CHOREOGRAPHER: LEARNING AND ADAPTING SKILLS IN IMAGINATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We present Choreographer, a model-based agent that exploits its world model to learn and adapt skills in imagination. Choreographer is able to learn skills from offline unlabeled data and leverage them for effectively adapting to downstream tasks and for exploring the environment thoroughly, to find sparse rewards. Our method decouples the exploration and skill learning processes, being able to discover skills in the latent state space of the model. For adapting to downstream tasks, the agent uses a meta-controller to evaluate and adapt the learned skills efficiently by deploying them in parallel in imagination.

Project website: `https://doubleblind-repos.github.io/`

1 PROBLEM SETTING

Learning a set of options or skills from unsupervised data can provide an optimal set of behaviors to quickly adapt and generalize to new tasks [9]. Most previous work [8; 28; 12; 1] has focused on learning skills by maximizing their discriminability in the environment. This implies that the skill discovery process is tied to the data collection/exploration process, with the downside that skills have been shown to explore less than other exploration strategies [6; 24; 36].

Another important issue with skill-based methods arises when adapting the skills learned without supervision to downstream tasks: how to exploit the skills in an efficient way, i.e. using the least number of environment interactions? While one could exhaustively test all the options in the environment, this can be expensive with a finite number of skills [8] or intractable, for infinite [22; 28].

Mutual information skill learning. Option discovery can be formalized as learning a skill representation z to condition skill policies $\pi(a|x, z)$. The objective of these policies is to maximize the mutual information between skills and trajectories of environment observations $\tau_x = \{x_t, x_{t+1}, ...\}$:

$$I(\tau_x; z) = H(\tau_x) - H(\tau_x|z) \qquad \textit{// forward form} \tag{1}$$

$$= H(z) \; - H(z|\tau_x) \qquad \textit{// reverse form} \tag{2}$$

The skill policy $\pi(a|x, z)$ results in a family of diversified policies with predictable behavior.

2 CHOREOGRAPHER

As a solution to the problems of skill-based methods, we propose **Choreographer**, an agent able to discover and learn skills offline from unsupervised data. Choreographer leverages a world model to learn skills behaviors in imagination, detaching the exploration and options discovery processes, so that skills can be learned offline. The agent can also efficiently adapt and combine the skills for downstream tasks. It uses its model to evaluate multiple skill behaviors in parallel in imagination, and decides which skills to employ and adapt for downstream tasks using a meta-controller.

An overview of the skill-related models is presented in Figure 1 and an algorithm in Appendix G.

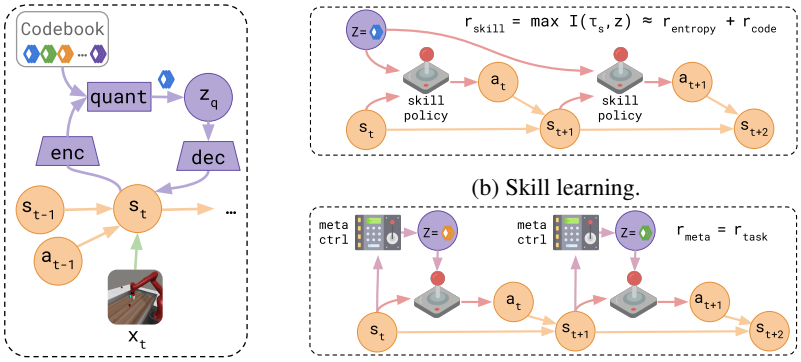
World model. Choreographer learns a task-agnostic world model [13; 16] composed of:

Posterior: $q_\phi(s_t|s_{t-1}, a_{t-1}, x_t)$, Prior: $p_\phi(s_t|s_{t-1}, a_{t-1})$, Reconstruction: $p_\phi(x_t|s_t)$.

The model states s_t have both a deterministic component, modeled using the recurrent state of a GRU [7], and a discrete stochastic component [17]. The remaining modules are multi-layer perceptrons (MLPs) or CNNs for pixel inputs [26]. The model is trained end-to-end by optimizing an evidence lower bound on the log-likelihood of data:

$$\mathcal{L}_{\text{wm}} = D_{\text{KL}}[q_\phi(s_t|s_{t-1}, a_{t-1}, x_t) \| p_\phi(s_t|s_{t-1}, a_{t-1})] - \mathbb{E}_{q_\phi(s_t)}[\log p_\phi(x_t|s_t)], \tag{3}$$

where sequences of observations x_t and actions a_t are sampled from a replay buffer.



(a) Skill discovery. (b) Skill learning. (c) Skill adaptation.

Figure 1: **Choreographer.** Overview of the skill components learned by the agent.

Skill discovery. To discover skills whose discriminability is independent from the data collection process, we turn the skill discovery problem into a representation learning one. We learn a skill discrete representation z on top of the model states using a VQ-VAE [48]. This representation aims to encode model states into the N codes of a codebook, learning $p(z|s)$, as well as the inverse relation, decoding codes to model states, via $p(s|z)$. Crucially, as opposed to most previous work [48; 37], we choose to learn a single-code representation, where each model state gets assigned only one vector in the codebook. Our skill codes can thus be seen as centroids partitioning the model state space into N clusters [38], so that intuitively each skill occupies one area of the state space.

In the skill auto-encoder, model states are first processed into embeddings using a skill encoder $E(s)$. Then, each embedding is assigned to the closest code in the codebook $z_q^{(i)}, i \in 1...N$, through:

$$\text{quantize}(E(s)) = z_q^{(k)} \qquad k = \arg\min_j \|E(s) - z_q^{(j)}\|_2 \tag{4}$$

The indices are mapped back to their corresponding vectors in the codebook, from which the decoder $D(z_q^k)$ reconstructs the corresponding model states. The training objective is:

$$\mathcal{L}_z = \|s - D(z_q^{(k)})\|_2^2 + \beta \|\text{sg}(z_q^{(k)}) - E(s)\|_2^2, \tag{5}$$

where $\text{sg}(\cdot)$ indicates to stop gradients. The first loss term is the reconstruction error of the decoder, while the second term encourages the embedded vectors to stay close to the codes in the codebook. In order to stabilize training when learning large codebooks of skills, we introduce a code *resampling technique*, detailed in Appendix C. This ensure that all the codes in the codebook are actively used by the autoencoder, preventing the infamous issue of index collapse [21; 38].

Skill learning. Choreographer learns a world model, which allows the agent to imagine trajectories of future model states $\tau_s = \{s_t, s_{t+1}, ...\}$. A major novelty is that we learn a skill representation that is linked to model states rather than to environment observations. As a consequence, for skill learning, we define the objective of Choreographer to maximize the mutual information between trajectories of model states and skills $I(\tau_s, z)$. In visual environments, this is particularly useful as the model states s_t can have a much more compact representation than observations x_t.

