Higher Order and Self-Referential Evolution for Population-based Methods

Samuel Coward
University of Oxford
scoward@robots.ox.ac.uk

Chris Lu
University of Oxford

Alistair Letcher
University of Oxford

Minqi Jiang
University College London

Jack Parker-Holder
University of Oxford

Jakob Nicolaus Foerster
University of Oxford

Abstract

Due to their simplicity and support of high levels of parallelism, evolutionary algorithms have regained popularity in machine learning applications such as curriculum generation for Reinforcement Learning and online hyperparameter tuning. Yet, their performance can be brittle with respect to evolutionary hyperparameters, e.g. the mutation rate. To address this, *self-adaptive* mutation rates, i.e. mutation rates that also evolve, have previously been proposed. While this approach offers a partial solution, it still relies on an a priori set meta-mutation rates. Inspired by recent work Lu et al. [2023], which demonstrates specific cases where evolution is able to implicitly optimize for *higher-order meta-mutation* rates, we investigate whether these higher-order mutations can make evolutionary algorithms more robust and improve their overall performance. We also analyse *self-referential* mutations, which mutate the final order meta-mutation parameter. Our results show that self-referential mutations improve robustness to initial hyperparameters in Population-based Training (PBT) for online hyperparameter tuning, and curriculum learning using Unsupervised Environment Design (UED). We also observe that *self-referential* mutations result in more complex adaptation in competitive multi-agent settings. Our research presents first steps towards robust fully self-tuning systems that are hyperparameter free.

1 Introduction

Evolutionary algorithms refer to the class of methods that optimize an objective by: maintaining a population of solutions and, selecting and stochastically mutating high-performing solutions; while removing low performing solutions from the population. Due to their simplicity, and ease of parallelization, they have been widely adopted in machine learning settings such as hyperparameter tuning [Jaderberg et al., 2017] and curriculum generation for Reinforcement Learning [Parker-Holder et al., 2022]. Central to evolutionary algorithms is the mutation operator, the mechanism by which the population explores the space of solutions. This mutation operator is parameterized by a mutation rate and in general no single, fixed mutation rate works well on all domains, restricting the applicability of these methods without any tuning.

To overcome this, one promising direction is to augment evolutionary algorithms with *self-adaptive* mutation rates. Here, each individual in the population possesses its own mutation rate, which is then co-evolved using a *fixed* meta-mutation rate [Kumar et al., 2022]. These methods are particularly attractive because they potentially enable evolutionary algorithms to be applicable 'out-of-the-box' across a broader array of domains. However, the introduction of a fixed meta-mutation operator shifts performance sensitivity from the mutation rate to the meta-mutation rate. This naturally begs the question whether stopping at one level of meta-mutation is the right answer and if it might even be possible to get rid of fixed hyperparameters all together.

For a specific fitness function and mutation operator pair recent work [Lu et al., 2023] proved that higher-order self-adaptive mutation rates can be implicitly optimized through "standard" (zeroth-order) evolution. Building on this work, we investigate to what extent higher-order self-adaptive mutation rates can improve the robustness and generality of evolutionary algorithms when applied to *practical machine* learning settings. Additionally, to eliminate the final order meta-mutation parameter entirely, we also analyze *self-referential* mutations, where the final meta-mutation parameter is mutated by itself. We evaluate our methods on Population-Based Training (PBT) for online hyperparameter tuning and Unsupervised Environment Design (UED) for curriculum generation, both of which employ evolutionary algorithms for achieve state-of-the-art results. Additionally, we also evaluate them on competitive multiagent settings.

This work can be summarised as follows:

1. We generalize Lu et al. [2023] proof to a larger class of fitness functions and mutation operators.
2. We empirically analyze under what scenarios higher-order self-adaptive and self-referential mutations improve rates of convergence in toy settings.
3. We empirically evaluate higher-order self-adaptive and self-referential mutations in a PBT and UED settings. In particular, we find that self-referential mutations improve hyperparameter robustness in a non-stationary PBT tasks and in UED.
4. Finally, we find higher-order meta-parameters result in more efficient adaptation in competitive multi-agent learning tasks, resulting in better performance.

