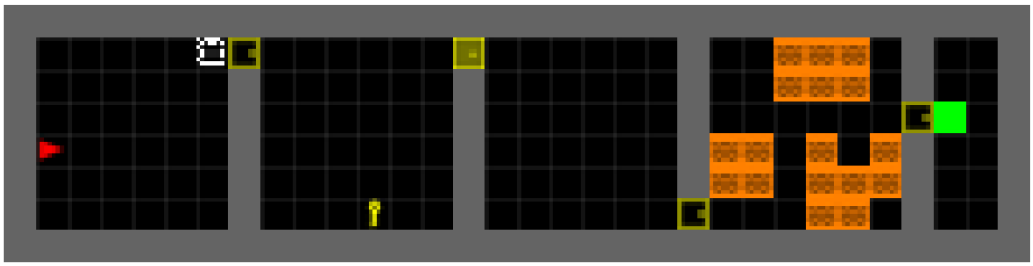

Figure 3: A sample of level *L5* of the *Room Chain*, with the four room types in an order that is randomised every episode. The agent is red and the goal green; the obstruction is white, the door and key yellow, and the lava orange. Appendix F shows the other levels.

Table 3: *Room Chain*. Five levels with 16 seeds per arm; every cell is the success rate as mean±std over seeds. *L1* is the root, where only the *tabula-rasa* arm trains, and the other three arms start from *L2*. Bold: best per level.

Level	*Tabula-rasa*	*Curriculum*	Discovered options + *Tabula-rasa*	Discovered options + *Curriculum*
L1	0.999±.005	—	—	—
L2	0.000±.000	0.939±.009	**0.958±.005**	0.955±.010
L3	0.000±.000	0.350±.452	0.387±.255	**0.935±.010**
L4	0.000±.000	0.337±.435	0.097±.258	**0.914±.017**
L5	0.000±.000	0.303±.392	0.000±.000	**0.850±.022**

of success rate on top of the curriculum and cut the standard deviation across seeds from about 0.4 to about 0.02.

Two controls rule out the obvious alternatives (Table 14). Raising the curriculum's entropy coefficient threefold or tenfold leaves *L4* at 0.000, so the curriculum does not fail for lack of exploration, and replacing the options with a shape-matched random set leaves *L4* at 0.000 as well, which again shows that what carries the chain is the content of the options rather than their presence.

8 Conclusion and Outlook

We set out to ask whether an agent can discover action sequences it can trust, and how far to trust them without looking again. PACE proposes an answer to both inside the single-step PPO recipe, by self-distilling the base policy into open-loop options and predicting, at the initiating state, how long each of them holds. Observations per episode compress by up to about 8× on deterministic tasks at a similar success rate and steps to goal, and stochasticity shortens commitments and shrinks savings. The options discovered on each task let *tabula-rasa* training beat parameter-inheriting curriculum on the *DoorKey Ladder*, and lifts the *Room Chain*'s last level from 0.303 to 0.850.

The approach has limits. The options are fixed action sequences, scored by whether the replay still agrees with the base policy's greedy action, so wherever several different action sequences are equally good the agreement signal falls and the commitments shorten; and along the chain the combination still loses ground level by level, from 0.955 to 0.850, which the options slow without stopping. Replacing that agreement on single actions with an equivalence on a goal state or on a value is future work.

The next step is an end-to-end loop that takes options in and retires them online, and a move beyond MiniGrid, where we have so far run only a three-seed probe on Craftax (Appendix F.2). Augmenting the action space also enlarges it, and a larger action space makes exploration more random, so each expansion should be able to drop primitive actions and not only add options. Applying the same idea to a language model could sharply reduce the number of forward passes it runs, and with them the cost of deployment.

AI USE STATEMENT

We used generative AI tools for the organisation of experiment scripts, for polishing and proofreading the text, and for part of: the search for and organisation of related work, the implementation and debugging of the experiment code, and the drafting of figures and tables. We did not use generative AI tools for any task beyond those listed. All AI-assisted work was reviewed by the authors: the code was checked and tested, every reported number was recomputed from the saved run logs and checkpoints, and all text was revised. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The Option-Critic architecture. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 31, pp. 1726–1734, 2017. doi: 10.1609/aaai.v31i1.10916.

André Biedenkapp, Raghu Rajan, Frank Hutter, and Marius Lindauer. TempoRL: Learning when to act. In *International Conference on Machine Learning (ICML)*, volume 139, pp. 914–924, 2021.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Yash Katariya, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: Composable transformations of Python+NumPy programs. `https://github.com/jax-ml/jax`, 2018.

Qian Chen, Junqiao Zhao, Hongtu Zhou, Hang Yu, Yanping Zhao, Chen Ye, and Guang Chen. ACSAC: Adaptive chunk size actor-critic with causal transformer Q-network. *arXiv preprint arXiv:2605.11009*, 2026.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. BabyAI: A platform to study the sample efficiency of grounded language learning. In *International Conference on Learning Representations (ICLR)*, 2019.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo de Lazcano, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan K. Terry. Minigrid & Miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, volume 36, 2023.

Will Dabney, Georg Ostrovski, and André Barreto. Temporally-extended ϵ-greedy exploration. In *International Conference on Learning Representations (ICLR)*, 2021.

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 (ICLR)*, 2019.

Roy Fox, Sanjay Krishnan, Ion Stoica, and Ken Goldberg. Multi-level discovery of deep options. *arXiv preprint arXiv:1703.08294*, 2017.

Ionel-Alexandru Hosu, Traian Rebedea, and Razvan Pascanu. Meta-learning how to share credit among macro-actions. *arXiv preprint arXiv:2506.13690*, 2025.

Yuu Jinnai, Jee Won Park, David Abel, and George Konidaris. Discovering options for exploration by minimizing cover time. In *International Conference on Machine Learning (ICML)*, volume 97, pp. 3130–3139, 2019.

Heecheol Kim, Masanori Yamada, Kosuke Miyoshi, and Hiroshi Yamakawa. Macro action reinforcement learning with sequence disentanglement using variational autoencoder. *arXiv preprint arXiv:1903.09366*, 2019.

George Konidaris and Andrew G. Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 22, pp. 1015–1023, 2009.

Mehmet Koseoglu and Ayca Ozcelikkale. How to miss data? Reinforcement learning for environments with high observation cost. In *ICML Workshop on the Art of Learning with Missing Values (Artemiss)*, 2020.

Mehmet Koseoglu, Ece Kunduracioglu, and Ayca Ozcelikkale. Learning to observe with reinforcement learning. In *ICML Workshop on Reinforcement Learning Theory*, 2021.

Merlijn Krale, Thiago D. Simão, and Nils Jansen. Act-Then-Measure: Reinforcement learning for partially observable environments with active measuring. In *International Conference on Automated Planning and Scheduling (ICAPS)*, volume 33, pp. 212–220, 2023. doi: 10.1609/icaps.v33i1.27197.

Aravind S. Lakshminarayanan, Sahil Sharma, and Balaraman Ravindran. Dynamic action repetition for deep reinforcement learning. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 31, pp. 2133–2139, 2017. doi: 10.1609/aaai.v31i1.10918.

Qiyang Li, Zhiyuan Zhou, and Sergey Levine. Reinforcement learning with action chunking. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, 2025.

Jarek Liesen, Chris Lu, and Robert Tjarko Lange. rejax: Hardware-accelerated reinforcement learning algorithms in pure JAX. `https://github.com/keraJLi/rejax`, 2024.

Marlos C. Machado, Marc G. Bellemare, and Michael Bowling. A Laplacian framework for option discovery in reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 70, pp. 2295–2304, 2017.

Amy McGovern and Richard S. Sutton. Macro-actions in reinforcement learning: An empirical analysis. Technical Report 98-70, Department of Computer Science, University of Massachusetts Amherst, 1998.

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

C. F. Maximilian Nagy, Onur Celik, Emiliyan Gospodinov, Florian Seligmann, Weiran Liao, Aryan Kaushik, and Gerhard Neumann. SEAR: Sample efficient action chunking reinforcement learning. *arXiv preprint arXiv:2603.01891*, 2026.

HyunJi Alex Nam, Scott Fleming, and Emma Brunskill. Reinforcement learning with state observation costs in action-contingent noiselessly observable Markov decision processes. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, 2021.

Karl Pertsch, Youngwoon Lee, and Joseph J. Lim. Accelerating reinforcement learning with learned skill priors. In *Conference on Robot Learning (CoRL)*, volume 155, pp. 188–204, 2020.

Eduardo Pignatelli, Jarek Liesen, Robert Tjarko Lange, Chris Lu, Pablo Samuel Castro, and Laura Toni. NAVIX: Scaling MiniGrid environments with JAX. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, volume 38, 2025.