We rewrite the mutual information objective as:

$$I(\tau_s, z) = H(\tau_s) - H(\tau_s|z) = H(\tau_s) + \mathbb{E}_{s \sim d_{\pi_z}(s), z \sim \text{Unif}(z)}[\log p(s|z)], \tag{6}$$

where the second term uses $d_{\pi_z}(s)$ to express the stationary distribution of model states visited by the skill policy conditioned on the skill code z, and codes are sampled from a uniform distribution. In order to maximize this objective with RL, we estimate the two terms to be part of an overarching reward function. The entropy term is estimated using a K-NN particle-based estimator [44], obtaining $r_{\text{ent}} \propto \sum_{i=1}^{K} \log \|s - s_i^{\text{K-NN}}\|_2$. For the second term, we can rollout skill-conditioned policies in imagination, obtaining sampled trajectories of model states, and use the distance between the states visited and the decoded state corresponding to the conditioning skill code z to estimate the likelihood [11], obtaining $r_{\text{code}} \propto -\|D(z) - s\|_2^2$.

Choreographer learns N code-conditioned skill policies using an actor-critic algorithm:

$$r_{\text{skill}} = r_{\text{ent}} + r_{\text{code}}, \qquad \text{Skill actor:} \quad \pi_{\text{skill}}(a_t|s_t, z), \qquad \text{Skill critic:} \quad v_{\text{skill}}(s_t, z), \tag{7}$$

where the policies are encouraged to reach states corresponding to their skill-conditioning code z, by the r_{code} term, but also to be dynamic and visit different states in one trajectory, by the r_{ent} term. Given that the reward terms can be computed using the agent's model, the skill actor-critics are trained in the agent's imagination, backpropagating through the model dynamics [17]. By doing so, we are able to disentangle exploration and skill learning, and thus skill policies can be learned offline. Skill visualizations are available in Appendix E and F and on the project website.

Skill adaptation. Choreographer learns skill-conditioned policies in a completely unsupervised fashion, leveraging representation and reinforcement learning. After pre-training offline on unsupervised data, the agent can leverage the learned world model and skills for adapting to downstream tasks in an efficient way. Exhaustively deploying the skill policies in the environment, to understand which ones correlate best with the downstream task, is expensive and often unnecessary. Choreographer can again exploit its world model to deploy and evaluate the skill policies in imagination.

In order to coordinate the multiple skills available, Choreographer learns a *meta-controller*, a policy deciding which skill codes use to condition the skill policy:

$$r_{\text{meta}} = r_{\text{task}}, \qquad \text{Meta-ctrl actor:} \quad \pi_{\text{meta}}(z|s_t), \qquad \text{Meta-ctrl critic:} \quad v_{\text{meta}}(s_t). \tag{8}$$

The meta-controller is updated with rollouts in imagination using the task rewards, which are predicted by the model at adaptation time. The outputs of the meta-controller's actor network are discrete (skill) vectors and we use policy gradients to update the actor [49]. In order to fine-tune the pre-trained skill policies, we backpropagate the gradients to the skill actors too [23].

3 Experiments

Choreographer learns model and skills from offline unsupervised data to efficiently adapt for downstream tasks. In our evaluation, the agent is first pre-trained on unlabelled data (no reward information) collected by another exploration agent, during an unsupervised interaction phase with the environment. Then, we compare the performance at adaptation/fine-tuning time, where the agent receives task rewards when interacting with the environment. We note that all other baselines are not designed to work offline: their agents are learned online during the unsupervised pre-training stage. Implementation, experimental details and hyperparameters are given in Appendix D.

State-based URLB. Choreographer decouples exploration and skill learning, and can thus learn from offline pre-collected data. To show this, we use the pre-collected datasets of exploratory data released in the ExORL paper [52]. We take the first 2M steps from the datasets corresponding to the state-based URLB [24] domains and use these to pre-train Choreographer offline, learning the world model and the skill-related components. In Figure 2, we compare task performance after 100k adaptation steps of our method to previous results on the benchmark [24]. In particular, we show the performance when pre-training Choreographer using the data collected by the RND approach [5]. In Appendix E, we show results using different data.

Following Agarwal et al. [2], we show that Choreographer performs best both in terms of bootstrapped interquartile mean (IQM) and optimality gap, particularly outperforming the other approaches on the Walker and Quadruped and performing comparably to CIC [25] in the Jaco. We highlight that Choreographer performs far better than the other skill-based approaches, SMM [27],

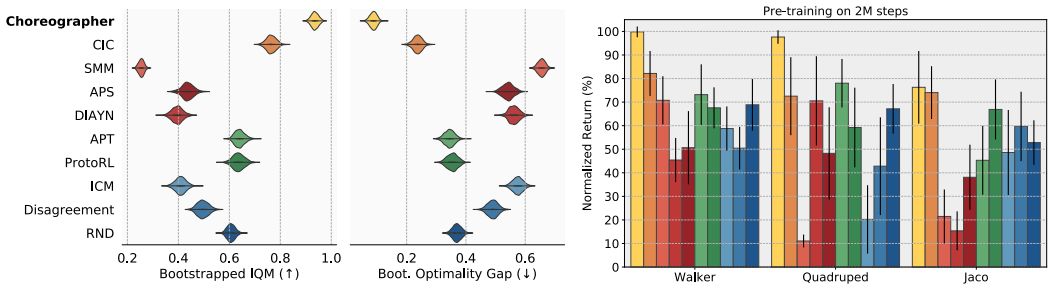


Figure 2: **Offline state-based URLB.** On the left, Choreographer (ours), pre-trained with offline exploratory data, performs best against baselines, both in terms of IQM and Optimality Gap. On the right, mean and standard deviations across the different domains of URLB are detailed (10 seeds).

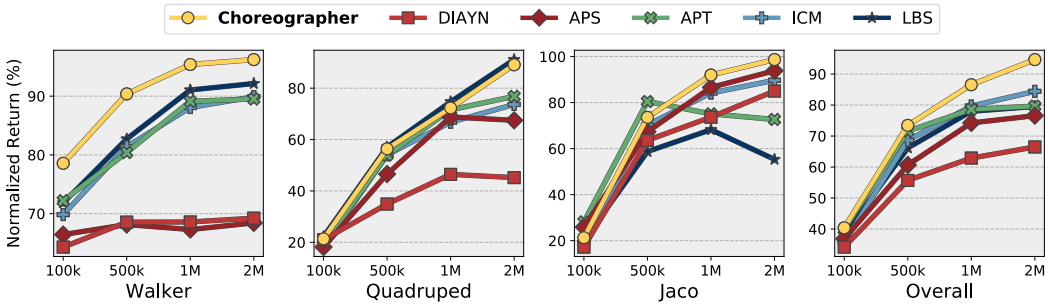


Figure 3: **Pixel-based URLB.** Performance of Choreographer (ours) as a function of pre-training steps. Scores are asymptotically normalized, and averaged across tasks for each method. (10 seeds)

APS [28], and DIAYN [8], and also strongly outperforms RND, which is the strategy used to collect its pre-training data, showing it's a stronger method to exploit the same kind of unsupervised data.