Ultimately, this has the potential to pave the way for self-tuning methods which have the potential to drastically improve the robustness of ML methods and reduce the dependence on hyperparameters.

2 Related Work

Self-adaptive Evolutionary Optimization: Several prior works have applied self-adaptive mutation rates to stationary optimization problems [Bäck et al., 1992, Gomez, 2004]. These methods are promising and evolutionary plausible. However, they are prone to premature convergence due to the risk of getting stuck in local optima: any random change is expected to decrease fitness, which encourages the population to mutate as little as possible [Rudolph, 2001]. Some works attempt to address this by using separate populations [Kumar et al., 2022], ad hoc systems [Kramer, 2010], or storing a large covariance matrix [Hansen and Ostermeier, 2001]. Instead, we believe that self-adaptive mutation is best suited for the nonstationary and competitive settings rather than the stationary problems previously explored.

Multi-Level Reasoning in Multi-Agent Interactions: Other works have investigated the multi-level reasoning that emerges from multi-agent interactions. For example, the Cognitive Hierarchies [Camerer et al., 2004] framework and its related instances, such as K-Level reasoning [Costa-Gomes and Crawford, 2006], train a hierarchy of best response agents. This has been investigated in zero-shot coordination settings [Cui et al., 2021]. Instead, our work investigates multi-level online *adaptation*, in which we do not calculate a stationary best response.

Higher-Order and Self-Referential Meta-Learning: Prior work has also investigated higher-order meta-learning for optimiser hyperparameters [Chandra et al., 2022] and opponent shaping [Willi et al., 2022]. Unlike our work, these works use gradient-based methods, which limits their applicability. Other works use evolution-based algorithms to meta-learn optimizers [Metz et al., 2021], RL algorithms [Lu et al., 2022a, Jackson et al., 2023a,b], or evolutionary algorithms [Lange et al., 2023] with some preliminary results on self-referential learning. These works use computationally expensive bi-level optimisation schemes to perform meta-optimisation, which quickly becomes intractable for higher orders.

To the best of our knowledge, self-referential learning for self-improvement was first articulated in [Schmidhuber, 1987]. Follow-up work presented self-referential neural architectures [Irie et al., 2022] and algorithms [Kirsch and Schmidhuber, 2022] that explicitly avoid handcrafted optimisation with an evolution-like approach. Our work is an instantiation of this overall approach via a low-overhead, practical modification to any evolutionary algorithm.

3 Background

3.1 Genetic Algorithms

Genetic algorithms are a class of methods for maximizing an objective f by evolving a population of candidate solutions $\mathcal{P} = \{x_i\}_{i=1}^N$. At each evolutionary iteration (i.e. generation), the highest performing solutions survive. The remainder of the population is replaced by mutations of the survivors. Mutations are typically stochastic, and are characterized by a mutation operator M such that an offspring x'_i is generated by sampling $x'_i \sim M(x_i; \sigma)$, where σ is the mutation rate hyperparameter. For example, a Gaussian mutation operator $M(x_i; \sigma) = \mathcal{N}(x_i; \sigma)$ is commonly used where σ represents the variance.

3.2 Non-stationary Objective

The objective is non-stationary when, for generation t, the objective depends on some unseen context c_t; that is, the objective at each generation is some function f_{c_t}. We refer to optimization of such objectives as *non-stationary optimization*. For instance, the context may represent time-varying latent parameters. Under the assumption that the objectives induced by temporally local contexts have similar maximizers, population-based evolution can bootstrap from high performing solutions of the previous generation, avoiding the need to optimize from scratch. However, to effectively bootstrap from the previous generation, the mutation operation must be able to, in some sense, predict changes to the objective landscape. For example, if the distance between maximizers of subsequent generations exceeds the maximum delta achievable by mutation, then the population cannot "keep up" with the rate of change of the objective landscape, and thus may eventually fail.