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

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

Sahil Sharma, Aravind S. Lakshminarayanan, and Balaraman Ravindran. Learning to repeat: Fine grained action repetition for deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017.

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

Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J. Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in Minecraft. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 31, pp. 1553–1561, 2017. doi: 10.1609/aaai.v31i1.10744.

Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pp. 6306–6315, 2017.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. FeUdal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 70, pp. 3540–3549, 2017.

A Statistical Accounting and Testing Conventions

A.1 Accounting for Observations and Step Counts

Option statistics for the baseline methods are truncated at environment termination: only committed segments that were actually executed open-loop are counted; the commitment length column reports, for every method, the primitive steps executed per observation.

A.2 Deployment Comparison Conventions

Deployment-side success rates are compared with PPO's on per-seed paired differences, with a one-sided 95% lower bound (Appendix A.3). Each set of numbers appears only once in the main text; for reproduction, the conventions of this appendix and Appendix D are authoritative.

A.3 Non-Inferiority Tests for Success Rates

For the main comparison of PACE against single-step PPO, we report per-seed paired differences in success rate. Both arms use the same batch of 32 frozen validation seeds and deploy on the same batch of 256 environments with the same evaluation random seed, so for each seed i the paired difference $d_i = S_i^{\mathrm{PACE}} - S_i^{\mathrm{PPO}}$ can be computed; seeds and evaluation instances are shared between the two arms, so the variance due to instance difficulty cancels in the subtraction. Table 4 lists the mean difference, its one-sided 95% paired-t lower bound $L = \bar{d} - t_{0.95,31} \cdot \mathrm{SE}$, and whether L lies above $-\epsilon$ for ϵ=0.01 and ϵ=0.02; we call a method's success rate similar to PPO's when $L > -\epsilon$ at ϵ=0.02.

Table 4: Non-inferiority tests for success rates. n=32 paired seeds; differences are deployment success rates, PACE−PPO; "per-seed signs" are the positive/zero/negative counts of the paired differences d_i, a raw statistic independent of the choice of ϵ.

Task	Paired diff.	Diff. sd	Signs $+/0/-$	95% lower bound L	$L > -0.01$	$L > -0.02$
DK5	$+0.0000$	0.0000	$0/32/0$	0.0000	yes	yes
DK8	$+0.0039$	0.0332	$10/12/10$	-0.0061	yes	yes
DO6-N3P2	-0.0094	0.0207	$9/6/17$	-0.0156	no	yes
SC11-N5	-0.0010	0.0182	$16/0/16$	-0.0064	yes	yes

A.4 Training-Budget Axis and Tail-Window Convergence Disclosure

Training budgets for all arms of Experiment 2 are counted in environment primitive frames; per-level budgets are given in Section 7 of the main text. To rule out the possibility that any arm's conclusion is depressed by budget truncation, Table 5 reports each arm's tail-window convergence: for each seed, take its final success, the mean success rate over the last 10% of evaluation points on the training evaluation curve, subtract the mean over the preceding 10% of points, then average over seeds. Almost every arm moves by less than ± 0.6 percentage points between the two windows, and the four that move more do so within the plateau or downwards, which budget truncation cannot

produce. Arms whose window difference is exactly 0 stayed at zero throughout rather than being cut short, and the room-chain *tabula-rasa* arms *L2–L5* belong to this class and are not listed.

Table 5: Tail-window convergence disclosure for Experiment 2 (16 seeds per arm; window difference = mean success rate over the last 10% of evaluation points minus the preceding 10% window, computed per seed then averaged over seeds; ± is the between-seed standard deviation, in percentage points).

Task level	Arm	final_success	Window diff. (pp)
DK ladder, first-person view			
DK8	Actions	0.990±.002	+0.34±1.07
DK10	Actions	0.973±.003	+0.29±0.30
DK10	Actions and Curriculum	0.974±.004	−0.04±0.44
DK10	Actions and Discovered options	0.978±.004	+0.04±0.39
DK10	Actions and Random options	0.909±.235	+0.28±0.64
DK12	Actions	0.793±.300	+0.82±1.13
DK12	Actions and Curriculum	0.937±.009	+0.11±0.80
DK12	Actions and Discovered options	0.949±.013	+0.19±0.78
DK12	Actions and Random options	0.247±.396	+2.16±6.29
DK14	Actions	0.554±.375	+0.58±1.34
DK14	Actions and Curriculum	0.850±.014	−0.22±1.49
DK14	Actions and Discovered options	0.882±.019	−0.19±1.59
DK14	Actions and Random options	0.000±.000	+0.00±0.00
DK16	Actions	0.251±.325	+0.34±1.76
DK16	Actions and Curriculum	0.726±.017	+0.72±1.83
DK16	Actions and Discovered options	0.885±.030	+0.18±2.35
DK16	Actions and Random options	0.000±.000	+0.00±0.00
Room chain, occluded first-person view			
L2	Curriculum	0.939±.009	+0.35±0.88
L2	Discovered options + *Curriculum*	0.955±.010	+0.47±0.66
L3	Curriculum	0.350±.452	+0.30±1.12
L3	Discovered options + *Curriculum*	0.935±.010	+0.49±1.24
L4	Curriculum	0.337±.435	−0.11±0.86
L4	Discovered options + *Curriculum*	0.914±.017	+0.05±0.62
L5	Curriculum	0.303±.392	−0.73±1.74
L5	Discovered options + *Curriculum*	0.850±.022	+0.25±2.08

B Training and Deployment Pseudocode

Algorithm 1 lists one PACE training iteration and the deployment loop, with the losses of Section 4. The rollout, advantage estimation, and PPO update are those of single-step PPO; PACE adds, on the same batch, the forward passes that produce the distillation targets and the agreement labels. At deployment, since c_j is non-increasing, the rule $h = 1 + \#\{j : c_j \geq \tau\}$ stops at the first position whose confidence falls below τ; the h actions are executed open-loop, then the agent reads the next state and initiates the next option. Appendix E checks robustness over τ and $h_{\max}$.

Algorithm 1 PACE: one training iteration, and deployment

Require: base policy $\pi(\cdot \mid s;\theta_a)$, option policy π_Ω with encoder, codebook $\mathcal{E}$ and decoder (θ_ω), termination function $\beta(\cdot \mid s;\theta_h)$, window $h_{\max}$, coefficient λ_{d}, threshold τ

Training iteration

Collect a rollout with $a_t \sim \pi(\cdot \mid s_t;\theta_a)$ and compute $\hat{A}_t$ as in PPO
for all windows $(s_t,\ a_{t:t+h_{\max}-2},\ s_{t+1:t+h_{\max}-1})$ in the rollout **do**
  $e_t^* \leftarrow$ nearest codebook entry to $f^{\mathrm{enc}}(s_t)$ ▷ Eq. (2)
  for $j = 1, \ldots, h_{\max}-1$ **do**
    $\mu_j \leftarrow \mathbf{1}[\text{window stays inside the trajectory and the episode up to } t{+}j]$
    $p_j \leftarrow \mathrm{sg}\big[\pi(\cdot \mid s_{t+j};\theta_a)\big]$ ▷ teacher at the reached state
    $\hat{p}_j \leftarrow \pi_\Omega^j(\cdot \mid s_t, a_{t:t+j-1};\theta_\omega)$ ▷ decoder on e_t^*, executed prefix, position j
    $\alpha_j \leftarrow \mathbf{1}[\arg\max \hat{p}_j = \arg\max p_j]$ ▷ agreement; $\alpha_j \leftarrow 1$ if the goal was reached before $t{+}j$
    $m_j \leftarrow \mu_j \prod_{i<j} \alpha_i$ ▷ at-risk mask
  end for
  $b \leftarrow \sigma\big(\beta(s_t;\theta_h)\big)$ ▷ first-disagreement probabilities $b_1, \ldots, b_{h_{\max}-1}$
end for
$\mathcal{L}_{\mathrm{PPO}}(\theta_a)$ on $(s_t, a_t, \hat{A}_t)$; $\mathcal{L}_{\mathrm{VQ}}(\theta_\omega)$ on (z_t, e_t^*)
$\mathcal{L}_{\mathrm{distill}}(\theta_\omega) \leftarrow \sum_j \mu_j \,\mathrm{KL}(p_j \,\|\, \hat{p}_j) \,/\, \max(1, \sum_j \mu_j)$
$\mathcal{L}_{\mathrm{term}}(\theta_h) \leftarrow -\sum_j m_j \big[(1-\alpha_j)\log b_j + \alpha_j \log(1-b_j)\big] / \max(1, \sum_j m_j)$
Update $\theta_a, \theta_\omega, \theta_h$ on $\mathcal{L}_{\mathrm{PPO}} + \mathcal{L}_{\mathrm{VQ}} + \lambda_{\mathrm{d}}\mathcal{L}_{\mathrm{distill}} + \mathcal{L}_{\mathrm{term}}$, each on its own group

