

# Trying Again Instead of Trying Longer: Prior Learning for Automatic Curriculum Learning

Rémy Portelas, Katja Hofmann, Pierre-Yves Oudeyer

► **To cite this version:**

Rémy Portelas, Katja Hofmann, Pierre-Yves Oudeyer. Trying Again Instead of Trying Longer: Prior Learning for Automatic Curriculum Learning. ICLR 2020 BeTR-RL (Beyond “Tabula Rasa” in Reinforcement Learning ) workshop, Apr 2020, Addis Abeba / Virtual, Ethiopia. hal-03099913

**HAL Id: hal-03099913**

**<https://hal.inria.fr/hal-03099913>**

Submitted on 6 Jan 2021

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# TRYING AGAIN INSTEAD OF TRYING LONGER: PRIOR LEARNING FOR AUTOMATIC CURRICULUM LEARNING

**Rémy Portelas**  
INRIA (FR)  
remy.portelas@inria.fr

**Katja Hofmann**  
Microsoft Research (UK)

**Pierre-Yves Oudeyer**  
INRIA (FR)

## ABSTRACT

A major challenge in the Deep RL (DRL) community is to train agents able to generalize over unseen situations, which is often approached by training them on a diversity of tasks (or environments). A powerful method to foster diversity is to procedurally generate tasks by sampling their parameters from a multi-dimensional distribution, enabling in particular to propose a different task for each training episode. In practice, to get the high diversity of training tasks necessary for generalization, one has to use complex procedural generation systems. With such generators, it is hard to get prior knowledge on the subset of tasks that are actually learnable at all (many generated tasks may be unlearnable), what is their relative difficulty and what is the most efficient task distribution ordering for training. A typical solution in such cases is to rely on some form of Automated Curriculum Learning (ACL) to adapt the sampling distribution. One limit of current approaches is their need to explore the task space to detect progress niches over time, which leads to a loss of time. Additionally, we hypothesize that the induced noise in the training data may impair the performances of brittle DRL learners. We address this problem by proposing a two stage ACL approach where 1) a teacher algorithm first learns to train a DRL agent with a high-exploration curriculum, and then 2) distills learned priors from the first run to generate an "expert curriculum" to re-train the same agent *from scratch*. Besides demonstrating 50% improvements on average over the current state of the art, the objective of this work is to give a first example of a new research direction oriented towards refining ACL techniques over multiple learners, which we call *Classroom Teaching*.

## 1 INTRODUCTION

**Automatic CL.** The idea of organizing the learning sequence of a machine is an old concept that stems from multiple works in reinforcement learning (Selfridge et al., 1985; Schmidhuber, 1991), developmental robotics (Oudeyer et al., 2007) and supervised learning (Elman, 1993; Bengio et al., 2009), from which the Deep RL community borrowed the term *Curriculum Learning* (CL). Automatic CL refers to approaches able to autonomously adapt their task sampling distribution to their evolving learner with minimal expert knowledge. Several ACL approaches have recently been proposed (Florensa et al., 2018; Racanire et al., 2019; OpenAI et al., 2019; Portelas et al., 2019; Colas et al., 2019; Pong et al., 2019; Jabri et al., 2019; Laversanne-Finot et al., 2018).

**ACL and exploration.** One of the limits of ACL is that when applied to a large parameterized task space with few learnable subspaces, as when considering a rich procedural generation system, they lose a lot of time finding the "optimal parameters" at a given point in time (e.g. the niches of progress in Learning Progress-based approaches) through *task exploration*. We also hypothesize that these additional tasks presented to the DRL learner have a cluttering effect on the gathered training data, which adds noise in its already brittle gradient-based optimization and leads to sub-optimal performances.

**Proposed approach.** Given this hypothesized drawback of task exploration, we propose to study whether ACL techniques could be improved by having a two stage approach consisting in 1) a preliminary run with ACL from which prior knowledge on the task space is extracted, and 2) a

second independent run leveraging this prior knowledge to propose a better curriculum to the DRL agent. For its simplicity and versatility, we choose to develop such an approach with ALP-GMM (Portelas et al., 2019), a recent ACL algorithm for continuous task spaces.

**Related work.** Within DRL, *Policy Distillation* (Czarnecki et al., 2019) consists in leveraging a previously trained policy, the "teacher", and use it to perform *behavior cloning* by training a "student" policy to jointly maximize its reward on one or several tasks while minimizing the distance between its action distribution compared to the teacher's. This allows to speed up the learning of bigger architectures and/or to leverage task-experts to train a single learner on a set of tasks. From this point of view, this work can be seen as a complementary approach interested in how to perform *Curriculum Distillation* when considering a continuous space of tasks.

Similar ideas were developed for supervised learning by Hacoen & Weinshall (2019). In their work, authors propose an approach to infer a curriculum from past training for an image classification task: they use a first network trained without curriculum and use its predictive confidence for each image as a difficulty measure that a subsequently trained network uses for curriculum generation. The idea that a knowledge distillation procedure can be beneficial even when the teacher and student policies have identical architectures has also been studied in supervised learning (Furlanello et al., 2018; Yim et al., 2017). In this work, we propose to extend these concepts to DRL scenarios.

## 2 METHODS

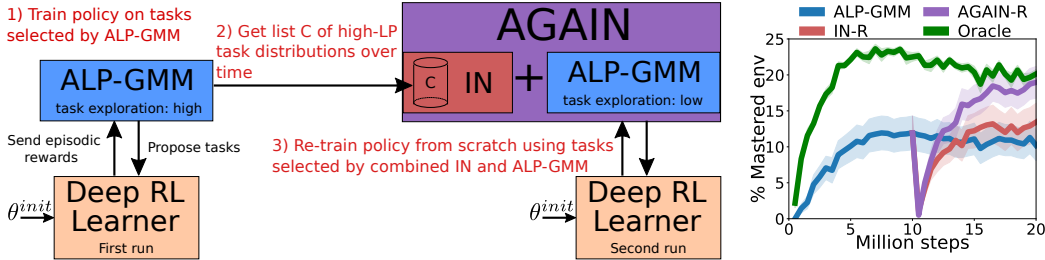


Figure 1: **Left:** Schematic pipeline of Alp-Gmm And Inferred progress Niches (AGAIN), our proposed approach, which first leverages a preliminary run with a high-exploration ALP-GMM curriculum generator to infer an expert curriculum combined with a low-exploration ALP-GMM in a second run with the initial Deep RL learner, re-trained from scratch. **Right:** Given identical training time, our combined approach outperforms regular ALP-GMM and even matches Oracle curriculum in a complex parametric BipedalWalker environment.