Pixel-based URLB. Choreographer can be trained in parallel to the exploration process, learning offline from the data collected by an exploration policy [31]. We use the pixel-based setup of URLB to see how the performance evolves over time as the size of the pre-training data buffer grows. We take snapshots of the agent at 100k, 500k, 1M, and 2M steps and fine-tune them for 100k steps. In Figure 3, we show the performance of Choreographer, trained on the data collected with an LBS exploration policy [30] compared to the strong baselines from Rajeswar et al. [36]. Additional baselines are presented in Appendix E.

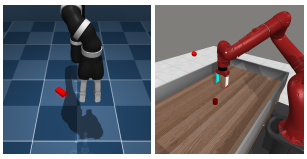
Overall, our performance is steadily on top, with similar performance to APT [29], LBS [30], and ICM [34] until 500k steps, and becoming the one best-performing algorithm from 1M steps on. As also noticed in other works [24], other skill-based approaches APS and DIAYN struggle on the benchmark. Furthermore, both LBS and APT's performance in Jaco starts decaying after 500k steps. As also discussed in Rajeswar et al. [36], this is due to the methods learning behaviors that are good for exploration but that may commit to non-rewarding areas of the environment during adaptation, failing to solve Jaco sparse rewards tasks. Our approach overcomes this limitation by learning a set of diverse policies, consistently providing a more general initialization that is easier to adapt [9].

Sparse rewards. Choreographer can exploit pre-trained skills policies to reach different model states, efficiently skimming the environment to find sparse rewards. We test this hypothesis using sparse goal-reaching tasks in two pixel-based environments: Jaco from the DMC Suite, and Meta-World [53]. In order to solve these tasks, the agent needs to leverage the pre-trained behavior to reach within a certain radius of a target goal located in the environment with (nearly) no feedback.

Jaco. We zoom in on the performance on the four reaching tasks from URLB. In Table 1, we show the fraction of runs reaching near the target, without any update of the pre-trained policies, over 100 evaluation episodes. Only Choreographer and random actions found the goals in all runs, the other methods have to rely on entropic exploration to find the goals[1].

Meta-World. We sampled three sets of 50 goals, in the Sawyer environment from MetaWorld. For each goal in the sets, we evaluate the policies pre-trained by different methods, testing them for 100 evaluation episodes, without any fine-tuning. The performance in Table 1 shows that Choreographer finds the most goals in the environment. All the other approaches struggled in the task, with random actions being the best alternative for higher chances of finding sparse rewards for these goals.

[1]For the results in Figure 3 all the baselines (except Choreographer) had an action entropy maximization term in the adaptation objective [14], helping them to find the goals.



Domain	Task	Choreo	Random	APS	DIAYN	APT	ICM	LBS	P2E	RND
Jaco	Bottom left	**1.0**	**1.0**	0.67	0.67	0.67	0.33	0.0	0.67	0.67
	Bottom right	**1.0**	**1.0**	0.67	0.67	0.33	0.67	0.0	1.0	0.33
	Top left	**1.0**	**1.0**	0.67	0.67	0.0	0.67	0.33	0.33	0.0
	Top right	**1.0**	**1.0**	0.0	0.67	0.33	0.0	0.33	0.67	0.33
MetaWorld	Goal set 1	**0.78**	0.47	0.31	0.08	0.24	0.33	0.33	0.51	0.35
	Goal set 2	**0.80**	0.49	0.18	0.02	0.16	0.37	0.18	0.41	0.43
	Goal set 3	**0.84**	0.51	0.12	0.04	0.18	0.18	0.61	0.37	0.22

Table 1: **Sparse rewards.** Finding goals in the environment with sparse rewards using the pre-trained behaviors. The table shows the fraction of goals found by each method per task. (3 seeds)

4 DISCUSSION

In this work, we presented Choreographer a model-based agent for learning skills offline from unsupervised data. The skills can be efficiently adapted for downstream tasks, obtaining state-of-the-art performance, as shown in URLB, or used to explore the environment quickly and find sparse rewards more frequently than with other approaches, as shown in robotic goal-reaching tasks.

Pre-training agents from offline unsupervised data is an important challenge in RL, in order to be able to exploit large amounts of unlabelled data for adapting with greater efficiency or generalizing to new tasks. In unsupervised reinforcement learning, there has been significant progress on how to improve the unsupervised data collection stage, leading to stronger and stronger exploration approaches. We believe the intuition behind Choreographer, of combining representation and reinforcement learning, could become a general strategy for bootstrapping research on how to exploit unsupervised data from an offline perspective.

REFERENCES

[1] Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. Variational option discovery algorithms. *ArXiv*, abs/1807.10299, 2018.

[2] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 34, 2021.

[3] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[4] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013. URL `https://arxiv.org/abs/1308.3432`.

[5] Yuri Burda, Harrison Edwards, Amos J. Storkey, and Oleg Klimov. Exploration by random network distillation. *ICLR*, 2019.

[6] Víctor Campos, Alexander Trott, Caiming Xiong, Richard Socher, Xavier Giro i Nieto, and Jordi Torres. Explore, discover and learn: Unsupervised discovery of state-covering skills. In *ICML*, 2020.

[7] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *SSST@EMNLP*, 2014.

[8] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *ICLR*, 2019.

[9] Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. The information geometry of unsupervised reinforcement learning, 2021. URL `https://arxiv.org/abs/2110.02719`.

[10] Jonas Gehring, Gabriel Synnaeve, Andreas Krause, and Nicolas Usunier. Hierarchical skills for efficient exploration, 2021. URL `https://arxiv.org/abs/2110.10809`.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[12] Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control, 2016. URL `https://arxiv.org/abs/1611.07507`.

[13] David Ha and Jürgen Schmidhuber. World models. 2018. doi: 10.5281/ZENODO.1207631. URL `https://zenodo.org/record/1207631`.

[14] Tuomas Haarnoja, Aurick Zhou, P. Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

[15] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *ICLR*, 2019.

[16] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *ICML*, pp. 2555–2565, 2019.

[17] Danijar Hafner, Timothy P Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *ICLR*, 2021.

[18] Danijar Hafner, Kuang-Huei Lee, Ian Fischer, and Pieter Abbeel. Deep hierarchical planning from pixels, 2022. URL https://arxiv.org/abs/2206.04114.

[19] Steven Stenberg Hansen, Guillaume Desjardins, Kate Baumli, David Warde-Farley, Nicolas Heess, Simon Osindero, and Volodymyr Mnih. Entropic desired dynamics for intrinsic control. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021.

[20] Elad Hazan, Sham M. Kakade, Karan Singh, and Abby Van Soest. Provably efficient maximum entropy exploration. *ArXiv*, abs/1812.02690, 2019.

[21] Łukasz Kaiser, Aurko Roy, Ashish Vaswani, Niki Parmar, Samy Bengio, Jakob Uszkoreit, and Noam Shazeer. Fast decoding in sequence models using discrete latent variables, 2018. URL https://arxiv.org/abs/1803.03382.