Deployment

while episode not over **do**
  read s_t; $\omega_0 \leftarrow \arg\max_a \pi(a \mid s_t;\theta_a)$; $b \leftarrow \sigma(\beta(s_t;\theta_h))$; $c_j \leftarrow \prod_{i\le j}(1-b_i)$
  $h \leftarrow 1 + \#\{j :\ c_j \ge \tau\}$
  for $j = 1, \ldots, h-1$ **do** $\omega_j \leftarrow \arg\max_a \pi_\Omega^j(a \mid s_t, \omega_{0:j-1};\theta_\omega)$
  end for
  execute $\omega_0, \ldots, \omega_{h-1}$ open-loop; $t \leftarrow t + h$
end while

C Implementation and Porting of the Baseline Methods

All baseline methods share the same training stack as PACE (running obs/reward normalisation, swish MLP encoder, orthogonal initialisation, GAE, the PPO clipped objective, deterministic greedy evaluation) and go through exactly the same hyperparameter sweep and deployment evaluation protocol (Appendix D); all differences between methods are concentrated in the multi-step execution mechanism itself. Budgets match those of Section 5.2 with one exception: TempoRL and DAR were trained on $4\times$ the frames given to PACE on *DK5* and on $3.2\times$ elsewhere, so that both are reported at a budget where their training curves have stopped improving. Fixed and SEAR train with a window of $h{=}10$; SEAR is deployed at every fixed length from 1 to 10 and the best is reported: $h^*{=}9$, 7, 1 and 10 on *DK5*, *DK8*, *DO6-N3P2* and *SC11-N5*. PACE uses the confidence threshold $\tau{=}0.9$ in Table 1; the stochasticity ladder of Section 6.2 calibrates τ per point. This section describes, in order of appearance in the main text, how each of the five baseline methods is implemented or ported in this project.

C.1 PPO: The Single-Step Base

Standard clipped PPO (Schulman et al., 2017): the single-step policy $\pi(a \mid s)$ is trained directly on the primitive action space with the clipped objective and GAE advantages, and deployed greedily step by step, one observation per primitive step. It serves throughout the paper as both the success-rate floor and the steps-to-goal reference; PACE's base policy is trained with exactly the same objective and data, so all comparisons take it as the common starting point.

C.2 FIXED: FIXED-LENGTH AUTOREGRESSIVE CHUNKED PPO

The actor replaces the single-step head with an autoregressive action window of length h, $\prod_{j=0}^{h-1} \pi_\Omega^j(\omega_j \mid s_t, \omega_{0:j-1})$, and each option executes the whole window open-loop (truncated at episode termination; the actual executed length is denoted ℓ). One window constitutes one SMDP transition: the segment return is $\sum_{i=0}^{\ell-1} \gamma^i r_{t+i}$, the value bootstrap is contracted by γ^ℓ, and GAE is computed on the chunk-level sequence; the PPO importance ratio uses the joint log-probability of the executed positions, $\sum_{j<\ell} \log \pi_\Omega^j$, and the entropy is the mean over executed positions. Reward normalisation runs inside the window at the primitive-step level, matching single-step PPO step by step, so the normalisation statistics do not drift because of chunking. At deployment, one observation per chunk.

C.3 SEAR: RANDOMLY REPLANNED CHUNKING WITH DEPLOYMENT-TIME LENGTH SELECTION

It uses the same autoregressive chunk actor as Fixed, but during training each option executes only a random prefix, its length drawn uniformly from $\{1, \ldots, h\}$ (random replanning), so that all prefix lengths get trained. The core of SEAR (Nagy et al., 2026) is value-target densification: the original paper trains one causal-transformer critic that, at the same initiation state, outputs a value for every prefix of the chunk, each regressed on its own multi-horizon target. In our on-policy state-value setting, the values of all horizons share the same fixed point V^π, so we port it in a single-head equivalent form: for a chunk with actual executed length ℓ, construct for each visited intermediate state s_{t+j} ($j = 1, \ldots, \ell - 1$) the n-step target to the end of the chunk

$$G_j = \sum_{i=j}^{\ell-1} \gamma^{\,i-j}\, r_{t+i} + \gamma^{\,\ell-j}\,(1 - \text{done})\, V(s_{t+\ell}), \tag{10}$$

and add $\sum_j \big(V(s_{t+j}) - \text{sg}[G_j]\big)^2$ to training as an auxiliary value loss (targets are snapshots taken at collection time). The number of targets obtained per chunk transition matches SEAR; the difference is that SEAR places targets of different horizons on the prefix outputs of one critic at the same anchor state, whereas this variant places single-horizon targets on the same head at each intermediate state, so the critic's state distribution differs, making this an analogous port rather than a verbatim replication. At deployment, a grid search over the global truncation length $h^* \in \{1, \ldots, h\}$ is run under the evaluation protocol and the best value is reported, which amounts to granting it a post-hoc length-selection advantage; all SEAR numbers in the main text are results at that best h^*.

Why not port the original. The original SEAR works in the dense-reward environments of Meta-World. We made a faithful port of its SAC stack; on this paper's online tasks with sparse success rewards it does not reach the goal, with returns at zero even on the smallest *DK5*; the main comparison therefore uses the single-head variant described above, ported to the shared PPO stack. The Q-chunking family (Li et al., 2025; Chen et al., 2026) relies on offline-demonstration warm-up, which differs from this paper's setting, so it is not compared.

C.4 TEMPORL: ACTION-CONDITIONED LEARNED SKIPS

Following the original idea of TempoRL (Biedenkapp et al., 2021), we adapt it from its value-function stack onto the same PPO stack as PACE. Each option is formed in two steps: first sample a primitive $a \sim \pi(a \mid s)$ (a 7-way action head), then sample a skip length $h \sim \pi(h \mid s, a)$ (a J-way skip head, J=10 aligned with $h_{\max}$); the skip head reads the concatenation of shared features and the chosen action's embedding; this action conditioning is exactly what separates TempoRL from action-agnostic repetition. Then a is repeated open-loop for h steps (stopping at episode termination; actual length ℓ). The option's log-probability is $\log \pi(a \mid s) + \log \pi(h \mid s, a)$, and the two heads are trained jointly by the same PPO clipped objective on SMDP transitions (bootstrap contracted by γ^ℓ). Skips unroll against the underlying environment inside the agent; the action head stays 7-way and the action space does not inflate with J; each option consumes one observation read. The learned per-action skips are self-limiting: for one-shot actions such as pickup and door toggling they learn h=1, with no need to hand-specify which actions are repeatable.

C.5 DAR: A Repeat-Action Space

An on-policy implementation of FiGAR/DAR-style action repetition (Sharma et al., 2017; Lakshminarayanan et al., 2017): plain PPO with the 15 repeat options $\{\text{turn left, turn right, forward}\} \times \{2, 4, 6, 8, 10\}$ added to the action space alongside the 7 primitives (22-way). Options unroll open-loop inside the agent (stopping at episode termination) and are discounted by γ^{ℓ} under SMDP semantics (the union semantics of Section 3); everything else is identical to single-step PPO. The machinery is implemented exactly as the mechanism that plugs the selected options into the action space in Experiment 2; only the set of options differs. Whether primitive or multi-step, each option consumes one observation read.

Meaningful-action accounting (Section 6.1). DAR's 22-way action set pruned to 14 meaningful actions (7 primitives, left/right turn $\times 2$, forward $\times\{2, 4, 6, 8, 10\}$) has mean length $41/14 \approx 2.9$; TempoRL samples an action first and then a skip length uniformly over its meaningful range, giving $\frac{1}{7}(2 \cdot 1.5 + 5.5 + 4 \cdot 1) \approx 1.8$. Both estimates are conservative: each one-shot action appears at most once on the optimal route, so their uniform weight of $4/7$ overstates their share, and the commitment lengths actually available to the optimal policy are longer than these estimates.

The two adaptive baselines (TempoRL, DAR) go through exactly the same sweep protocol as the other methods (24 configurations $\times$ 3 seeds $\rightarrow$ 32-seed validation; the only method-specific hyperparameters are J and the option set, both fixed); their training budgets are given in Appendix C.

D Protocol, Hyperparameters, and Champion Configurations

D.1 Room types and examples

Table 6: Room types at a glance (examples in Figure 4).