**ALP-GMM** ALP-GMM (Portelas et al., 2019) is a Learning Progress (LP) based ACL technique for continuous task spaces that does not assume prior knowledge on the task space. It is inspired by previous works in developmental robotics (Baranes & Oudeyer, 2009; Moulin-Frier et al., 2014). ALP-GMM frames the task sampling problem into an EXP-4 non-stationary Multi-Armed bandit setup (Auer et al., 2002) in which arms are Gaussians spanning over the space of tasks' parameters whose utility is defined with a local LP measure. The essence of ALP-GMM is to periodically fit a Gaussian Mixture Model (GMM) on recently sampled tasks' parameters *concatenated with their respective LP*. Then, the Gaussian from which to sample a new task is chosen proportionally to its mean LP dimension. Task exploration happens initially through a bootstrapping period of random task sampling and during training by occasional random task sampling with probability  $\rho_{rnd}$ .

**Inferred progress Niches (IN)** Using ALP-GMM is convenient for our target experiments as deriving an expert curriculum from an initial run is straightforward: one simply needs to gather the sequence of GMMs  $\mathcal{C}_{raw}$  that were periodically fitted along training:

$$\mathcal{C}_{raw} = \{p(1), \dots, p(T)\} \mid p(t) = \sum_{i=1}^{K_t} LP_{kt} \mathcal{N}(\mu_{ti}, \Sigma_{ti}), \quad (1)$$

with  $T$  the total number of GMMs in the list,  $K_t$  their respective number of components and  $LP_{kt}$  the Learning Progress of each Gaussian. By keeping only Gaussians with  $LP_{kt}$  above a predefined

threshold  $\delta_{LP}$ , we can get a curated list  $\mathcal{C}$ . We name the resulting ACL approach Inferred progress Niches (IN) and propose 3 variants to select which GMM from  $\mathcal{C}$  is used to sample tasks over time during the second run:

- **Pool-based (IN-P)** A rather crude approach is to disregard the ordering of  $\mathcal{C}$  and consider the entire trajectory of GMMs as one single pool  $P$  of Gaussians, ie. one big mixture having  $K_p = \sum_{t=1}^T K_t$  components.
- **Time-based (IN-T)** In this version  $\mathcal{C}$  is stepped in periodically at the same rate than the preliminary ALP-GMM run (ie. a new GMM every 250 episodes in our experiments).
- **Reward-based (IN-R)** Another option is to iterate over  $\mathcal{C}$  only once the mean episodic reward over tasks recently sampled from the current GMM matches or surpasses the mean episodic reward recorded during the initial run (on the same GMM).

Regardless the selection process, given a GMM, a new task is selected by sampling a tasks' parameter on a Gaussian selected proportionally to its  $LP_{kt}$  value.

**Mixing both in AGAIN.** Simply using one of the 3 proposed IN algorithms directly for the second run lacks adaptive mechanisms towards the characteristics of the second agent, whose initial parameters and data training stream are different from the first, which could lead to failure cases where the expert curriculum and the second learner are no longer "in phase". Additionally, if the initial run failed to discover progress niches, IN is bound to fail. As such we propose to combine IN with an ALP-GMM teacher (with low task-exploration) in the second run. The resulting Alp-Gmm And Inferred progress Niches approach, AGAIN for short, samples tasks from a GMM that is composed of the current mixture of both ALP-GMM and IN. See figure 1 for a schematic pipeline and appendix B for details.

### 3 EXPERIMENTS AND RESULTS

**Evaluation procedure.** We propose to test our considered variants and baselines on a parametric version of BipedalWalker proposed by Portelas et al. (2019), which generates walking tracks paved with stumps whose height and spacing are defined by a 2-D parameter vector used for the procedural generation of tasks. This continuous task space has boundaries set in such a way that a substantial part of the space consists in unfeasible tracks for the default walker. As in their work, we also test our approaches with a modified short-legged walker, which constitutes an even more challenging scenario (as the task space is unchanged). All ACL variants are tested when paired with a Soft-Actor Critic (Haarnoja et al., 2018) policy. Performance is measured by tracking the percentage of mastered tasks from a fixed test set. See appendix C for details.

**Is re-training from scratch beneficial?** The end performances of all tested conditions are summarized in table 1. Interestingly, for all tested variants, retraining the DRL agent from scratch in the second run gave superior end performances than fine-tuning using the weights of the first run *in all tested variants*. This showcase the brittleness of gradient-based training and the difficulty of transfer learning. Despite this, even fine-tuned variants reached superior end-performances than classical ALP-GMM, meaning that the change in curriculum strategy in itself is already beneficial.

**Is it useful to re-use ALP-GMM in the second run?** In the default walker experiments, AGAIN-R, T and P conditions mixing ALP-GMM and IN in the second run reached lower mean performances than their respective IN variants. However, the exact opposite is observed for IN-R and IN-T variants in the short walker experiments. This can be explained by the difficulty of short walker experiments for ACL approaches, leading to 16/30 preliminary 10M steps long ALP-GMM runs to have a mean end-performance of 0, compared to 0/30 in the default walker experiments. All these run failures led to many GMMs lists  $\mathcal{C}$  used in IN to be of very low-quality, which illustrates the advantage of AGAIN that is able to emancipate from IN using ALP-GMM.

**Highest-performing variants.** Consistently with the precedent analysis, mixing ALP-GMM with IN in the second run is not essential in default walker experiments, as the best performing ACL approach is IN-P. This most likely suggests that the improved adaptability of the curriculum when

using AGAIN is outbalanced by the added noise (due to the low task-exploration). However in the more complex short walker experiments, mixing ALP-GMM with IN is essential, especially for AGAIN-R, which substantially outperforms ALP-GMM and other AGAIN and IN variants, reaching a mean end performance of 19.0. The difference in end-performance between AGAIN-R and Oracle, our hand-made expert using privileged information who obtained 20.1, is not statistically significant ( $p = 0.6$ ).