[22] Jaekyeom Kim, Seohong Park, and Gunhee Kim. Unsupervised skill discovery with bottleneck option learning. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 5572–5582. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/kim21j.html.

[23] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013. URL https://arxiv.org/abs/1312.6114.

[24] Michael Laskin, Denis Yarats, Hao Liu, Kimin Lee, Albert Zhan, Kevin Lu, Catherine Cang, Lerrel Pinto, and Pieter Abbeel. Urlb: Unsupervised reinforcement learning benchmark, 2021. URL https://arxiv.org/abs/2110.15191.

[25] Michael Laskin, Hao Liu, Xue Bin Peng, Denis Yarats, Aravind Rajeswaran, and Pieter Abbeel. Cic: Contrastive intrinsic control for unsupervised skill discovery, 2022. URL https://arxiv.org/abs/2202.00161.

[26] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pp. 319–345. Springer, 1999.

[27] Lisa Lee, Benjamin Eysenbach, Emilio Parisotto, Eric Xing, Sergey Levine, and Ruslan Salakhutdinov. Efficient exploration via state marginal matching, 2019. URL https://arxiv.org/abs/1906.05274.

[28] Hao Liu and P. Abbeel. Aps: Active pretraining with successor features. In *ICML*, 2021.

[29] Hao Liu and Pieter Abbeel. Behavior from the void: Unsupervised active pre-training. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 18459–18473. Curran Associates, Inc., 2021.

[30] Pietro Mazzaglia, Ozan Çatal, Tim Verbelen, and B. Dhoedt. Curiosity-driven exploration via latent bayesian surprise. *ArXiv*, abs/2104.07495, 2021.

[31] Russell Mendonca, Oleh Rybkin, Kostas Daniilidis, Danijar Hafner, and Deepak Pathak. Discovering and achieving goals via world models. In *Advances in Neural Information Processing Systems*, 2021.

[32] Juan José Nieto, Roger Creus, and Xavier Giro-i Nieto. Unsupervised skill-discovery and skill-learning in minecraft, 2021. URL https://arxiv.org/abs/2107.08398.

[33] Seohong Park, Jongwook Choi, Jaekyeom Kim, Honglak Lee, and Gunhee Kim. Lipschitz-constrained unsupervised skill discovery, 2022. URL https://arxiv.org/abs/2202.00914.

[34] Deepak Pathak, Pulkit Agrawal, Alexei Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. *ICML*, 2017.

[35] Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. In *ICML*, 2019.

[36] Sai Rajeswar, Pietro Mazzaglia, Tim Verbelen, Alexandre Piché, Bart Dhoedt, Aaron Courville, and Alexandre Lacoste. Unsupervised model-based pre-training for data-efficient control from pixels, 2022. URL https://arxiv.org/abs/2209.12016.

[37] Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2, 2019. URL https://arxiv.org/abs/1906.00446.

[38] Aurko Roy, Ashish Vaswani, Arvind Neelakantan, and Niki Parmar. Theory and experiments on vector quantized autoencoders, 2018. URL https://arxiv.org/abs/1805.11063.

[39] Christoph Salge, Cornelius Glackin, and Daniel Polani. Changing the environment based on empowerment as intrinsic motivation. *Entropy*, 16(5):2789–2819, may 2014. doi: 10.3390/e16052789. URL https://doi.org/10.3390%2Fe16052789.

[40] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1312–1320, Lille, France, 07–09 Jul 2015. PMLR.

[41] J. Schmidhuber. Curious model-building control systems. In *[Proceedings] 1991 IEEE International Joint Conference on Neural Networks*, pp. 1458–1463 vol.2, 1991. doi: 10.1109/IJCNN.1991.170605.

[42] Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *ICML*, 2020.

[43] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. In *International Conference on Learning Representations*, 2020.

[44] Harshinder Singh, Neeraj Misra, Vladimir Hnizdo, Adam Fedorowicz, and Eugene Demchuk. Nearest neighbor estimates of entropy. *American journal of mathematical and management sciences*, 23(3-4):301–321, 2003.

[45] Aravind Srinivas and Pieter Abbeel. Unsupervised learning for reinforcement learning. 2021. URL https://icml.cc/media/icml-2021/Slides/10843_QHaHBNU.pdf.

[46] Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163, jul 1991. ISSN 0163-5719. doi: 10.1145/122344.122377. URL https://doi.org/10.1145/122344.122377.

[47] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy P. Lillicrap, and Martin A. Riedmiller. Deepmind control suite. *CoRR*, abs/1801.00690, 2018.

[48] Aaron van den Oord, Oriol Vinyals, and koray kavukcuoglu. Neural discrete representation learning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/7a98af17e63a0ac09ce2e96d03992fbc-Paper.pdf.

[49] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

[50] Kevin Xie, Homanga Bharadhwaj, Danijar Hafner, Animesh Garg, and Florian Shkurti. Latent skill planning for exploration and transfer, 2020. URL https://arxiv.org/abs/2011.13897.

[51] Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Reinforcement learning with prototypical representations. 2021.

[52] Denis Yarats, David Brandfonbrener, Hao Liu, Michael Laskin, P. Abbeel, Alessandro Lazaric, and Lerrel Pinto. Don't change the algorithm, change the data: Exploratory data for offline reinforcement learning. *ArXiv*, abs/2201.13425, 2022.

[53] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019. URL https://arxiv.org/abs/1910.10897.

APPENDIX

A BASELINES

For state-based experiments, we adopt the baselines from the CIC paper [25], which can be grouped in three categories: knowledge-based (blue in plots), which are ICM [34], RND [5], Disagreement [35]; data-based (green in plots), which are APT [29] and ProtoRL [51]; and competence-based (red/orange in plots) APS [28], SMM [27] and DIAYN [8].

For pixel-based experiments, we compare to the strong baselines from Rajeswar et al. [36], combining some of the above approaches with the Dreamer algorithm.

B RELATED WORK

Unsupervised RL. Among unsupervised RL methods, we can distinguish [45]: knowledge-based methods, which aim to visit states of the environment that maximize the agent's model error/uncertainty [41; 34; 5], data-based methods, which aim to maximize the entropy of the data collected [20; 51; 29], competence-based methods, which learn a set of options or skills that allow the agent to control the environment [12; 8; 19]. Knowledge and data-based methods focus on expanding the available information, while competence-based methods, such as Choreographer, aim to provide empowerment [39].

Exploration and skills. Exploration is a major challenge in skill learning approaches. Previous work has attempted to solve this issue by using state maximum entropy [25; 28] or introducing variations of the skill discovery objective [22; 33]. The idea of decoupling exploration and skill discovery has been less explored and mostly applied in simpler environments [6; 32]. To the best of our knowledge, Choreographer represents the first approach that enables scaling the idea to complex environments and tasks successfully.