Room type	Mechanism
DoorKey (DKN)	pick up the key→open the door→reach the goal, with reward only at the final goal; N is the map side length
Dynamic-Obstacles (DON-NxPy)	x moving obstacles, each moves once every y steps; touching one terminates the episode
SimpleCrossing (SCN-Nw)	procedurally generated maze: w candidate wall lines are placed at random, then passages are randomly carved through them
Obstruction room	the passage is blocked by a movable obstacle; the agent must pick it up and move it aside to pass

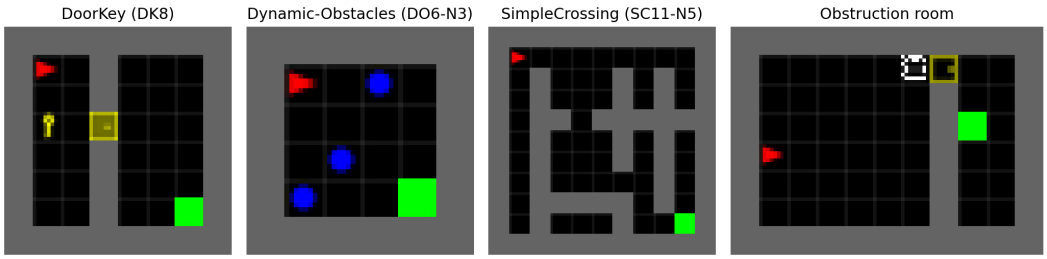

Figure 4: Example rooms (matching Table 6 row by row). In the obstruction room, the white box next to the exit door is the obstacle that must be moved aside.

D.2 Layouts and Observation Settings

Rooms are $N \times N$ grids; the oriented agent has the $|\mathcal{A}| = 7$ MiniGrid primitives. Fixed layout randomises only internal structure per episode; fully random re-places start, goal, and all structure.

Full observation (full): whole grid, per-cell symbolic type, color, state. First-person view + pocket (fpvp): $7\times7\times3$ symbolic window, sight unobstructed, plus a 1-dimensional carried-object pocket channel. Occluded first-person view (fpo): 5×5 window plus pocket channel; walls and closed doors block sight. Experiment 1 uses full observation, which separates observation compression from state estimation; Experiment 2 moves to the first-person settings, where sparse-reward exploration is hardest. No component is shared across settings: each experiment trains and deploys PACE under its own setting and selects its options there, so only the mining procedure carries over.

D.3 Unified Experimental Pipeline

All main comparisons follow the same pipeline: a 24-configuration sweep $\times$ 3 seeds $\rightarrow$ champion selection $\rightarrow$ 32-seed validation $\rightarrow$ deterministic deployment evaluation (256 fresh environments, greedy $\arg\max$, frozen observation-normalisation statistics, each method evaluated in its own deployment mode); Experiment 2 uses a 16-seed protocol, and the dynamic-obstacle points on the stochasticity ladder use a 6-configuration sweep $\times 3 \rightarrow 16$ validation. Unless stated otherwise, $h_{\max}$=10. Per-episode step limit $T_{\max}$: 100 for *DK5–DK10* and the *Dynamic-Obstacles* tasks, 144 for *DK12*, 200 for *SC11-N5*; in Experiment 2, 196 for *DK14* and 256 for *DK16*, and for the *Room Chain* 64 at *L1*, 96 at *L2*/*L3*, 144 at *L4*, 192 at *L5*. The backbone is a rejax-equivalent PPO (running obs+reward normalisation, standard GAE, swish, orthogonal initialisation), shared by all methods; only the multi-step execution mechanism differs. The internal run codes of the environment points on the stochasticity ladder are, in order: *DO6-N1-static* = `fn1s`, *DO6-N1P2* = `fn1`, *DO6-N2P2* = `fn2`, *DO6-N3P2* = `fn3`, *DO6-N4P2* = `fn4`, *DO6-N3P3* = `fn3p3`, *DO6-N2P1* = `fn2p1`, *DO6-N3P1* = `fn3p1` (internal codes omit the default period P2).

D.4 Hyperparameter Sweep Grid

Each sweep's configuration set starts from 3 named configurations (`rejax`, `navix-default`, `high-explore`) and is filled up to 24 by sampling without replacement from the Cartesian product in the table below (6 under the reduced protocol); each configuration runs 3 seeds, the champion is selected by end-of-training deterministic success rate, and more seeds are then run for validation.

Hyperparameter	Values
Learning rate `lr`	{2.5e-4, 5e-4, 1e-3}
Entropy coefficient `ent_coef`	{0.01, 0.05, 0.1}
GAE λ	{0.9, 0.95, 0.99}
Clipping `clip_eps`	{0.1, 0.2}
Number of minibatches	{8, 32}
Number of epochs	{2, 4}
Learning-rate annealing `anneal_lr`	{yes, no}
Discount γ (applied in the environment)	{0.95, 0.99}

D.5 PACE-Specific Hyperparameters

Training window $h_{\max}$=10, base-policy step length h=1; codebook size M=16, code dimension d_z=16, commitment coefficient λ_{c}=0.25, quantisation auxiliary-loss weight λ_{VQ}=1.0; distillation weight λ_{d}=0.1 (`distill_coef`), with λ_{d} ramped linearly from 0 to its target value over the first 50% of frames (`distill_ramp_frac`= 0.5, letting the base policy stabilise first); confidence threshold τ=0.9. The termination function's BCE is gradient-isolated from the base policy and the option body, so it does not participate in the ramp. Each branch's encoder is a 2-layer swish MLP (hidden 64); the action embedding and the position embedding are 32-dimensional each, used only by the option-body branch and absent from the base-policy branch.

D.6 Champion Configurations for All Main-Text Experiments

Table 7 lists, by main-text experiment group, the sweep-winning configuration of every run that involves new training; hyperparameter meanings follow the grid in the previous subsection.

Table 7: Champion hyperparameter configurations for all main-text experiments of Experiment 1 (each row = the winning configuration of one sweep, taken from each experiment's validation CSV; "Config" is the sweep-internal configuration name, kept for traceability). "Budget" is the arm's training budget in primitive frames. Experiments not listed here involve no new training: the τ scan only sweeps the threshold on already-trained checkpoints, the forced-h scan directly reuses the checkpoints of Table 1, and the DO6-N3P2 point of the stochasticity ladder reuses the two arms of the main comparison and is not listed separately. All arms of Experiment 2 (Section 7) share a single configuration rand14 in their 16-seed confirmation runs (lr 10^{-3}, ent 0.05, λ=0.9, clip 0.2, mb 8, ep 2, no anneal, γ=0.99), with no per-arm differences, so they are not listed as separate rows.