Condition	Short walker	Default walker
AGAIN-R	19.0 ± 12.0*	41.6 ± 6.3*
AGAIN-R(fine-tune)	11.4 ± 12.9	39.9 ± 4.6
IN-R	13.4 ± 14.4	43.5 ± 9.6*
IN-R(fine-tune)	11.2 ± 12.3	40.8 ± 5.6
AGAIN-T	15.1 ± 11.9	40.6 ± 11.5
AGAIN-T(fine-tune)	11.4 ± 11.8	40.6 ± 3.8*
IN-T	13.5 ± 13.3	43.5 ± 6.1*
IN-T(fine-tune)	10.7 ± 12.3	40.3 ± 7.6
AGAIN-P	13.6 ± 12.5	41.9 ± 5.1*
AGAIN-P(fine-tune)	11.1 ± 12.0	41.5 ± 3.9*
IN-P	14.5 ± 12.6	<b>44.3</b> ± 3.5*
IN-P(fine-tune)	12.2 ± 12.5	41.1 ± 3.8*
ALP-GMM	10.2 ± 11.5	38.6 ± 3.5
Oracle	<b>20.1</b> ± 3.4*	27.2 ± 15.2 <sup>-</sup>
Random	2.5 ± 5.9 <sup>-</sup>	20.9 ± 11.0 <sup>-</sup>

Table 1: **Experiments on Stump Tracks with short and default bipedal walkers.**

The average performance with standard deviation after 10 Millions steps (IN and AGAIN variants) or 20 Million steps (others) is reported (30 seeds per condition). For IN and AGAIN we also test variants that do not retrain the weights of the policy used in the second run *from scratch* but rather *fine-tune* them from the preliminary run. \*<sup>-</sup> Indicates whether performance difference with ALP-GMM is statistically significant ie.  $p < 0.05$  in a post-training Welch’s student t-test (\* for performance advantage w.r.t ALP-GMM and <sup>-</sup> for performance disadvantage).

## 4 CONCLUSION AND DISCUSSION

In this work we presented Alp-Gmm And Inferred progress Niches, a simple yet effective approach to learn prior knowledge over a space of tasks to design a curriculum tailored to a DRL agent. Instead of following the same exploratory ACL approach over the entire training, AGAIN performs a first preliminary run with ALP-GMM, derives a list of progress niches from it, and uses this list to build an expert curriculum that is combined with a low task-exploration ALP-GMM teacher for a second run of the same DRL agent, trained from scratch.

**Beyond tabula rasa?** In this work we showed that a non-tabula rasa curriculum generator that leveraged prior knowledge over the task space (from a preliminary run) outperformed the regular approach that learned to generate an entire curriculum from scratch. However, we also demonstrated that, from the point of view of the DRL learner, it is actually *better* to restart tabula rasa (with a non-tabula rasa curriculum generator), which is a very interesting perspective and opens several lines for future work.

**Classroom Teaching** Beyond proposing a two-stage ACL technique for a single DRL agent, the experimental setup of this work could be seen as a particular case of a broader problem we propose to name *Classroom Teaching* (CT). CT defines a family of problems in which a *meta-ACL* algorithm is tasked to either sequentially or simultaneously generate multiple curricula tailored for each of the learning students, all having potentially varying abilities. CT differs from the problems studied in population-based developmental robotics (Forestier et al., 2017) and evolutionary algorithms (Wang et al., 2019) as in CT the number and characteristics of learners are predefined, and the objective is to foster maximal learning progress over all learners rather than iteratively constructing high-performing policies. Studying CT scenarios brings DRL closer to human education research problems and might stimulate the design of methods that alleviate the expensive use of expert knowledge in current state of the art assisted education (Clément et al., 2015; Koedinger et al., 2013).

## REFERENCES

- Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. The nonstochastic multi-armed bandit problem. *SIAM journal on computing*, 32(1):48–77, 2002.
- Adrien Baranes and Pierre-Yves Oudeyer. R-IAC: robust intrinsically motivated exploration and active learning. *IEEE Trans. Autonomous Mental Development*, 1(3):155–169, 2009. doi: 10.1109/TAMD.2009.2037513.
- Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, pp. 41–48, 2009. doi: 10.1145/1553374.1553380.
- Hamparsum Bozdogan. Model selection and akaike’s information criterion (aic): The general theory and its analytical extensions. *Psychometrika*, 52(3):345–370, Sep 1987. ISSN 1860-0980. doi: 10.1007/BF02294361.
- Benjamin Clément, Didier Roy, Pierre-Yves Oudeyer, and Manuel Lopes. Multi-Armed Bandits for Intelligent Tutoring Systems. *Journal of Educational Data Mining (JEDM)*, 7(2):20–48, June 2015.
- Cédric Colas, Pierre-Yves Oudeyer, Olivier Sigaud, Pierre Fournier, and Mohamed Chetouani. Curious: Intrinsically motivated modular multi-goal reinforcement learning. In *International Conference on Machine Learning*, pp. 1331–1340, 2019.
- Wojciech Marian Czarnecki, Razvan Pascanu, Simon Osindero, Siddhant M. Jayakumar, Grzegorz Swirszcz, and Max Jaderberg. Distilling policy distillation. *CoRR*, abs/1902.02186, 2019.
- Jeffrey L. Elman. Learning and development in neural networks: the importance of starting small. *Cognition*, 48(1):71 – 99, 1993. ISSN 0010-0277. doi: 10.1016/0010-0277(93)90058-4.
- Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pp. 1514–1523, 2018.
- Sébastien Forestier, Yoan Mollard, and Pierre-Yves Oudeyer. Intrinsically motivated goal exploration processes with automatic curriculum learning. *CoRR*, abs/1708.02190, 2017.
- Tommaso Furlanello, Zachary Chase Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born-again neural networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pp. 1602–1611, 2018.
- Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018.
- Guy Hacohen and Daphna Weinshall. On the power of curriculum learning in training deep networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2535–2544, Long Beach, California, USA, 09–15 Jun 2019. PMLR.
- Allan Jabri, Kyle Hsu, Abhishek Gupta, Ben Eysenbach, Sergey Levine, and Chelsea Finn. Unsupervised curricula for visual meta-reinforcement learning. In *Advances in Neural Information Processing Systems 32*, pp. 10519–10530. Curran Associates, Inc., 2019.
- Kenneth R. Koedinger, Emma Brunskill, Ryan Shaun Joazeiro de Baker, Elizabeth A. McLaughlin, and John C. Stamper. New potentials for data-driven intelligent tutoring system development and optimization. *AI Magazine*, 34(3):27–41, 2013.
- Adrien Laversanne-Finot, Alexandre Pere, and Pierre-Yves Oudeyer. Curiosity driven exploration of learned disentangled goal spaces. In *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pp. 487–504. PMLR, 29–31 Oct 2018.