3.3 Self-adaptive Mutation Rates

Mutations rates are made self-adaptive by associating each solution in the population with its own mutation rate Kumar et al. [2022], which itself is subject to meta-mutations. For a population of solution-mutation rate pairs $\mathcal{P} = \{(x_i, \sigma_i)\}_{i=1}^N$, offspring are generated by

$$x'_i \sim M(x_i, \sigma_i)$$
$$\sigma'_i \sim M_{\text{META}}(\sigma_i, \sigma_{\text{META}}),$$

where M_{META} is the meta-mutation operator parameterized by a *fixed* meta-mutation rate σ_{META}.

3.4 Higher-order Self-Adaptation

Lu et al. [2023] introduced higher-order self-adaptive evolution for real-valued objectives. Individuals in the population were represented by the concatenation of a scalar solution x and n-orders of scalar meta-mutation rates $(x, \sigma_1, \ldots, \sigma_n)$, where only x contributed to fitness. Upon selection for mutation, offspring were sampled as follows:

$$x' \sim \mathcal{N}(x + \sigma_1, \beta), \tag{1}$$
$$\sigma'_i \sim \mathcal{N}(\sigma_i + \sigma_{i+1}, \beta), i < n, \tag{2}$$
$$\sigma'_n \sim \mathcal{N}(\sigma_n, \beta) \tag{3}$$

where β is a scalar noise hyperparameter. Mutations are made *self-referential* by allowing the last-order meta-mutation rate to be mutated subject to itself.

The authors prove, for this specific mutation operation and some integer $k > 1$, evolution using top-k selection implicitly optimizes higher-order mutation rates. Additionally, they show such mutations yield improved mean fitness over time of the population when applied to synthetic time-series forecasting.

4 Method

4.1 Higher-order Self-Adaptive Mutations

We generalize higher-order self-adaptive mutation rates presented in Lu et al. [2023]. We define higher-order mutation parameters as parameters associated with each solution of the population that, control: how a given solution of a population mutates, how its associated mutation rate mutates, how its associated meta-mutation rat mutates, etc. Crucially, higher-order mutation rates do not *directly* contribute to the fitness of an individual, but only indirectly through their impact on its evolution. Each member can thus be expressed as the concatenation of the solution, which directly determines the fitness, followed by a sequence of higher-order mutation parameters $(x, \sigma_1, \ldots, \sigma_n)$. Mutation is performed as follows:

$$x' \sim M_0(x; \sigma_1), \tag{4}$$

$$\sigma_i' \sim M_i(\sigma_i; \sigma_{i+1}), \forall i < n, \tag{5}$$

$$\sigma_n' \sim M_n(\sigma_n; \sigma_{\text{META}}) \tag{6}$$

where M_i is the i-th order parameterized meta-mutation operator, σ_{META} is the mutation rate for the last-order meta-mutation rate. Note that the self-adaptive mutation paradigm introduced above is a specific instance of this framework (by setting $n = 1$).

4.2 Self-referential Mutations

Higher-order self-adaptive mutations still require hyperparameter σ_{META} to be predetermined. Similarly to Lu et al. [2023], we can eliminate σ_{META} by making the last-order meta-mutation rate mutate *self-referential*:

$$\sigma_n' \sim M_n(\sigma_n; \sigma_n).$$

That is, the mutation rate of the last-order meta-mutation rate is itself.

5 Theoretical Results

Lu et al. [2023] proof of implicit optimization of higher-order mutation parameters is limited to scalar objectives. We expand their proof to incorporate real-vector input objective functions and mutation operators.