Task	Method	Config	Budget	lr	ent	λ	clip	mb	ep	Ann.	γ
Main comparison (Table 1)											
DK5	PPO	rand22	1.0M	1e-3	0.05	0.95	0.2	32	4	yes	0.95
DK5	Fixed	rand11	1.0M	1e-3	0.01	0.95	0.2	32	4	yes	0.95
DK5	SEAR	rand11	1.0M	1e-3	0.01	0.95	0.2	32	4	yes	0.95
DK5	TempoRL	rand8	4.2M	5e-4	0.01	0.9	0.1	8	4	no	0.99
DK5	DAR	rand11	4.2M	1e-3	0.01	0.95	0.2	32	4	yes	0.95
DK5	PACE	rand22	1.0M	1e-3	0.05	0.95	0.2	32	4	yes	0.95
DK8	PPO	rand14	10.5M	1e-3	0.05	0.9	0.2	8	2	no	0.99
DK8	Fixed	rand14	10.5M	1e-3	0.05	0.9	0.2	8	2	no	0.99
DK8	SEAR	rand14	10.5M	1e-3	0.05	0.9	0.2	8	2	no	0.99
DK8	TempoRL	rand8	33.6M	5e-4	0.01	0.9	0.1	8	4	no	0.99
DK8	DAR	rand8	33.6M	5e-4	0.01	0.9	0.1	8	4	no	0.99
DK8	PACE	rand20	10.5M	2.5e-4	0.05	0.95	0.1	32	4	no	0.99
DO6-N3P2	PPO	rand20	10.5M	2.5e-4	0.05	0.95	0.1	32	4	no	0.99
DO6-N3P2	Fixed	rand16	10.5M	1e-3	0.1	0.9	0.1	32	2	no	0.95
DO6-N3P2	SEAR	rand6	10.5M	5e-4	0.1	0.95	0.1	32	4	no	0.95
DO6-N3P2	TempoRL	rand14	33.6M	1e-3	0.05	0.9	0.2	8	2	no	0.99
DO6-N3P2	DAR	rand14	33.6M	1e-3	0.05	0.9	0.2	8	2	no	0.99
DO6-N3P2	PACE	rand8	10.5M	5e-4	0.01	0.9	0.1	8	4	no	0.99
SC11-N5	PPO	rand8	5.2M	5e-4	0.01	0.9	0.1	8	4	no	0.99
SC11-N5	Fixed	rand3	5.2M	1e-3	0.05	0.95	0.1	8	2	yes	0.95
SC11-N5	SEAR	rand8	5.2M	5e-4	0.01	0.9	0.1	8	4	no	0.99
SC11-N5	TempoRL	rand8	16.8M	5e-4	0.01	0.9	0.1	8	4	no	0.99
SC11-N5	DAR	rand8	16.8M	5e-4	0.01	0.9	0.1	8	4	no	0.99
SC11-N5	PACE	rand8	5.2M	5e-4	0.01	0.9	0.1	8	4	no	0.99
k scan (Fig. 5; k=10 arms as above)											
DK8	Fixed k=3	rand8	10.5M	5e-4	0.01	0.9	0.1	8	4	no	0.99
DK8	Fixed k=5	rand14	10.5M	1e-3	0.05	0.9	0.2	8	2	no	0.99
DK8	SEAR k=3	rand14	10.5M	1e-3	0.05	0.9	0.2	8	2	no	0.99
DK8	SEAR k=5	rand14	10.5M	1e-3	0.05	0.9	0.2	8	2	no	0.99
DO6-N3P2	Fixed k=3	rand14	10.5M	1e-3	0.05	0.9	0.2	8	2	no	0.99
DO6-N3P2	Fixed k=5	rand20	10.5M	2.5e-4	0.05	0.95	0.1	32	4	no	0.99
DO6-N3P2	SEAR k=3	rand4	10.5M	2.5e-4	0.1	0.95	0.2	32	4	no	0.99
DO6-N3P2	SEAR k=5	rand23	10.5M	1e-3	0.1	0.95	0.1	8	4	yes	0.99
Design ablations (Appendix E)											
DK8	Marginal BCE	rand20	10.5M	2.5e-4	0.05	0.95	0.1	32	4	no	0.99
DK5	Marginal BCE	rand22	1.0M	1e-3	0.05	0.95	0.2	32	4	yes	0.95
DO6-N3P2	Marginal BCE	rand8	10.5M	5e-4	0.01	0.9	0.1	8	4	no	0.99
DK8	K=5, pruning	rand20	10.5M	2.5e-4	0.05	0.95	0.1	32	4	no	0.99
DK8	K=20, pruning	rand20	10.5M	2.5e-4	0.05	0.95	0.1	32	4	no	0.99
Stochasticity ladder (Fig. 1)											

Task	Method	Config	Budget	lr	ent	λ	clip	mb	ep	Ann.	γ
DO6-N1-static	PPO	rand5	10.5M	5e-4	0.05	0.95	0.2	8	4	no	0.95
DO6-N1-static	PACE	rand14	10.5M	1e-3	0.05	0.9	0.2	8	2	no	0.99
DO6-N1P2	PPO	`high-explore`	10.5M	5e-4	0.1	0.95	0.2	8	4	no	0.99
DO6-N1P2	PACE	rand14	10.5M	1e-3	0.05	0.9	0.2	8	2	no	0.99
DO6-N3P3	PPO	rand8	10.5M	5e-4	0.01	0.9	0.1	8	4	no	0.99
DO6-N3P3	PACE	rand8	10.5M	5e-4	0.01	0.9	0.1	8	4	no	0.99
DO6-N2P2	PPO	rand8	10.5M	5e-4	0.01	0.9	0.1	8	4	no	0.99
DO6-N2P2	PACE	rand14	10.5M	1e-3	0.05	0.9	0.2	8	2	no	0.99
DO6-N4P2	PPO	rand8	10.5M	5e-4	0.01	0.9	0.1	8	4	no	0.99
DO6-N4P2	PACE	rand18	10.5M	5e-4	0.05	0.99	0.1	32	2	no	0.99
DO6-N2P1	PPO	rand4	10.5M	2.5e-4	0.1	0.95	0.2	32	4	no	0.99
DO6-N2P1	PACE	rand14	10.5M	1e-3	0.05	0.9	0.2	8	2	no	0.99
DO6-N3P1	PPO	`high-explore`	10.5M	5e-4	0.1	0.95	0.2	8	4	no	0.99
DO6-N3P1	PACE	rand20	10.5M	2.5e-4	0.05	0.95	0.1	32	4	no	0.99

D.7 Reproducibility Essentials and Extra Cost

This subsection reports a detailed parameter and computation-cost accounting for the small-scale networks the experiments build on. All three subnetworks (base policy θ_a, option body θ_ω, termination function θ_h) are 2-layer swish MLPs (hidden 64). Exact parameter counts (per module, for *DK8*'s 192-dimensional observation and $h_{\max}$=10): single-step PPO totals 33,544 (actor encoder+head 16,967, critic 16,577); PACE totals 73,570 (2.2×), of which the base-policy branch is 16,967 (identical in structure to the PPO actor), the option-body branch 22,279, the termination function 17,097, and the critic 16,577 (used in training only). What is actually loaded at deployment is PPO 16,967 versus PACE 56,343 (3.3×). In fp32, the full parameters of every method on every task stay under 0.5 MB. The training budget is fixed per task and listed arm by arm in the "Budget" column of Table 7: *DK5* uses 1.0M (2^{20}) primitive frames (64 parallel environments × 64-step rollouts), *DK8* and the *DO6* series use 10.5M, *SC11-N5* uses 5.2M, with the same budget, the same backbone, and the same sweep scale as all baseline methods (the 3.2–4× budgets of TempoRL/DAR are given in Appendix C), so that the comparison stays fair. Budgets are always counted in environment primitive frames: one option of a multi-step method unrolls multiple primitive frames internally, and the frames actually unrolled are charged to the budget; the total environment interaction of every method is therefore strictly comparable under one accounting. Per-sample forward computation during training grows linearly with $h_{\max}$.

Total parameter counts for the six methods are given in Table 8. By parameter count, the extra heads of the four multi-step baselines add about 2%–54% over single-step PPO, and PACE sits at 2.1×–2.3× (absolute scale again < 0.5 MB in all cases): the three branches keeping mutually independent parameters (Section 4) is where the extra parameters come from. The same table also shows that parameters by themselves do not buy performance: the cheapest method, DAR (only +2%–5%), and the most expensive, TempoRL (+51%–54%), fail alike on the hard tasks (Table 1); what PACE's extra parameters buy is mutually non-interfering gradient channels, not capacity itself. Whether those extra parameters are what the advantage rests on is tested directly on the *DoorKey Ladder* in Appendix F, Table 12.

Table 8: Total parameter count for the six methods on each task, counting all training-time parameters (biases, embeddings, and the critic used only during training).

	Total parameters			
Method	DK5	DK8	DO6	SC11
PPO	18,568	33,544	22,792	55,432
Fixed	23,880	38,856	28,104	60,744
SEAR	24,530	39,506	28,754	61,394
TempoRL	28,514	50,978	34,850	83,810
DAR	19,543	34,519	23,767	56,407
PACE	43,618	73,570	52,066	117,346

E Design-Choice Ablations

This section collects the supporting figures and tables for the ablations of Section 6, together with the remaining checks at the level of variants and sensitivity.

E.1 Hardware and Wall-Clock

All experiments run on NVIDIA RTX 4090 D (24 GB) GPUs under JAX. Representative single-seed training wall-clock (re-measured on an RTX 5090 with the same stack; *DK8*, full observation, compile time excluded): single-step PPO (10.5M frames) 11 s, Fixed 15 s, SEAR 46 s, PACE 132 s; TempoRL and DAR take about 215 s each at their $3.2\times$ budget (33.5M frames). Selecting options from 16 pooled deployment seeds takes about 3 minutes on CPU.

E.2 Chunk-Length Degradation Scan for Fixed-Length Chunking

This is the full evidence for the tension described in the design paragraph of Section 6.1: on *DK8* and *DO6-N3P2* we fix everything else and vary only the training chunk length $h \in \{3, 5, 10\}$, evaluating Fixed and SEAR (Figure 5). First, longer chunks train worse: on both tasks the success rates of both fixed-length schemes decrease monotonically with h and never exceed single-step PPO; the collapse is faster on *DK8*, where SEAR/Fixed drop from $0.999/0.951$ at h=3 to $0.51/0.03$ at h=10, and post-hoc selection of the best global length does not stop it. Second, SEAR degenerates completely on the stochastic task: on *DO6-N3P2* it selects h^*=1 at every h, that is, its best post-hoc choice is not to act open-loop at all. Successful episodes also pay in route: on *DK8* the steps to goal of both schemes grow with h; relative to PPO's 16.7 steps to goal, Fixed pays $+35\%/+96\%/+174\%$ at h=3/5/10 and SEAR pays $+16\%/+68\%/+131\%$; the longer the open-loop commitment, the longer the detour.