- Clément Moulin-Frier, Sao Mai Nguyen, and Pierre-Yves Oudeyer. Self-organization of early vocal development in infants and machines: The role of intrinsic motivation. *Frontiers in Psychology (Cognitive Science)*, 4(1006), 2014. ISSN 1664-1078. doi: 10.3389/fpsyg.2013.01006.
- OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jadwiga Tworek, Peter Welinder, Lilian Weng, Qi-Ming Yuan, Wojciech Zaremba, and Lefei Zhang. Solving rubik’s cube with a robot hand. *ArXiv*, abs/1910.07113, 2019.
- Pierre-Yves Oudeyer, Frdric Kaplan, and Verena V Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE transactions on evolutionary computation*, 11(2):265–286, 2007.
- Vitchyr H. Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. *CoRR*, abs/1903.03698, 2019.
- Rmy Portelas, Cdric Colas, Katja Hofmann, and Pierre-Yves Oudeyer. Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments, 2019.
- Sbastien Racanire, Andrew Lampinen, Adam Santoro, David Reichert, Vlad Firoiu, and Timothy Lillicrap. Automated curricula through setter-solver interactions. *arXiv preprint arXiv:1909.12892*, 2019.
- Jrgen Schmidhuber. Curious model-building control systems. In *In Proc. International Joint Conference on Neural Networks, Singapore*, pp. 1458–1463. IEEE, 1991.
- Oliver G. Selfridge, Richard S. Sutton, and Andrew G. Barto. Training and tracking in robotics. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence. Los Angeles, CA, USA, August 1985*, pp. 670–672, 1985.
- Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. Paired open-ended trailblazer (POET): endlessly generating increasingly complex and diverse learning environments and their solutions. *CoRR*, abs/1901.01753, 2019.
- Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7130–7138, 2017.

## A ALP-GMM

ALP-GMM relies on an empirical per-task computation of Absolute Learning Progress (ALP), allowing to fit a GMM on a concatenated space composed of tasks’ parameters and respective ALP. Given a task  $\tau_{new} \in \mathcal{T}$  whose parameter is  $p_{new} \in \mathcal{P}$  and on which the policy collected the episodic reward  $r_{new} \in \mathbb{R}$ , Its ALP is computed using the closest previous tasks  $\tau_{old}$  (Euclidean distance) with associated episodic reward  $r_{old}$ :

$$alp_{new} = |r_{new} - r_{old}| \quad (2)$$

All previously encountered task’s parameters and their associated ALP, parameter-ALP for short, recorded in a history database  $\mathcal{H}$ , are used for this computation. Contrastingly, the fitting of the GMM is performed every  $N$  episodes on a window  $\mathcal{W}$  containing the  $N$  most recent parameter-ALP. The resulting mean ALP dimension of each Gaussian of the GMM is used for proportional sampling. To adapt the number of components of the GMM online, a batch of GMMs having from 2 to  $k_{max}$  components is fitted on  $\mathcal{W}$ , and the best one, according to Akaike’s Information Criterion (Bozdogan, 1987), is kept as the new GMM. In our experiments we use the same hyperparameters as in Portelas et al. (2019) ( $N = 250$ ,  $k_{max} = 10$ ), except for the percentage of random task sampling  $\rho_{rnd}$  which we set to 10% (we found it to perform better than 20%) when running ALP-GMM alone or 2% when combined with IN in the second phase of AGAIN. See algorithm 1 for pseudo-code and figure 2 for a schematic pipeline. Note that in this paper we refer to ALP as LP for simplicity (ie.  $LP_{kt}$  in  $\mathcal{C}$  from eq. 1 is equivalent to the mean ALP of Gaussians in ALP-GMM).

---

### Algorithm 1 Absolute Learning Progress Gaussian Mixture Model (ALP-GMM)

---

**Require:** Student policy  $\pi_\theta$ , parametric procedural environment generator  $E$ , bounded parameter space  $\mathcal{P}$ , probability of random sampling  $\rho_{rnd}$ , fitting rate  $N$ , max number of Gaussians  $k_{max}$

- 1: Initialize  $\pi_\theta$
  - 2: Initialize parameter-ALP First-in-First-Out window  $\mathcal{W}$ , set max size to  $N$
  - 3: Initialize parameter-reward history database  $\mathcal{H}$
  - 4: **loop**  $N$  times ▷ Bootstrap phase
  - 5:     Sample random  $p \in \mathcal{P}$ , send  $E(\tau \sim \mathcal{T}(p))$  to  $\pi_\theta$ , observe episodic reward  $r_p$
  - 6:     Compute ALP of  $p$  based on  $r_p$  and  $\mathcal{H}$  (see equation 2)
  - 7:     Store  $(p, r_p)$  pair in  $\mathcal{H}$ , store  $(p, ALP_p)$  pair in  $\mathcal{W}$
  - 8: **loop** ▷ Stop after  $K$  inner loops
  - 9:     Fit a set of GMM having 2 to  $k_{max}$  kernels on  $\mathcal{W}$
  - 10:     Select the GMM with best Akaike Information Criterion
  - 11:     **loop**  $N$  times
  - 12:          $\rho_{rnd}\%$  of the time, sample a random parameter  $p \in \mathcal{P}$
  - 13:         Else, sample  $p$  from a Gaussian chosen proportionally to its mean ALP value
  - 14:         Send  $E(\tau \sim \mathcal{T}(p))$  to student  $\pi_\theta$  and observe episodic reward  $r_p$
  - 15:         Compute ALP of  $p$  based on  $r_p$  and  $\mathcal{H}$
  - 16:         Store  $(p, r_p)$  pair in  $\mathcal{H}$ , store  $(p, ALP_p)$  pair in  $\mathcal{W}$
  - 17: **Return**  $\pi_\theta$
-