Behavior learning in imagination. The Dreamer algorithm [15; 17] introduced behavior learning in imagination, establishing a modern implementation of the Dyna framework [46]. The idea has been applied for exploration [42; 36; 30], goal-directed behavior [18; 31], and planning amortization [50]. In particular, Rajeswar et al. [36] showed the effectiveness of fine-tuning behavior in imagination on the URL benchmark from pixels, combining unsupervised RL for exploration with Dreamer. Choreographer not only yields better performance but also decouples exploration and behavior pre-training, as the skill policies can be learned offline.

C CODE RESAMPLING

In RL, we mostly face datasets that grow arbitrarily over time, as the agent collects new data. These datasets may become large (in our case, up to $2e^6$ environment transitions) and thus require a long time to run clustering algorithms, such as k-means. Furthermore, as the data increases, the algorithm may need to be run again, several times, to comply with the new data.

By using a VQ-VAE, we are able to perform clustering of the model state space, as well as to decode the cluster centers into their corresponding model states (which is necessary for Choreographer skill learning objective). However, when training single-code representations with VQ-VAE, the codebook may suffer from *index collapse* [21], where only a few of the code vectors get trained due to a preferential attachment process. One way to mitigate this issue is to use sliced latent vectors, which assign multiple smaller codes to the same embedding [21; 37], but this would imply losing the one-to-one association between codes and skills.

Learning a diversified set of skill codes is important to learn useful skill policies for adaptation. Because of index collapse [21], VQ-VAEs tend to ignore some of the skill codes, choosing not to use them during training. As Choreographer samples skills uniformly for skill learning (Eq. 6), it would be inefficient to train policies that are conditioned on inactive codes.

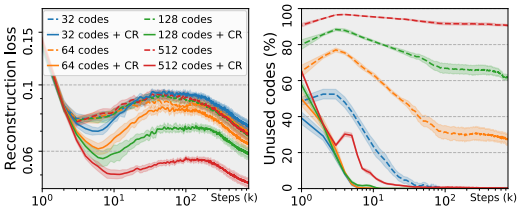
In order to mitigate index collapse we propose a *code resampling* technique which consists of two steps:

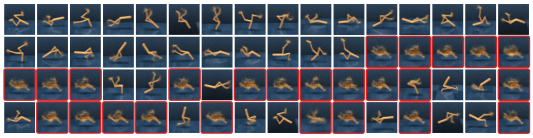
1. keeping track of the inactive codes during training, which is achieved by simply looking at the codes to which no embedding is assigned for M consecutive training batches;

2. re-initializing the inactive codes, with values of embeddings $E(s)$ from the latest training batch, selected with a probability $\frac{d_q^2(E(s))}{\sum_s d_q^2(E(s))}$, where $d_q(\cdot)$ is the euclidean distance from the closest code in the codebook, i.e. $d_q(E(s)) = \min_{i \in 1 \dots N} \|E(s) - z_q^{(i)}\|_2^2$.

The second step ensures that every M training batches all the codes become active again.

As we show in Figure 5 in the pixel-based URLB environments, code resampling circumvents this issue, making nearly all the codes active during training, reducing the reconstruction loss, and avoiding skill code collapse (see examples of collapsed codes in the Figure).





(a) Reconstruction of 64 codes, no code resampling. Inactive, collapsed codes highlighted in red.

Figure 5: **Code resampling (CR).** When using more than 32 codes, CR reduces the number of unused codes ($\sim$0% after 10k steps) and leads to a lower reconstruction loss. (3 seeds $\times$ 3 domains)

The second step of our code resampling technique is inspired by the k-means++ initialization [3]. This technique is generally applied with fixed datasets to reduce the k-means convergence time, by providing a better initialization. Instead, we use the code re-initialization procedure to 'resample' codes every M training batches, making sure that any collapsed index/code is reactivated.

Our technique scales well for the RL setting, as it uses only the data in the previous batch to resample the code. Of course, this may lead to a suboptimal code choice, which can then be adjusted at following code resampling iterations, if necessary. We show in Figure 5 that in practice all the codes tend to stay active over training.

Algorithm 1: Code Resampling

```
if train_step mod M == 0:
    for code in inactive_codes:
        distance_matrix = l2norm_distance(current_batch, current_codes)
        probabilities = min(distance_matrix, axis=code_axis) \
                        / sum(min(distance_matrix, axis=code_axis))
        code = current_batch[sample_index(probs=probabilities)]
```

D Implementation

D.1 Agent

World Model. We follow the architecture of the DreamerV2 agent's world model [17]. Model states have both a deterministic and a stochastic component: the deterministic component is the 200-dimensional output of a GRU [7], with a 200-dimensional hidden layer; the stochastic component consists of 32 categorical distributions with 32 classes each. For states-input experiments, the encoder and decoder are 4-layers MLP with a dimensionality of 400. For pixels-based experiments, the encoder and decoder follow the architecture of DreamerV2, taking 64×64 RGB images as inputs.

Skill auto-encoder. The encoder and decoder networks are 4-layer MLP with a dimensionality of 400. The input and target of the auto-encoder are the deterministic part of the world model states. The codebook consists of 64 codes of dimensionality 16. The gradients of the skill auto-encoder objective are back-propagated through the decoder and the encoder using straight-through gradients [4]. The codes of the codebooks are updated using an exponential moving average of their assigned embeddings [37]. Code resampling happens every $M = 200$ training batches.

Both the world model and the skill-autoencoder networks are updated sampling batches of 50 sequences of 50 timesteps, using Adam with learning rate $3e^{-4}$ for the updates, and clipping gradients norm to 100.

Skill actor-critic. Skill actor-critic takes as input both model states and skill vectors [40]. We use the Dreamer algorithm to update them in latent imagination [15], imagining sequences of 15 steps with the model, starting from the world model training batch states. The actor and critic networks are instantiated as 4-layer MLP with a dimensionality of 400 and updated using Adam with a learning rate of $8e^{-5}$ and gradients clipped to 100. The actor's output is a truncated normal distribution and its gradients are backpropagated using the reparameterization trick [23].

Rewards are computed in latent imagination using:

$$r_{\text{skill}}(s, z) = \underbrace{\frac{1}{K}\sum_{i=1}^{K} \|s - s_i^{\text{K-NN}}\|_2}_{r_{\text{entropy}}} \underbrace{- \|D(z) - s\|_2}_{r_{\text{code}}} .$$

For the entropy term, where we approximate a particle-based entropy estimator [44], we follow Liu & Abbeel [29] in averaging the distance from the K-nearest neighbors ($K = 30$) and remove the $\log$. For the code term, we take the L2-norm directly instead of the squared L2-norm. With these minor modifications, the two terms are on the same scale and we found it unnecessary to add any scaling coefficient to balance the two reward terms.

Meta-controller actor-critic. The meta-controller actor-critic is instantiated and trained in the same way as the skill actor-critic, using the rewards provided by a reward predictor, which is trained along with the model to predict downstream task rewards, at adaptation time. While the meta-controller is updated using policy gradients [49], we can backpropagate the meta-controller loss to the skill actor-critic to fine-tune them, with the reparameterization trick.