Theorem 1. Let a population of members be given by $\mathbf{x} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \in \mathbb{R}^d$ with $\mathbf{x}_i \in \mathbb{R}^{d_i}$ and $\sum_{i=1}^n d_i = d$, and for mutation operations of the form:

$$\mathbf{x}_i^{t+1} \leftarrow \mathbf{x}_i^t + G_i(\mathbf{x}_{i+1}^t) + B_i,$$

for $B_i \sim \mathcal{N}(0; \beta)$, some predetermined β, $G_i : \mathbb{R}^{d_{i+1}} \to \mathbb{R}^{d_i}$ is some deterministic function. All else equal, an individual with a higher-order mutation parameter that results in higher performance is more likely selected.

We refer the reader to Appendix B for a proof and formal description of the theorem.

6 Synthetic Tasks

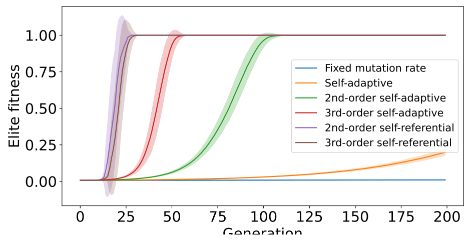
We empirically test higher-order self-adaptive and self-referential mutations on synthetic optimization tasks. For each task we use a population of size 128, where each individual's solution and mutation rates are initialized by sampling i.i.d from $\mathcal{N}(0, 10^{-10}\mathbf{I})$. At each evolutionary iteration, the top 10% performing individuals survive to the next generation while the remaining individuals are replaced by mutated variants of uniformly sampled survivors (via Section 4 with $M_i(\sigma_i; \sigma_{i+1}) = \mathcal{N}(\sigma_i, \text{diag}(\sigma_{i+1}^2))$ for all i).

We consider the following objectives:

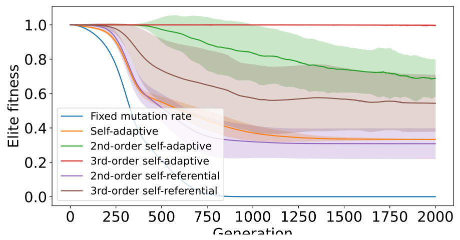
- **Gaussian ring**: $f(\mathbf{x}) = \exp\left(-5\left(\sqrt{\sum_{i=1}^{100} x_i^2} - 1\right)^2\right)$
- **Moving Gaussian**: $f_t(\mathbf{x}) = \frac{1}{3}\sum_{i=1}^{3} \exp\left(-10(x_1 - \alpha_i t^i)^2\right)$
- **Adversarial Gaussian Ring**: $f_{\mathbf{m}}(\mathbf{x}) = \exp\left(-5\left(\sqrt{\sum_{i=1}^{100}(x_i - m_i)^2} - 1\right)^2\right)$

where $\alpha_{\{1,2,3\}} = \{10^{-3}, 10^{-6}, 10^{-8}\}$, t is the evolutionary iteration (i.e. generation count), $\mathbf{m}$ is the elite (i.e. highest performing solution) of the previous generation. Note that the "Moving Gaussian" and "Adversarial Gaussian Ring" objectives are non-stationary.

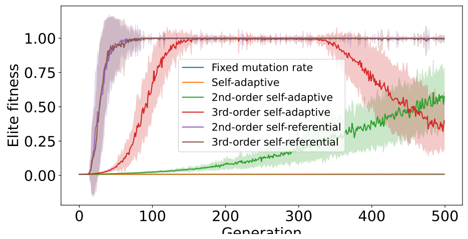
Results are shown in Figure 1. For the Gaussian ring stationary objective, we observe higher-order self-adaptation and self-referential mutations can speed up the rate of convergence. For non-stationary settings, higher-order mutations are far superior at tracking the objective maximizer.



(a) Gaussian ring



(b) Moving Gaussian



(c) Adversarial Gaussian Ring

Figure 1: Plot of mean elite fitness (i.e. top performing solution) in population over the course of evolution, measured over 40 random seeds, for synthetic benchmarks. Error bars correspond to standard deviation. For each task, fitness is bounded between $[0, 1]$.