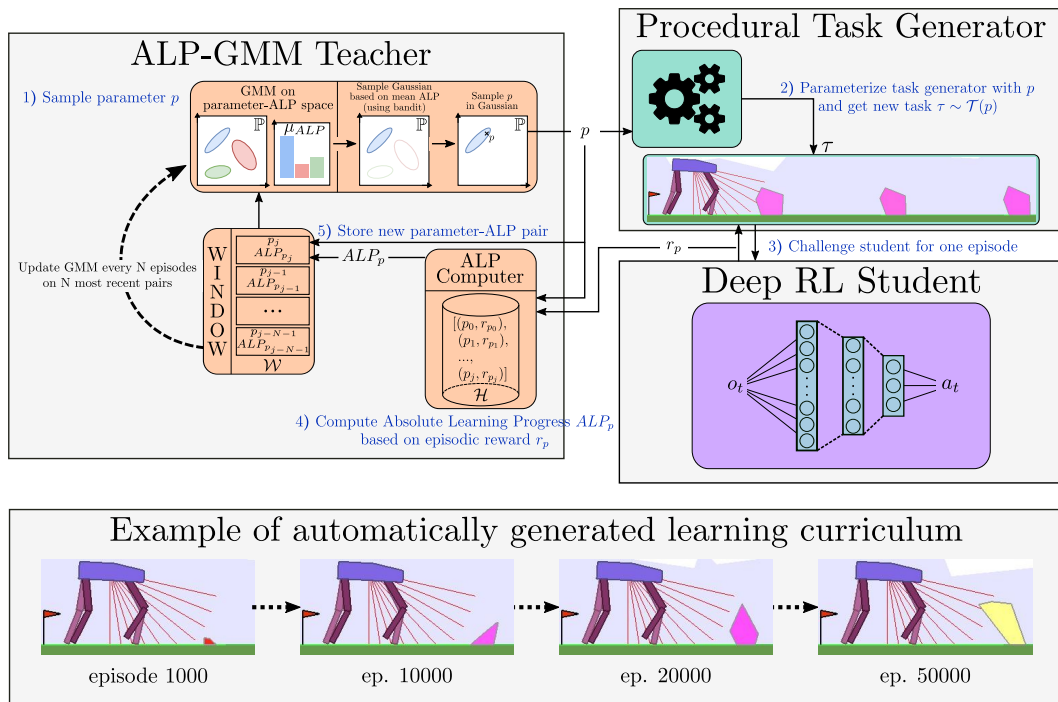


Figure 2: Schematic view of an ALP-GMM teacher’s workflow from Portelas et al. (2019)

## B AGAIN

**IN variants.** In order to filter the list  $\mathcal{C}_{raw}$  (see eq. 1) of GMMs collected after a preliminary run of ALP-GMM into  $\mathcal{C}$  and use it as an expert curriculum, we remove any Gaussian with a  $LP_{kt}$  below  $\delta_{LP} = 0.1$  (the LP dimension is normalized between 0 and 1, which requires to choose an approximate potential reward range, set to  $[-150, 350]$  for all experiments). When all Gaussians of a GMM are discarded, the GMM is removed from  $\mathcal{C}$ . In practice, it allows to remove non-informative GMMs corresponding to the initial exploration phase of ALP-GMM, when the learner has not made any progress (hence no LP detected by the curriculum generator).  $\mathcal{C}$  is then iterated over to generate a curricula with either of the Time-based (see algo 2), Pool-based (see algo 3) or Reward-based (see algo 4) IN. The IN-P approach does not require additional hyperparameters. The IN-T requires an update rate  $N$  to iterate over  $\mathcal{C}$ , which we set to 250 (same as the fitting rate of ALP-GMM). The IN-R approach requires to extract additional data from the first run, in the form of a list  $\mathcal{R}_{raw}$ :

$$\mathcal{R}_{raw} = \{\mu_r^1, \dots, \mu_r^t, \mu_r^T\} \text{ s.t. } |\mathcal{R}_{raw}| = |\mathcal{C}_{raw}|, \quad (3)$$

with  $T$  the total number of GMMs in the first run (same as in  $\mathcal{C}_{raw}$ ), and  $\mu_r^t$  the mean episodic reward obtained by the first DRL agent during the last 50 tasks sampled from the  $t^{th}$  GMM.  $\mathcal{R}$  is simply obtained by removing any  $\mu_r^t$  that corresponds to a GMM discarded while extracting  $\mathcal{C}$  from  $\mathcal{C}_{raw}$ . The remaining rewards are then used as thresholds in IN-R to decide when to switch to the next GMM in  $\mathcal{C}$ .

**AGAIN** In AGAIN (see algo. 5), the idea is to use both IN (R,T or P) and ALP-GMM (without the random bootstrapping period) for curriculum generation. We combine the changing GMM of IN and ALP-GMM over time, simply by building a GMM  $G$  containing Gaussians from the current GMM of IN and ALP-GMM. By selecting the Gaussian in  $G$  from which to sample a new task using their respective LP, This approach allows to adaptively modulate the task sampling between both, shifting the sampling towards IN when ALP-GMM does not detect high-LP subspaces and towards ALP-GMM when the current GMM of IN has low-LP Gaussians. Additionally, to have minimal task-exploration, which benefits ALP-GMM (allowing it to detect new progress niches), we sample random tasks with probability  $\rho_{low} = 2\%$  (compared with  $\rho_{high} = 10\%$  used for the preliminary ALP-GMM run).

---

### Algorithm 2 Inferred progress Niches - Time-based (IN-T)

---

**Require:** Student policy  $\pi_\theta$ , task-encoding parameter space  $\mathcal{P}$ , LP threshold  $\delta_{LP}$ , update rate  $N$ , experimental budget  $K$ , random sampling ratio  $\rho_{high}$ , parametric procedural environment generator  $E$

- 1: Init  $\pi_\theta$ , train it for  $K/2$  env. steps with ALP-GMM( $\rho_{high}, \mathcal{P}$ ) and collect  $\mathcal{C}_{raw}$  ▷ First run
  - 2: Get  $\mathcal{C}$  from  $\mathcal{C}_{raw}$  by removing any Gaussian with  $LP_{kt} < \delta_{LP}$ .
  - 3: Re-initialize  $\pi_\theta$  ▷ Second run
  - 4: Initialize expert curriculum index  $i_c$  to 0
  - 5: **loop** ▷ Stop after  $K/2$  environment steps
  - 6:     Set  $i_c$  to  $\min(i_c + 1, \text{len}(\mathcal{C}))$
  - 7:     Set current GMM  $G_{IN}$  to  $i_c^{th}$  GMM in  $\mathcal{C}$
  - 8:     **loop**  $N$  times
  - 9:         Sample  $p$  from a Gaussian in  $G_{IN}$  chosen proportionally to its  $LP_{kt}$
  - 10:         Send  $E(\tau \sim \mathcal{T}(p))$  to student  $\pi_\theta$
  - 11: **Return**  $\pi_\theta$
-

**Algorithm 3** Inferred progress Niches - Pool-based (IN-P)

**Require:** Student policy  $\pi_\theta$ , task-encoding parameter space  $\mathcal{P}$ , LP threshold  $\delta_{LP}$ , experimental budget  $K$ , random sampling ratio  $\rho_{high}$ , parametric procedural environment generator  $E$