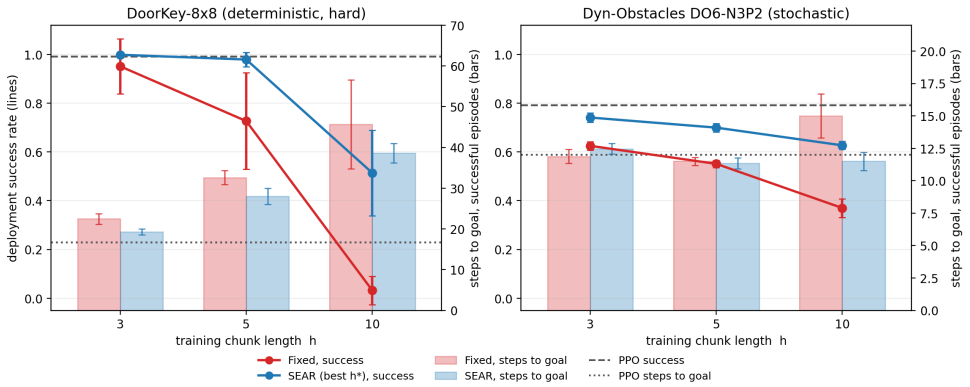

Figure 5: Degradation of fixed-length chunking with chunk length h (2 tasks $\times$ 2 algorithms, 32-seed mean$\pm$std; within a task and algorithm only the chunk length varies). Lines (left axis): deployment success rate; bars (right axis): steps to goal of successful episodes; horizontal dashed/dotted lines are the single-step PPO references for success rate and steps to goal. h is the training chunk length; Fixed deploys at its training length, SEAR deploys the best searched $h^* \leq h$ (*DK8*: h^*=3/5/7 at h=3/5/10; *DO6-N3P2*: h^*=1 at every h).

E.3 Operating Points of the Main Comparison

Table 1 reports success and observations as two columns, which leaves the trade-off between them to the reader. Figure 6 plots the same numbers as one point per method and task, so that the methods that hold PPO's success rate can be told apart from those that buy their observation savings with success. No new runs enter the figure.

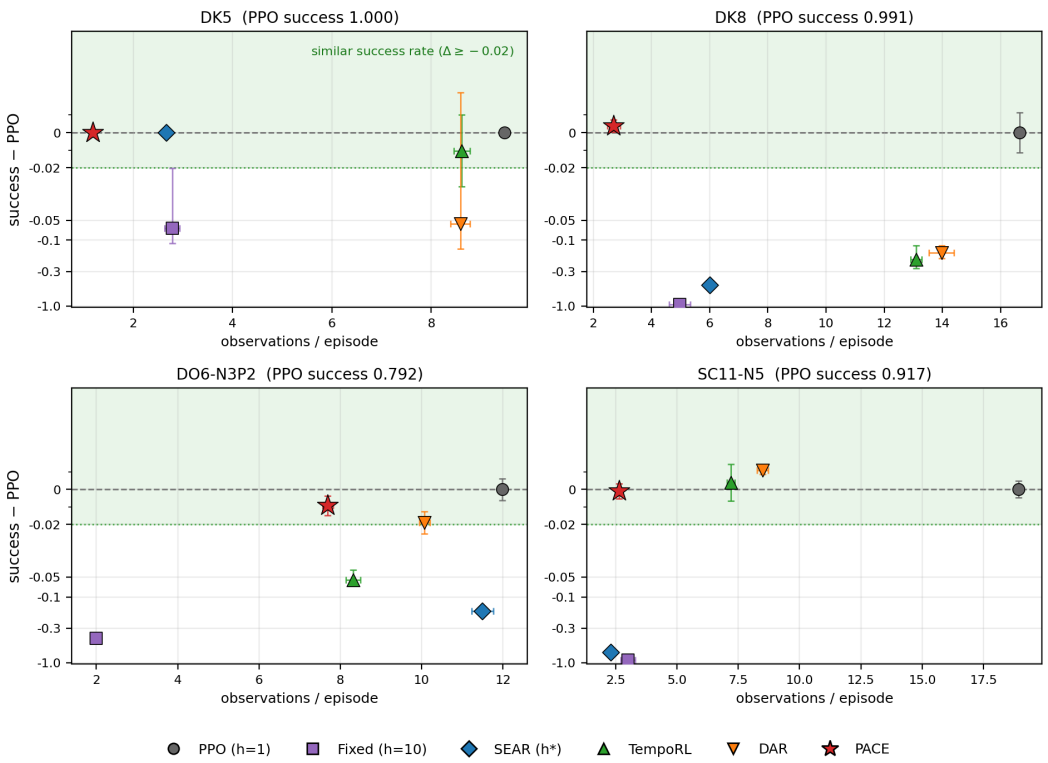

Figure 6: Graphical rendering of the operating points in Table 1: the vertical axis is the success-rate difference Δ relative to PPO, the horizontal axis is observations per episode, the green region marks a similar success rate ($\Delta \geq -\epsilon$, ϵ=0.02), and error bars are the 95% confidence interval of the seed mean. The data are the same as in Table 1, with no new measurements.

E.4 NECESSITY OF TERMINATION: FORCED FIXED-h SCAN

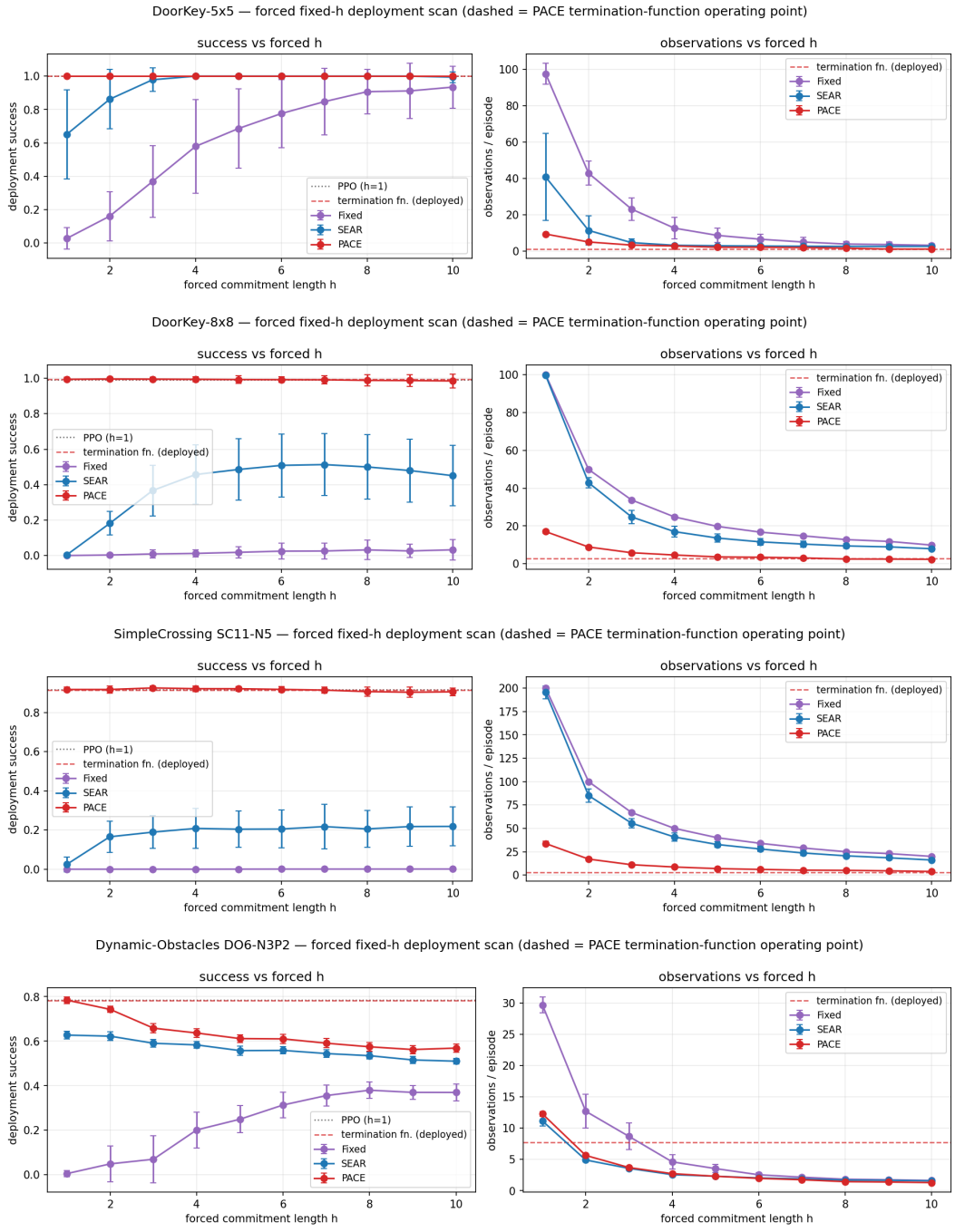

Figure 7: Fixed-length deployment scans, top to bottom *DK5*, *DK8*, *SC11-N5* and *DO6-N3P2* (Figure 2 in the main text averages the success panels of the four tasks); dashed line = the actual operating point of the termination function. On the deterministic tasks the same PACE policy keeps a high success rate under any forced h, *DK8* ≈ 1.0 and *SC11-N5* 0.90–0.93; whereas Fixed, deployed at the same fixed lengths, stays ≈ 0, and SEAR peaks no higher than $0.51/0.22$. No single fixed h reaches both the success rate and the observation count of the termination function's operating point on all four tasks at the same time.