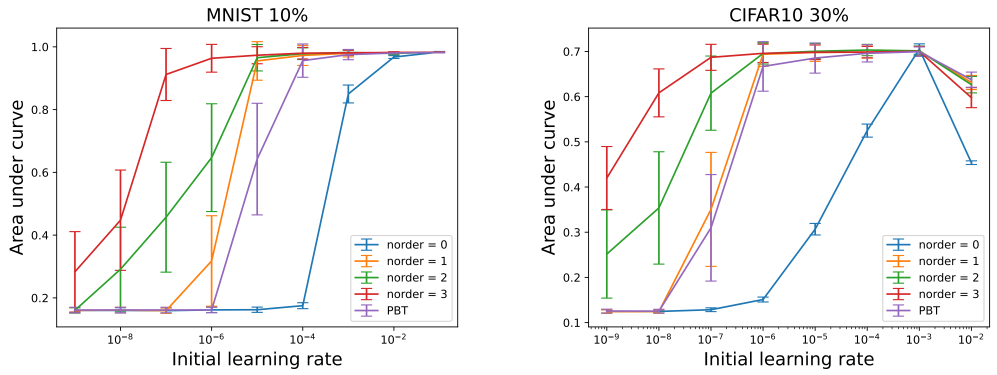


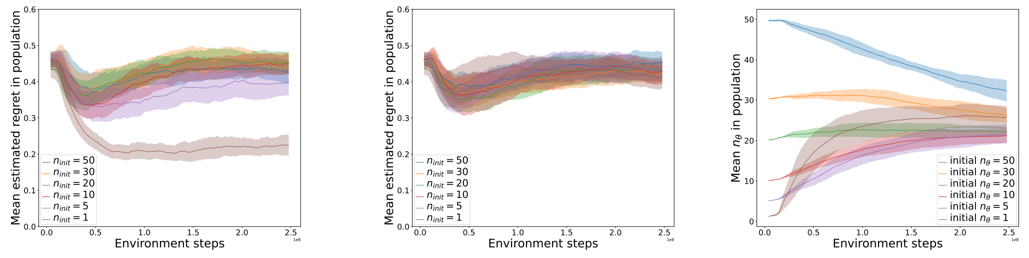
Figure 2: Performance sensitivity over the MNIST task (left) and CIFAR task (right). At the start of each generation, labels are permuted. Each member then trains on a subset of the training data. x-axis corresponds to initial learning rate. y-axis corresponds to mean performance over the course of training. The plots demonstrate that by increasing the number of order of meta-mutation parameters, we improve robustness to poor initialization. Each plot is measured over 5 random seeds; error bars correspond to standard error. Note that order 0 corresponds to performing PBT without exploration; that is, performing exploitation only.

7 Experiments

In this section, we augment practical evolutionary algorithms with higher-order self-adaptive and self-referential mutations.

7.1 Applied to Population Based Training

Population-based Training (PBT) [Jaderberg et al., 2017] is an evolutionary approach to online hyperparameter tuning. PBT maintains a population of model parameter and hyperparameters. Each evolutionary iteration, the population of models are partially trained subject to their corresponding hyperparameters; the highest performing model-hyperparameter pairs are then copied, and their hyperparameters are mutated.



(a) Vanilla ACCEL (b) Self-referential ACCEL (c) Evolution of Mutation Rates

Figure 3: (a) Each plot corresponds to the mean estimate of regret for levels in population over the course of training, where the number of edits applied to levels during mutation is fixed to n_{init}. When n_{init} is small, the evolutionary process is too slow to effectively search level space. When n_{init} is large, search is effectively random (mutated levels will differ drastically from their parent). (b) Each plot corresponds to the mean estimate of regret for levels in population over the course of training, where the number of edits applied to levels during mutation are themselves self-referentially mutable. We observe that regardless of the choice of n_{init}, high-estimated regret levels are consistently discovered. (c) plots showing the mean n_θ of the population of levels over the course of training; that is, the number of discrete edits applied to the corresponding level when sampling mutations. Each plot corresponds to a different n_{init}, measured over 5 random seeds with error bars corresponding to 1 standard deviation. We see that regardless of the choice of n_{init}, the population converges towards an n_θ matching the empirical best reported in Jiang et al. [2023]. Weighted score of level buffer over the course of training a recurrent policy using ACCEL. Each plot corresponds to a different choice of n_{init}. Training was conducted over 2.5×10^8 environment interactions. Each plot is measured over 5 random seeds; error bars are 1 standard deviation. (a, b, c) Error bars correspond to standard deviation measured over 5 random seeds.