- 1: Init  $\pi_\theta$ , train it for  $K/2$  env. steps with ALP-GMM( $\rho_{high}, \mathcal{P}$ ) and collect  $\mathcal{C}_{raw}$  ▷ First run
- 2: Get  $\mathcal{C}$  from  $\mathcal{C}_{raw}$  by removing any Gaussian with  $LP_{kt} < \delta_{LP}$ .
- 3: Re-initialize  $\pi_\theta$  ▷ Second run
- 4: Initialize pool GMM  $G_{IN}$ , containing all Gaussians from  $\mathcal{C}$
- 5: **loop** ▷ Stop after  $K/2$  environment steps
- 6:     Sample  $p$  from a Gaussian in  $G_{IN}$  chosen proportionally to its  $LP_{kt}$
- 7:     Send  $E(\tau \sim \mathcal{T}(p))$  to student  $\pi_\theta$
- 8: **Return**  $\pi_\theta$

**Algorithm 4** Inferred progress Niches - Reward-based (IN-R)

**Require:** Student policy  $\pi_\theta$ , task-encoding parameter space  $\mathcal{P}$ , LP threshold  $\delta_{LP}$ , memory size  $N$ , experimental budget  $K$ , random sampling ratio  $\rho_{high}$ , parametric procedural environment generator  $E$

- 1: Init  $\pi_\theta$ , train it for  $K/2$  env. steps with ALP-GMM( $\rho_{high}, \mathcal{P}$ ) and collect  $\mathcal{C}_{raw}$  ▷ First run
- 2: Get  $\mathcal{C}$  from  $\mathcal{C}_{raw}$  by removing any Gaussian with  $LP_{kt} < \delta_{LP}$ .
- 3: Additionally, collect list of inferred reward thresholds  $\mathcal{R}_{raw}$ , and get  $\mathcal{R}$  ▷ See eq. 3
- 4: Re-initialize  $\pi_\theta$  ▷ Second run
- 5: Initialize reward First-in-First-Out window  $\mathcal{W}$ , set max size to  $N$
- 6: Initialize expert curriculum index  $i_c$  to 0
- 7: **loop** ▷ Stop after  $K/2$  environment steps
- 8:     If  $\mathcal{W}$  is full, compute mean reward  $\mu_w$  from  $\mathcal{W}$
- 9:     If  $\mu_w$  superior to  $i_c^{th}$  reward threshold in  $\mathcal{R}$ , set  $i_c$  to  $\min(i_c + 1, \text{len}(\mathcal{C}))$
- 10:     Set current GMM  $G_{IN}$  to  $i_c^{th}$  GMM in  $\mathcal{C}$
- 11:     Sample  $p$  from a Gaussian in  $G_{IN}$  chosen proportionally to its  $LP_{kt}$
- 12:     Send  $E(\tau \sim \mathcal{T}(p))$  to student  $\pi_\theta$  and add episodic reward  $r_p$  to  $\mathcal{W}$
- 13: **Return**  $\pi_\theta$

**Algorithm 5** Alp-Gmm And Inferred progress Niches (AGAIN)

**Require:** Student policy  $\pi_\theta$ , task-encoding parameter space  $\mathcal{P}$ , random sampling ratio  $\rho_{low}$  and  $\rho_{high}$ , LP threshold  $\delta_{LP}$ , experimental budget  $K$ , parametric procedural environment generator  $E$

- 1: Init  $\pi_\theta$ , train it for  $K/2$  env. steps with ALP-GMM( $\rho_{high}, \mathcal{P}$ ) and collect  $\mathcal{C}_{raw}$  ▷ First run
- 2: Get  $\mathcal{C}$  from  $\mathcal{C}_{raw}$  by removing any Gaussian with  $LP_{kt} < \delta_{LP}$
- 3: re-initialize  $\pi_\theta$  ▷ Second run
- 4: Setup new ALP-GMM( $\rho = 0, \mathcal{P}$ ) ▷ See algo. 1
- 5: Setup either IN-T, IN-P or IN-R ▷ See algo. 2, 3 and 4
- 6: **loop** ▷ Stop after  $K/2$  environment steps
- 7:     Get composite GMM  $G$  from the current GMM of both ALP-GMM and IN
- 8:      $\rho_{low}\%$  of the time, sample a random parameter  $p \in \mathcal{P}$
- 9:     Else, sample  $p$  from a Gaussian chosen proportionally to its  $LP$
- 10:     Send  $E(\tau \sim \mathcal{T}(p))$  to student  $\pi_\theta$  and observe episodic reward  $r_p$
- 11:     Send  $(p, r_p)$  pair to both ALP-GMM and IN
- 12: **Return**  $\pi_\theta$

## C EXPERIMENTAL DETAILS

**Soft Actor-Critic** In our experiments, we use an implementation of Soft Actor-Critic provided by OpenAI<sup>1</sup>. We use a 2 layered (400,300) network for V, Q1, Q2 and the policy. Gradient steps are performed each 10 environment steps, with a learning rate of 0.001 and a batch size of 1000. The entropy coefficient is set to 0.005.

**Parametric BipedalWalker** Our proposed ACL variants choose parameters of tasks that encode the procedural generation of walking tracks paved with stumps in the BipedalWalker environments. As in Portelas et al. (2019), we bound the height dimension to  $[0, 3]$  and the spacing dimension to  $[0, 6]$  (regardless the walker morphology). The agent is rewarded for keeping its head straight and going forward and is penalized for torque usage. The episode is terminated after 1) reaching the end of the track, 2) reaching a maximal number of 2000 steps, or 3) head collision (for which the agent receives a strong penalty). See figure 3 for visualizations.

**Baselines** The Random curriculum baseline samples tasks’ parameters randomly over the parameter space. The Oracle condition is a hand-made curriculum that is very similar to IN-R, except that the list  $\mathcal{C}$  is built using expert knowledge, and all reward thresholds  $\mu_r^i$  in  $\mathcal{R}$  are set to 230, which is an episodic reward value often used in the literature as characterizing a default walker having a ”reasonably efficient” walking gate (Wang et al., 2019). Basically, Oracle starts proposing tasks from a Gaussian (with std of 0.05) located at the simplest subspace of the task space (ie. low stump height and high stump spacing) and then gradually moves the Gaussian towards the hardest subspaces (high stump height and low stump spacing) by small increments (50 steps overall) happening whenever the mean episodic reward of the DRL agent over the last 50 proposed tasks is superior to 230. In our experiments, consistently with (Portelas et al., 2019), which implements a similar approach, Oracle is prone to forgetting due to the strong shift in task subspace (which is why it is not the best performing condition for default walker experiments (see table 1).

**Computational resources.** To perform our experiments, we ran each condition for either 10 (IN and AGAIN variants) or 20 (others) Millions environment steps (30 repeats) using one cpu and one GPU (the GPU is shared between 8 runs), for approximately 30 hours of wall-clock time. It amounts to 16200 CPU-hours and 2025 GPU-hours. The preliminary ALP-GMM runs used in IN and AGAIN variants correspond to the first 10 Million steps of the ALP-GMM condition (whose end-performance after 20 Million steps is reported in table 1.