One issue we found is that when the rewards are too small (or zero) to predict, the predictions of the reward predictor and the critic are noisy around zero, and tend to collapse the skill behaviors during fine-tuning. In order to avoid that, we introduce a reward smoothing mechanism, which makes sure the critic and reward predictions are exactly zero, until a reward $\geq 1e^{-4}$ is found. By doing so, we ensure that, until an actual reward is encountered, the skill policies remain the ones learned during pre-training and can be used to explore the environment, looking for rewards. To provide consistent exploration, skills are sampled and followed for an entire trajectory, until rewards are encountered.

D.2 Experiments

URLB. The URL benchmark [24] is designed to compare different URL strategies. The experimental setup consists of two phases: a longer data collection/pre-training phase, where the agent has up to 2M environment steps to interact with the environment without rewards, and a shorter fine-tuning phase, where the agent has 100k steps to interact with a task-specific version of the environment, where it should both discover rewards and solve a downstream task. In the URLB, there are three domains from the DM Control Suite [47], Walker, Quadruped, and Jaco, with four downstream tasks to solve per each domain.

For state-based inputs, we used an action repeat of 1. For pixel-based inputs, we used an action repeat of 2. For the state-based experiments, we note that the Jaco environment in ExORL slightly differs from the one in URLB, as the position of the target is not provided to the agent in ExORL. Other than that, the Jaco tasks remain the same and we see that, nonetheless, Choreographer is able to find the targets without information about their position, performing comparably to the top-performing baselines.

For the sparse-rewards Jaco, we consider the target reached when the agent finds a reward $\geq 1e^{-4}$.

MetaWorld. In MetaWorld, the task can be adapted to be sparse, by setting rewards equal to the success metrics in the environment. For the `reach` tasks, success is when the agent reaches the target. In order to build the three goal sets, we adapted the code to sample 50 goals in the intervals: goal_low $= (-0.45, 0.3, 0.1)$, goal_high $= (0.45, 0.9, 0.5)$, with seeds $(1, 2, 3)$.

For both sparse experiments, the agents are pre-trained on 2M frames collected using their exploration policy (an LBS [30] exploration policy is used for Choreographer). Then, the pre-trained behavior is deployed for evaluation. For the skill-based approaches, which are DIAYN, APS and Choreographer, skills are randomly re-sampled every 50 steps.

D.3 Other Things We Tried

Skill Discovery. We tried alternative representations other than the skill auto-encoder, for skill discovery. In particular, we tried contrastive learning and soft clustering using the Sinkhorn-Knopp algorithm [51]. While both of them worked, we found the skill vectors found by the autoencoder to be more diverse and explainable (thanks to the reconstruction). We also tried using a standard discrete VAE, using straight-through gradients [4], compared to a VQ-VAE; however, the training was way more unstable.

Skill learning. We attempted to learn different numbers of skills, ranging from 16 up to 512. We found that the best performance was obtained when using a number of skills comprised between 64 and 128, and we stick to 64 for our experiments. Too few skills were sometimes unable to find useful behaviors in the Quadruped or to find sparse targets in the Jaco tasks. Too many skills, under the same computational budget, generally led to skills that didn't fully converge or were harder to fine-tune. Though, likely, with a higher computational budget, more skills might perform better.

Skill adaptation. The meta-controller generally samples skills at a time-frequency of 1. Other works have focused on learning skills that last several timesteps [43], using them for hierarchical control [10]. Choreographer, instead, tends to evaluate all skills in imagination and narrow down its choice to a restricted set of skills that are adapted for the downstream tasks. We found that sampling skills with lower frequency (we tried every 3,5,15,50 steps) does not lead to better adaptation performance, but often doesn't harm performance either. We believe that leveraging the skills for hierarchical problems is an interesting direction for future work.

E Additional Results

State-based URLB. In Table 2, we present detailed results and baselines on the state-based URLB. Choreographer performs best on all the Walker and Quadruped tasks and comparably to CIC in the Jaco tasks. The normalization scores for the results in the paper are the Expert results.

Pre-trainining for 2×10^6 environment steps													
Domain	Task	Expert	DDPG	ICM	Disagreement	RND	APT	ProtoRL	SMM	DIAYN	APS	CIC	Choreographer
Walker	Flip	799	538±27	417±16	346±13	474±39	544±14	456±12	450±24	319±17	465±20	631 ± 34	**960 ± 3**
	Run	796	325±25	247±21	208±15	406±30	392±26	306±13	426±26	158±8	134±16	486 ± 25	**635 ± 10**
	Stand	984	899±23	859±23	746±34	911±5	942±6	917±27	924±12	695±46	721±44	959 ± 2	**980 ± 1**
	Walk	971	748±47	627±42	549±37	704±30	773±70	792±41	770±44	498±27	527±79	885 ± 28	**967 ± 1**
Quadruped	Jump	888	236±48	178±35	389±62	637±12	648±18	617±44	96±7	660±43	463±51	595 ± 42	**830 ± 9**
	Run	888	157±31	110±18	337±30	459±6	492±14	373±33	96±6	433±29	281±17	505 ± 47	**779 ± 11**
	Stand	920	392±73	312±68	512±89	766±43	872±23	716±56	123±11	851±43	542±53	761 ± 54	**946 ± 5**
	Walk	866	229±57	126±27	293±37	536±39	770±47	412±54	80±6	576±81	436±79	723 ± 43	**921 ± 5**
Jaco	Reach bottom left	193	72±22	111±11	124±7	110±5	103±8	129±8	45±7	39±6	76±8	138 ± 9	137 ± 10
	Reach bottom right	203	117±18	97±9	115±10	117±7	100±6	132±8	46±11	38±5	88±11	145 ± 7	147 ± 10
	Reach top left	191	116±22	82±14	106±12	99±6	73±12	123±9	36±3	19±4	68±6	153 ± 7	164 ± 7
	Reach top right	223	94±18	103±11	139±7	100±6	90±10	159±7	47±6	28±6	76±10	163 ± 4	169 ± 10

Table 2: Mean and standard error performance of Choreographer on state-based URLB after pre-training for 2M steps and fine-tuning 100k steps. Baseline scores from [25].

In Figure 2, we showed the results of Choreographer pre-trained with 2M steps of exploratory data from the RND approach. In Figure 6, we compare to the results when pre-training with the data collected by other knowledge-based approaches, i.e. ICM, Disagreement, or random actions data.

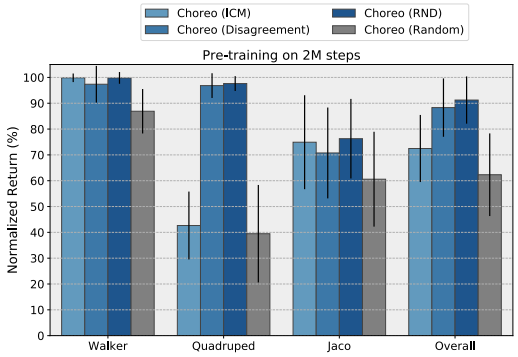


Figure 6: **Different data.** Training Choreographer's world model and skills using data from different exploration strategies.