We extend the standard PBT setting so that hyperparameters are associated with a set of higher-order mutation parameters, and enable hyperparameters to mutate self-referentially. We consider two supervised learning benchmarks: MNIST and CIFAR10. Each model is trained using a simple 2-layer convolutional neural network followed by a feedforward network. Models are trained using Stochastic Gradient Descent (SGD). At the start of every generation, each network resets the weights of final dense layer, and the target labels are permuted; in essence making the task an online learning problem. Each member is then trained on only 10% (MNIST) or 30% (CIFAR) of the training data. We associate the learning rate η with a sequence of higher order mutation parameters $\eta_1, \dots \eta_n$ for some order n, where η_1 corresponds to learning rate. During exploration, the learning rate and higher order mutation parameters are mutated as such:

$$\eta_i' \sim \mathcal{N}(\eta_i; \eta_{i+1}), \forall i < n$$
$$\eta_n' \sim \mathcal{N}(\eta_n; \eta_n),$$

To eliminate tuning of higher order mutation parameters, we initialize $\eta_i = 10 \cdot \eta_{i-1}$, for $i > 1$. As an additional baseline, we also compare against the original mutation operator proposed in the PBT paper, where, during exploration, hyperparameters are perturbed by a random scale factor of either 0.8 or 1.2. Figure 2 demonstrates that the inclusion of higher-order mutation parameters is effective at overcoming poor initialization, reducing the need for hyperparameter tuning.

7.2 Applied to Unsupervised Environment Design

Unsupervised Environment Design (UED) [Dennis et al., 2020] is a paradigm in Reinforcement Learning that aims to produce "robust" policies through automatic generation of environment curricula. UED formally concerns Underspecified POMDPs [Dennis et al., 2020], defined as $\mathcal{M} = (A, O, \Theta, S, \mathcal{T}, \mathcal{I}, \mathcal{R}, \gamma)$, where: A is the action space, O is the observation space, Θ is the space of underspecified parameters referred to as *levels*, S is the set state space, $\mathcal{T} : S \times A \times \Theta \to \Delta(S)$ is the level-conditioned transition function, $\mathcal{I} : S \to O$ is the mapping of states of observations, $\mathcal{R} : S \to \Delta(\mathbb{R})$ is the reward function, γ is the discount factor. At each stage of training, the goal of UED is to select levels in Θ that maximize some design objective, typically regret: the delta between expected return of an optimal policy on some level and the expected return of the current learning agent [1]. As such, regret-based UED is a natural example of a nonstationary, adversarial objective; repeated presentation of the same level to a learning agent will diminish its regret.

ACCEL [Parker-Holder et al., 2022] is an evolutionary approach to UED that optimizes regret by maintaining a population of levels and mutating high performing levels subject to an *fixed* a priori mutation operator. ACCEL's sensitivity to choice of mutation operator is illustrated in 4. We extend ACCEL by parameterizing the mutation operator and associating each level with (higher-order) mutation parameters which can freely self-adapt.