E.5 Termination Parameterisation: the Basic Marginal-BCE Variant

This variant replaces the mainline survival parameterisation with per-depth independent marginal BCE: it trains the confidence at each depth directly on the prefix-product labels. Because its per-depth confidences are not guaranteed monotone, deployment must explicitly take the leading contiguous run $h = \max\{\, j : c_1, \ldots, c_j \text{ all } \geq \tau \,\}$, so that a case where some middle position falls below the threshold and later positions rise above it again is not mistakenly included in the commitment. On *DK8* the variant reaches $0.996\pm.007$ / $2.79\pm.50$ observations per episode (mainline $0.995\pm.010$ / $2.70\pm.60$; on *DK5*, 1.20 versus the mainline 1.18): the two parameterisations have matched success rates, the mainline saves slightly on observations, and the mainline also has two structural properties, monotonicity by construction and an $\mathbb{E}[h]$ estimate. On the stochastic *DO6-N3P2* the survival parameterisation commits longer (7.69 observations per episode) and marginal BCE is more conservative; the two sit at adjacent operating points on the same trade-off curve.

E.6 Robustness to the Deployment Threshold τ and the Training Window $h_{\max}$

We scan $\tau \in [0.5, 0.99]$ on the trained models (Figure 8, 32 seeds): on deterministic *DK8* the success rate is almost insensitive to τ (0.990–0.995 throughout), and a smaller τ only yields longer commitments and fewer observations (down to 2.26 at τ=0.5); on stochastic *DO6-N3P2*, τ controls the trade-off between success rate and observations: success rate rises with τ and enters a plateau at τ=0.85–0.9 (0.781–0.782; even $\tau\to0.99$ only reaches 0.784), after which the observation count keeps rising ($7.69 \to 9.37$). This explains why the single constant τ=0.9 works across tasks: on the deterministic end any τ is safe, and on the stochastic end 0.9 lands right at the start of the plateau, where a more conservative threshold only adds observations without improving success rate.

For the training window, we retrain on *DK8* with the option body's depth changed from the mainline $h_{\max}$=10 to 5 and 20 (16 seeds each, τ=0.9). Deployment success rate is independent of $h_{\max}$: 0.998 ± 0.004 at $h_{\max}$=5, 0.995 ± 0.010 at $h_{\max}$=10, and 0.992 ± 0.010 at $h_{\max}$=20, all within the noise band of PPO (0.991) (Table 9). Mean length at initiation grows with $h_{\max}$ ($4.6 \to 7.5 \to 10.1$): at $h_{\max}$=5 the commitments press against the window cap, showing that the predictable segment length in this task exceeds the small window; at $h_{\max}$=20 the savings saturate ($3.78 \to 2.70 \to 2.57$ observations per episode). The segment length that can safely run open-loop from each state is thus set by the environment; enlarging $h_{\max}$ only removes the cap and yields no further compression.[1]

[1] The $h_{\max}$=$5/20$ runs implement gradient clipping per parameter branch (the base policy, the option body, and the termination function are each clipped independently), because under a single shared global clip the gradients of the option body and the termination function crowd out the effective step size of the base policy as $h_{\max}$ grows; with per-branch clipping the base policy's gradients and updates depend only on itself, and base-policy training is independent of $h_{\max}$ by construction. $h_{\max}$=10 is the mainline run of Section 6.1.

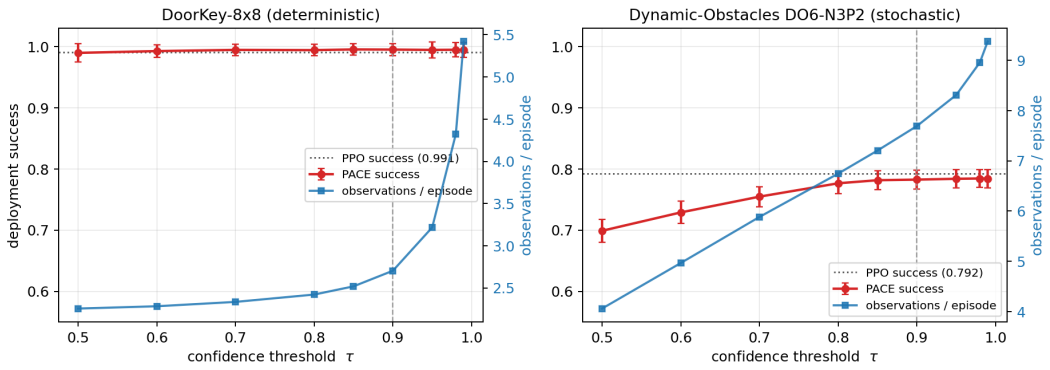

Figure 8: Sensitivity scan of the deployment threshold τ (32 seeds; red = success rate (left axis, error bars are seed std), blue = observations per episode (right axis), dotted line = PPO success rate, vertical dashed line = the paper-wide operating point τ=0.9). Left: *DK8*, where the success rate is almost insensitive to τ; right: *DO6-N3P2*, where the success rate enters a plateau at τ=0.85–0.9 and the observation count keeps rising thereafter. Observation counts follow the same successful-episodes-only convention as Table 1.

Table 9: Detailed deployment readings for the training-window $h_{\max}$ scan (*DK8*, τ=0.9). $h_{\max}$=5/20 use the per-branch clipping implementation (16 seeds, see footnote), $h_{\max}$=10 is the mainline run (32 seeds); success rate is unconditional, while observations per episode and mean length at initiation count successful episodes only.

$h_{\max}$	Seeds	Success ↑	Observations/episode ↓	Length at initiation
5	16	0.998±.004	3.78±.29	4.59
10	32	0.995±.010	2.70±.60	7.52
20	16	0.992±.010	2.57±.92	10.06

F Supplementary Material for Experiment 2

This appendix collects the material of Experiment 2 that the main text does not tabulate. Table 10 repeats the *DoorKey Ladder* with all five arms, so the curriculum and the manual options can be read next to the three arms of the main text. Table 11 asks from which level the curriculum should start at *DK16*, running it from the converged parameters of *DK8*, *DK10*, *DK12* and *DK14* in turn. Table 12 asks whether the gap to the discovered options is a matter of network size, by widening the tabula-rasa trunk until its parameter count matches PACE's. Table 13 asks whether the curriculum at *DK16* can be repaired by intervening on its parameters or on its exploration. Table 14 does the same for the room chain at *L4*, where the controls raise the curriculum's entropy coefficient or replace the options with a random set of the same shape. Table 15 lists the per-level comparisons of the per-seed means, and Section F.1 measures how much of a deployment is actually executed through the options.

Table 10: *DoorKey Ladder*, all five arms (16 seeds; arms as described in Section 7.1, success as in Table 2 in the main text, which carries the actions-only, discovered-options and random-options columns). At *DK16* the curriculum starts from the jump-off level of Table 11. Bold: best per level.

Level	Actions	Actions and Curriculum	Actions and Discovered options	Actions and Manual options	Actions and Random options
DK8	0.990±.002	—	—	—	—
DK10	0.973±.003	0.974±.004	**0.978±.004**	0.977±.003	0.909±.235
DK12	0.793±.300	0.937±.009	**0.949±.013**	0.917±.011	0.247±.396
DK14	0.554±.375	0.850±.014	**0.882±.019**	0.865±.015	0.000±.000
DK16	0.251±.325	0.726±.017	**0.885±.030**	0.850±.018	0.000±.000

Table 11: *DK16* curriculum jump-off fan (16 seeds; success as in Table 2 in the main text). jump$N$ starts *DK16* from the converged parameters of level *DK*N, so jump14 is the ordinary adjacent-rung transfer and the other three skip rungs; the five-arm panel below uses the best of the four, jump14. The per-level panel for all five arms is Table 10; the *DK16* patch controls are Table 13 below.