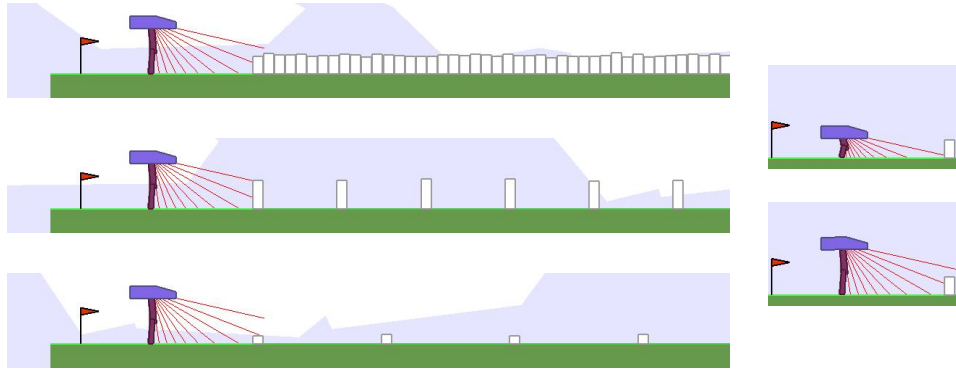
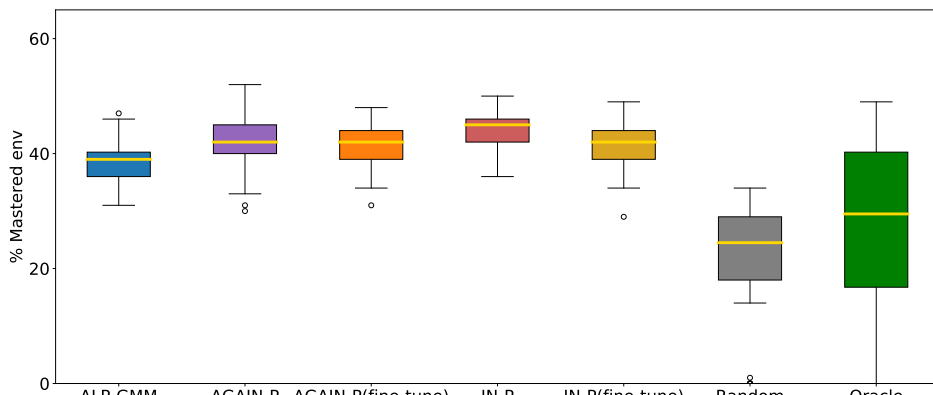


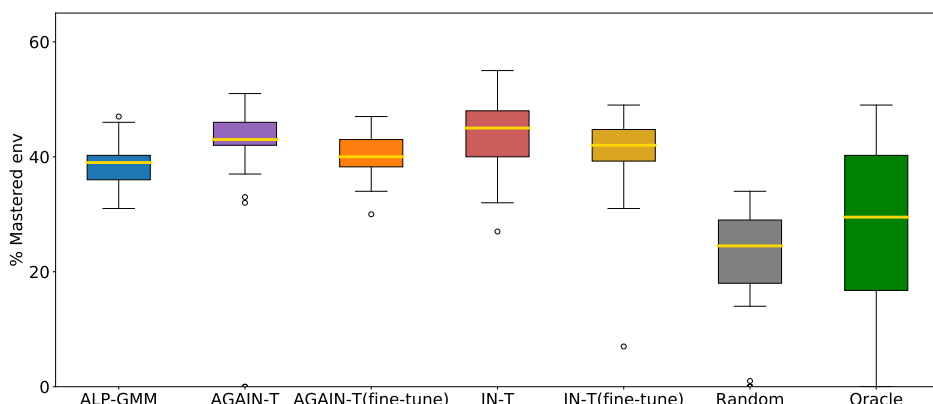
Figure 3: **Parameterized BipedalWalker environment.** **Left:** Examples of generated tracks. **Right:** The two walker morphologies tested on the environment. One parameter tuple  $(\mu_h, \delta_s)$  actually encodes a *distribution* of tasks as the height of each stump along the track is drawn from  $\mathcal{N}(\mu_h, 0.1)$ .

<sup>1</sup><https://github.com/openai/spinningup>

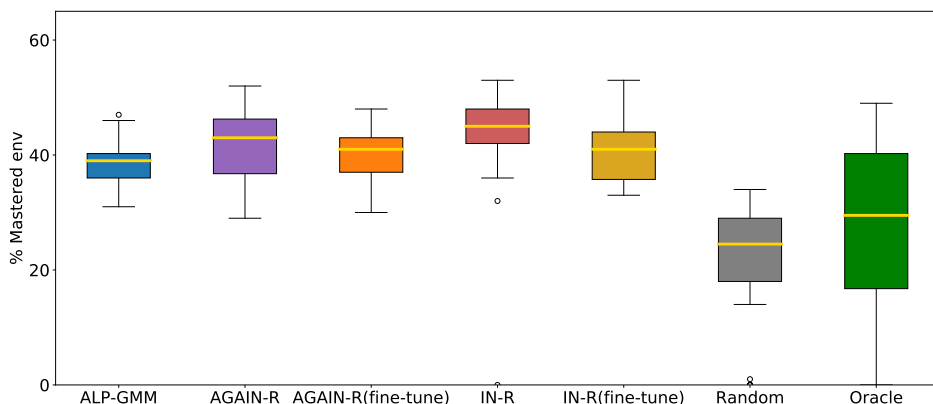
### D ADDITIONAL VISUALIZATIONS



(a) with **Pool-based** IN



(b) with **Time-based** IN



(c) with **Reward-based** IN

Figure 4: Box plots of the final performance of each condition with default bipedal walker after 20M environment steps. Gold lines are medians, surrounded by a box showing the first and third quartile, which are then followed by whiskers extending to the last datapoint or 1.5 times the inter-quartile range. Beyond the whiskers are outlier datapoints. From top to bottom, each box plot presents results when using either Pool-based, Reward-based, or Time-based IN, respectively.