We see that the quality of the model and the skill learned, and so the fine-tuning performance, depends on the data used. Choreographer is able to learn skills that widely explore the model state space. However, if the state space explored is little, then the skills will be also limited to a restricted area. This is what generally happens for the case of the random actions in the Figure, but also in the Quadruped for ICM (where we visually inspected the data from ExORL, verifying that ICM explores poorly in the Quadruped environment).

Pixel-based URLB. In Figure 7, we show additional results, including more baselines from [36], on the pixel-based URLB. Overall, Choreographer clearly outperforms all methods, while LBS is still competitive on the Quadruped.

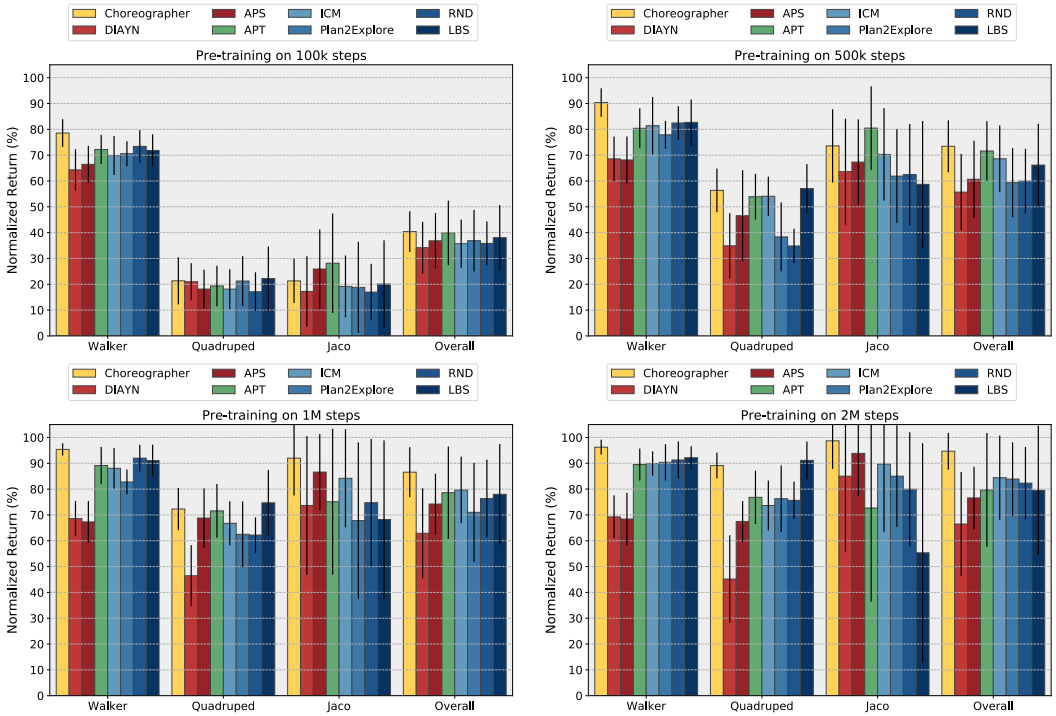


Figure 7: **Pixel-based URLB.** Additional baselines for the pixel-based URLB experiments.

Skill visualization. In Figure 8, we visualize some of the skills learned by Choreographer, in the pixel-based Walker and Quadruped environments. The codes conditioning the skill policies are shown by first reconstructing the corresponding model state, and next the image observation using the reconstruction model. We note that Choreographer's learned behaviors accurately reflect the skill learning objective (Eq. 7): they tend to reach the states corresponding to the conditioning skill codes, but they also dynamically visit different states. For the Walker, this leads to walking-like behaviors for standing-up skill codes (1st row), or behaviors that move around, by hopping or flipping on the torso/legs (2nd row). For the Quadruped, skills tend to learn tipping-over behaviors, where the agent reaches the skill code state alternating rotating and overturning moves (3rd/4th rows). Further visualizations are in Appendix F and on the project website.

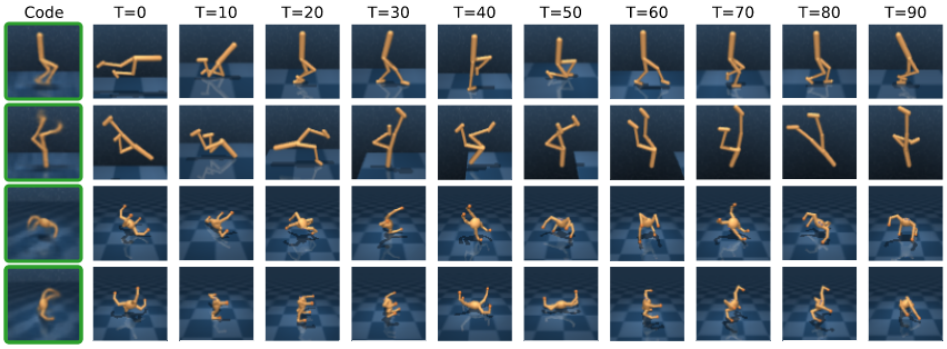


Figure 8: **Skills visualization.** Highlighted in green, reconstructions of the skill codes conditioning the skill behavior on their right, sampling frames over time. Animated results on the project website.

Zero-shot skill control. Pre-training skill policies is a general approach to provide a better initialization of the agent's behavior for fine-tuning to downstream tasks [9]. As shown in [9], it is not possible to learn a set of skill policies able to solve any task, without adaptation. We empirically reinforce this hypothesis with an ablation on the pixel-based URLB, where we learn the meta-controller, which interpolates the skill policies to solve the task, but we do not fine-tune the skill policies.

In Figure 3, we showed the performance of Choreographer in the pixel-based setting over time. This performance was achieved by learning the meta-controller and adapting the skill policies during fine-tuning. In Figure 9, we show the impact of fine-tuning the skill policies learned during pre-training, after 2M steps. This can be seen as restricting the agent's action space to a set of N state-dependent actions (=64 skills in this case), which the agent can use to solve the downstream task.

The performance on the Walker and Quadruped tasks is weaker, but the agent still performs comparably to some weaker baselines in Figure 3, such as APS and DIAYN. Instead, the performance on Jaco is much worse than the one with skill adaptation, that's because the task requires precision, to reach the target block, and adaptability, in case the block moves.