DK16 curriculum jump-off level	Success
Curriculum jump8	0.705±.050
Curriculum jump10	0.718±.027
Curriculum jump12	0.719±.021
Curriculum jump14	0.726±.017

Table 12: Parameter-matched baseline control on the *DoorKey Ladder* (16 seeds; success as in Table 2, whose actions-only column is reproduced here as the 64-unit arm). Both arms train *tabula-rasa* on primitives under the shared Experiment 2 configuration and differ only in trunk width; 116 units matches PACE's parameter count on this observation (27,912 → 62,648 parameters, 2.24×). Enlarging the parameter budget alone does not bring the baseline near the discovered options: at *DK16* it reaches 0.246, against the options' 0.885 (Table 2).

Task	*Tabula-rasa* (trunk 64, as in Table 2)	*Tabula-rasa* (trunk 116, parameter-matched)
DK8	0.990±.002	0.993±.002
DK10	0.973±.003	0.976±.003
DK12	0.793±.300	0.804±.305
DK14	0.554±.375	0.725±.275
DK16	0.251±.325	0.246±.318

Curriculum patches at *DK16*. The curriculum's exploration does not fail (no seed stays at zero on any level, Table 10); what stands out at *DK16* is how tightly its seeds cluster across all four jump-off levels. We tried parameter-level patches on jump14 (Table 13), addressing the two properties the arm carries into that level, its inherited parameters and its exploration strength: replacing c=0.2 or 0.5 of the inherited parameters with a random initialisation leaves success unchanged; an output-head reset fails half the seeds completely; a threefold entropy coefficient degrades performance overall. None improved on the unpatched curriculum, and the stronger interventions cost the early exploration benefit inheritance provides.

Table 13: Curriculum patch controls at *DK16* (16 seeds, all applied on top of curriculum jump14; measures as in Table 2).

Configuration	Success ↑
Curriculum jump14 (baseline)	$0.726 \pm .017$
+ random-init mixing c=0.2	$0.724 \pm .012$
+ random-init mixing c=0.5	$0.717 \pm .012$
+ output-head reset	$0.365 \pm .365$
+ entropy coefficient 3×	$0.526 \pm .011$

Table 14: Room-chain control groups, all applied at *L4* (occluded first-person view, 16 seeds; success as in Table 3 in the main text). Chain budgets are 10.5M primitive frames for *L2*/*L3* and 21.0M for *L4*/*L5*; the combination arm's options evolve level by level, *L2* using v1, *L3* v2, *L4* v3 and *L5* v4. The per-level panel for the four arms is Table 3 in the main text.

Control group (all on L4)	Success
Curriculum+entropy 3×	$0.000 \pm .000$
Curriculum+entropy 10×	$0.000 \pm .000$
Combination+random options	$0.000 \pm .000$

Table 15: Per-rung statistical tests: two-sided Welch t tests on the per-seed means, 16 seeds.

Comparison	Test	Statistic		p (two-sided)
DK16: discovered options vs curriculum jump14	Welch t	Δ=+0.159, d=6.38	Cohen	2.5×10^{-15}
DK12: discovered options vs manual options	Welch t	Δ=+0.032		2.64×10^{-8}
DK14: discovered options vs manual options	Welch t	Δ=+0.017		8.88×10^{-3}
DK16: discovered options vs manual options	Welch t	Δ=+0.035		5.07×10^{-4}

F.1 Option Usage Share

The share of deployment carried by the options can be measured directly from the deployment logs of Experiment 2. Each arm takes 16 seeds with 256 greedy-deployment episodes per seed, counting successful episodes only; each episode records its observation count and its primitive-frame count, and an option contributes frames equal to its entry length. The open-loop option-frame share is $1 - \text{observations}/\text{primitive frames}$, that is, the share of frames produced open-loop after an option's first frame, requiring no new observation. The logs contain only episode-level aggregates and no per-option tags, so this quantity is a strict lower bound on the share of frames executed through options; the options are 2–4 steps long, so the true share is at most twice this value. Table 16 gives the per-level numbers: the share rises monotonically with DK map size (32.6% → 40.2%); the room-chain levels sit between 36.7% and 43.6%, highest at the last level. The larger the task, the higher the share of deployment executed through the options.

Table 16: Option usage share at deployment (16 seeds $\times 256$ greedy-deployment episodes, each arm acts with the options selected at the previous level; observations and primitive frames are per-episode means, and the $\pm$ on the share is the across-seed standard deviation).

Task (arm)	Obs./ep.	Prim. frames/ep.	Open-loop option-frame share
DK10 (discovered options)	13.3	19.8	$32.6\pm1.9\%$
DK12 (discovered options)	15.7	24.3	$35.3\pm2.5\%$
DK14 (discovered options)	18.8	29.2	$35.4\pm2.6\%$
DK16 (discovered options)	22.0	36.9	$40.2\pm3.2\%$
L2 (combination)	16.8	28.6	$41.2\pm3.1\%$
L3 (combination)	23.2	36.7	$36.7\pm3.1\%$
L4 (combination)	28.3	47.8	$40.8\pm2.7\%$
L5 (combination)	35.6	63.2	$43.6\pm1.9\%$

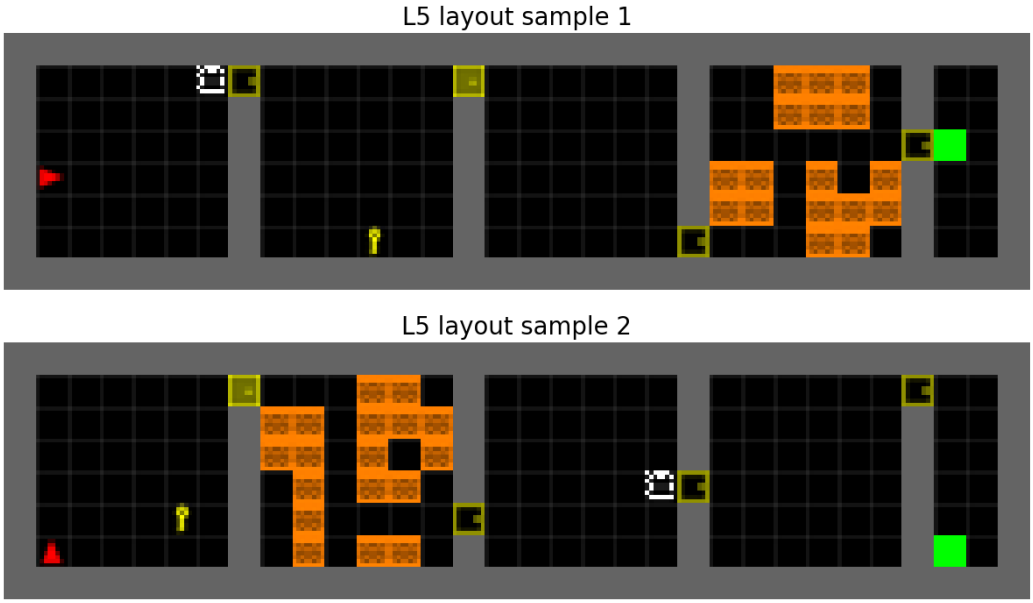

Figure 9: Two layout samples of the *Room Chain*'s final level *L5*, showing the per-episode re-randomisation of the room order (companion to Figure 3 in the main text).

F.2 A Probe on Craftax

We ported PACE onto Craftax-Symbolic as a probe. The baseline is the benchmark's recurrent PPO agent, a 512-unit GRU trained for 10^8 frames with the official hyperparameters, and PACE runs on that same recipe without tuning, reading one observation per option at deployment. The options arm adds one round of selection and incorporation, keeping the eight options with the largest $n_e(\ell_e - 1)$ and counting the budget in observations. Each arm runs three seeds, and the return is the benchmark's raw episode reward.

Table 17: Craftax probe, three seeds per arm. The observation ratio is observations per primitive frame, and compression is the baseline's observations per episode over the arm's, as in Section 6.1. The recurrent PPO baseline reads an observation at every frame, so it has no separate deployment.

Arm	Train ret.	Deploy ret.	Obs./ep.	Obs. ratio	Compr. ↑
PPO (recurrent)	26.16 ± 0.29	—	—	1.000	$1.0\times$
PACE	25.97 ± 0.80	25.82 ± 0.60	296	0.928	$1.1\times$
PACE + options	25.60 ± 0.68	26.18 ± 0.70	252	0.734	$1.4\times$