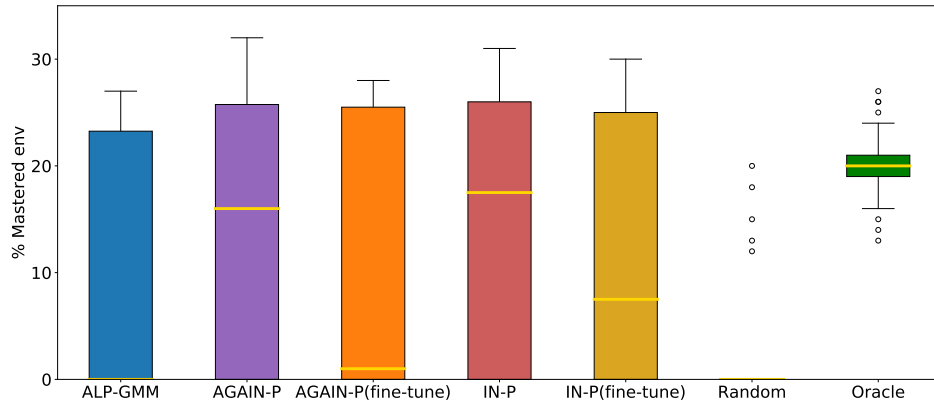
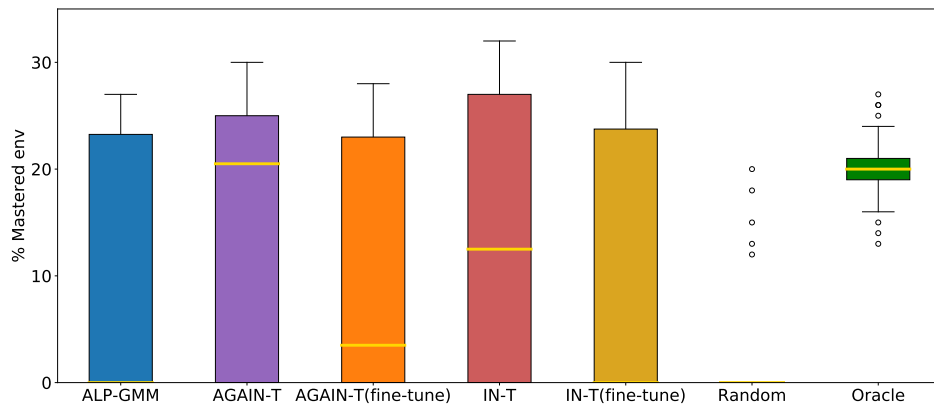
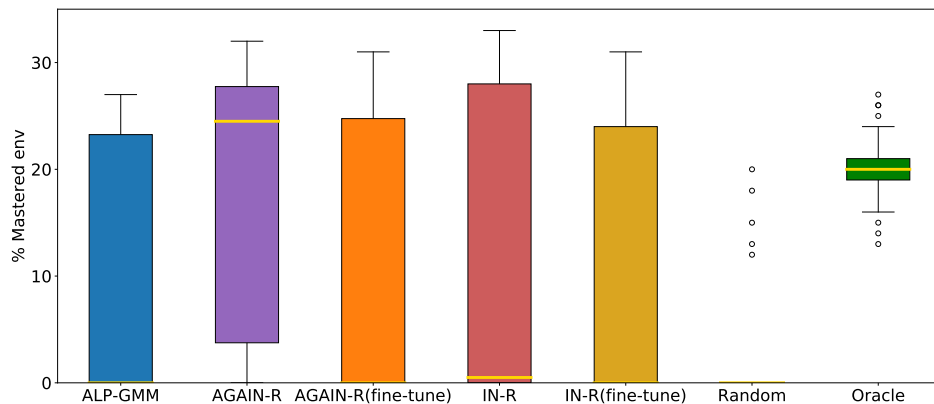
(a) with **Pool-based IN**(b) with **Time-based IN**(c) with **Reward-based IN**

Figure 5: **Box plots of the final performance of each condition with short bipedal walker after 20M environment steps.** Gold lines are medians, surrounded by a box showing the first and third quartile, which are then followed by whiskers extending to the last datapoint or 1.5 times the inter-quartile range. Beyond the whiskers are outlier datapoints. From top to bottom, each box plot presents results when using either Pool-based, Reward-based, or Time-based IN, respectively.

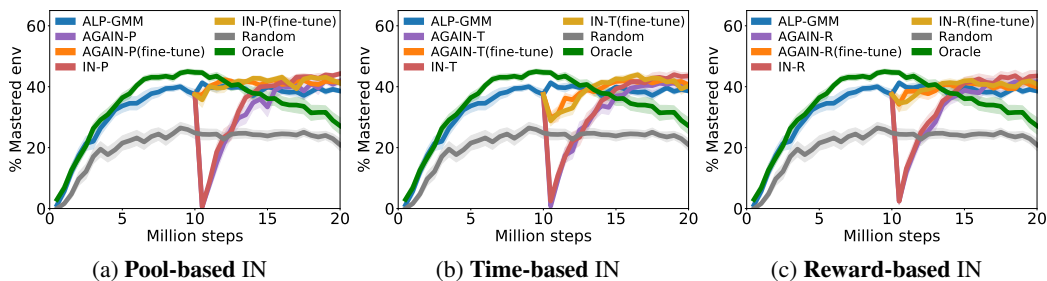


Figure 6: **Evolution of performance across 20M environment steps of each condition with default bipedal walker.** Each point in each curve corresponds to the mean performance (30 seeds), defined as the percentage of mastered tracks (ie.  $r > 230$ ) on a fixed test set. Shaded areas represent the standard error of the mean.

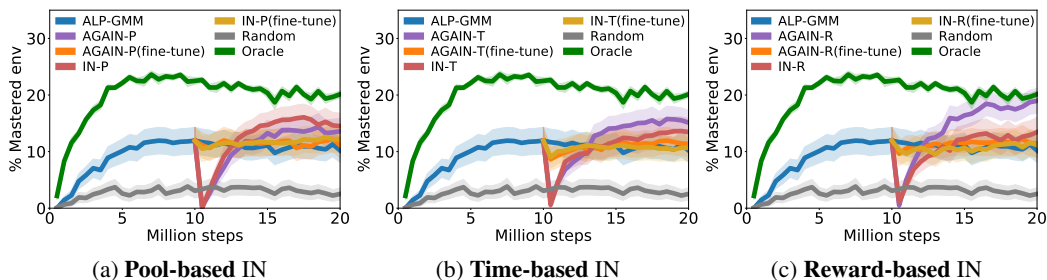


Figure 7: **Evolution of performance across 20M environment steps of each condition with short bipedal walker.** Each point in each curve corresponds to the mean performance (30 seeds), defined as the percentage of mastered tracks (ie.  $r > 230$ ) on a fixed test set. Shaded areas represent the standard error of the mean.