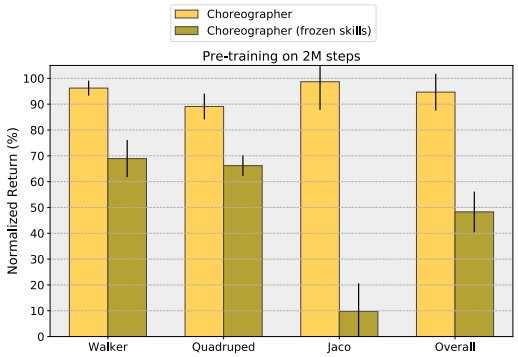


Figure 9: **Zero-shot skill control.** We attempt to solve the task using the pre-trained skill policies, without any fine-tuning.

F Additional Visualizations

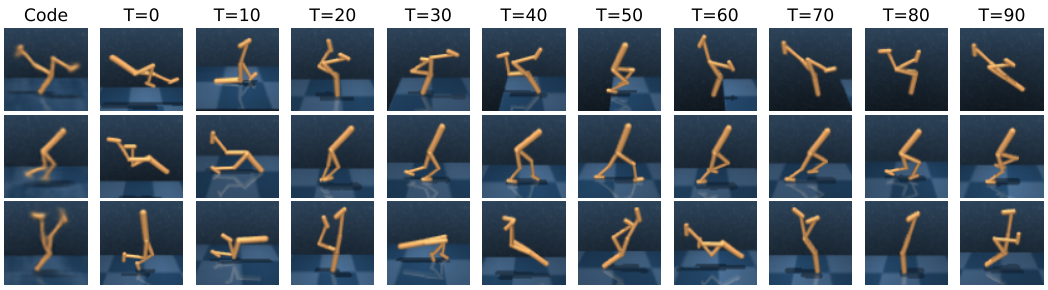


Figure 10: **Walker.** Sets of three skill codes and respective behaviors visualized for Walker.

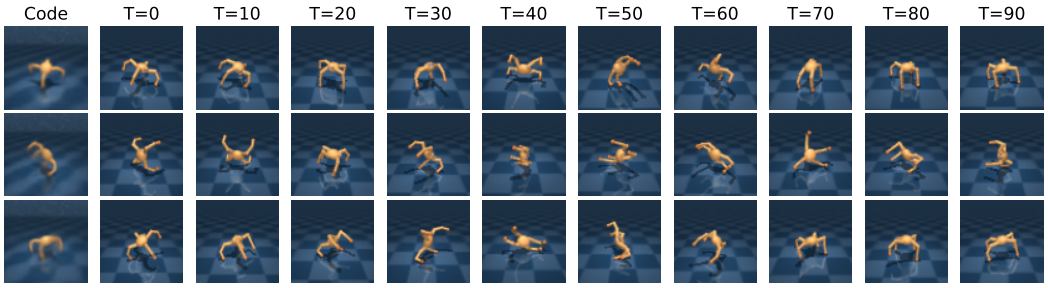


Figure 11: **Quadruped.** Sets of three skill codes and respective behaviors visualized for Quadruped.

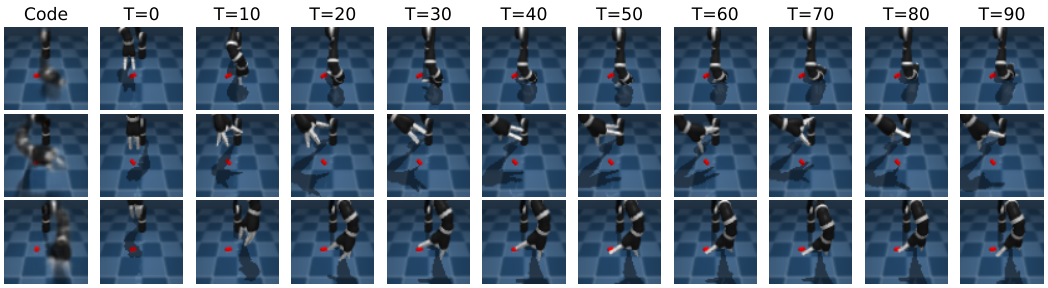


Figure 12: **Jaco.** Sets of three skill codes and respective behaviors visualized for Jaco.

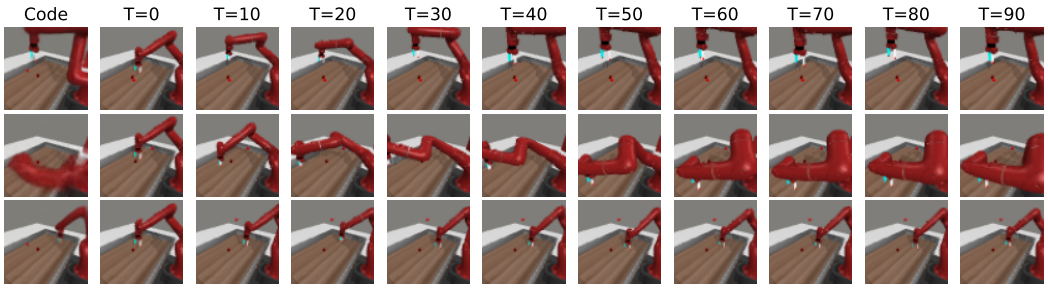


Figure 13: **MetaWorld.** Sets of three skill codes and respective behaviors visualized for MetaWorld.

G ALGORITHM

Algorithm 2: Choreographer

Initialize agent modules: world model, skill autoencoder, skill policies
while *is unsupervised phase* **do**
  `// Pre-training`
  if $t \bmod train_every_n_steps = 0$ **then**
    Draw a batch of past trajectories $\{x, a\}$ from the replay buffer.
    Update world model on batch (Eq. 3) and get states $\{s\}$.
    `// Skill discovery`
    Update skill autoencoder using the model states $\{s\}$ (Eq. 5).
    `// Skill learning`
    Sample skill codes from uniform distribution $z \sim p(z)$.
    Imagine skill trajectories $\{s, a, z\}$, sampling actions from $\pi_{\text{skill}}(a_t|s_t, z)$.
    Compute skill rewards r_{skill} (Eq. 7).
    Update skill actor-critic.
  end
end
`// Fine-tuning`
Reset replay buffer and initialize meta-controller.
while *is supervised phase* **do**
  Update model state with the posterior $s_t \sim q(s_t|s_{t-1}, a_{t-1}, x_t)$.
  Sample skill $z \sim \pi_{\text{meta}}(z|s_t)$ and action $a_t \sim \pi_{\text{skill}}(a_t|s_t, z)$.
  Collect new task observation from the environment x_{t+1}, r_{t+1}
  Add $(x_t, a_t, x_{t+1}, r_{t+1})$ to the replay buffer.
  if $t \bmod train_every_n_steps = 0$ **then**
    Draw batch of past trajectories $\{x, a, r\}$ from the replay buffer.
    Update world model on batch (Eq. 3) and get states $\{s\}$.
    `// Skill adaptation`
    Imagine traj. $\{s, a, z\}$, sampling skills from $\pi_{\text{meta}}(z|s_t)$ and actions from $\pi_{\text{skill}}(a_t|s_t, z)$.
    Predict task rewards r_{task}
    Update meta-controller actor-critic and propagate gradients to skill actors.
  end
end