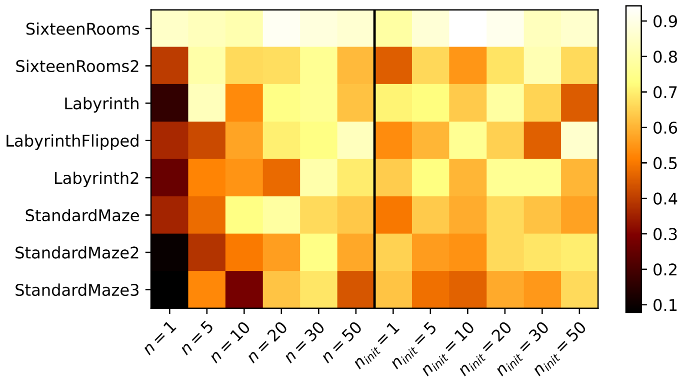


Figure 4: Heatmap showing solve rate of recurrent policy training using ACCEL on a set of holdout levels. (Left) uses standard ACCEL with fixed number of edits per mutation. (Right) ACCEL with the inclusion of mutation parameters in the evolutionary process. Each cell reports the mean and standard error of the learned policy on some level, measured over 5 random seeds. We observe that when allowing the mutation parameters to adapt learning is robust to selection of n_{init}.

We test on grid navigation task based on the MiniGrid environment [Chevalier-Boisvert et al., 2023], which requires an agent to navigate through partially observable levels to reach the goal. The agent is initially presented with levels devoid of obstacles (empty levels), with new levels being generated by mutating previously high-estimated regret members from the level buffer. New levels are generated by applying a finite number of random edits to the level; what edits to perform are selected uniformly at random, and include: adding an obstacle in a random position, removing an obstacle in a random position, and randomly move the goal position.

We associate each level θ in the population with an integer edit count n_θ. As such, each element of the level buffer is represented by a tuple of the form (θ, n_θ). On selection for mutation, a new child level θ' is generated by applying n_θ random edits to θ. $n_{\theta'}$ is generated by applying a uniformly sampled increment to n_θ, that is, $n_{\theta'} \sim \mathcal{U}\{n_\theta - \delta, n_\theta + \delta\}$ (following this sampling, $n_{\theta'}$ is clipped to always be greater than zero). When sampling new, empty levels, we initialize n_θ to some n_{init}. To make this setup self-referential, we set δ to be n_θ, making the number of edits its own mutation parameter.

In Figure 3b, we see that the inclusion of higher order mutation parameters are able to effectively overcome poor initial choice of n. Figure 3c shows that, when higher order mutation parameters are included in the evolutionary process, the mean n_θ in the level buffer naturally tends towards 20. Thus, higher order mutations enable ACCEL's ability to search level space to be robust to initial conditions.

Next, we test the effect improvements to robustness have on downstream performance. Ultimately, UED is concerned with improving learned agent performance on out-of-distribution tasks. We test

[1]In general, this quantity is unavailable. Thus, in practise, heuristics are used. See Appendix F for more details.

this by checking the performance on a set of holdout levels, illustrated in Appendix Figure 8. In comparison to fixed mutation parameters, we observe in Figure 4 that enabling mutation parameters to adapt drastically improves robustness to initial conditions on the holdout levels.

7.3 Future Directions: Opponent shaping

Since higher-orders of mutation seem to help with higher-orders of non-stationarity, we investigate the effect of higher-order mutation rates in multi-agent learning, and in particular, evolutionary opponent shaping. While PBT and ACCEL represent *non-stationary* optimisation settings, they are still *bounded* by the learning rate of the inner learning system. However, in competitive multi-agent settings, we can observe how higher and higher orders of meta-evolution results in ever increasing "orders" of non-stationarity.

In this setting, there are two opposing populations of agent parameters. At each generation the following happens:

1. All pairs of opposing agent parameters are evaluated head-to-head.
2. We select the top k members of both populations and copy them.
3. We mutate both populations according to their respective mutation operators.

We include more implementation details in Appendix G. We investigate higher-order self-adaptation in these settings in two, two-player zero-sum games: Regularized Matching Pennies and Kuhn Poker.

	Order 0	**Order 1**	**Order 2**
Order 0	-	-0.18 ± 0.02	-0.19 ± 0.02
Order 1	0.18 ± 0.02	-	-0.03 ± 0.02
Order 2	0.19 ± 0.02	0.03 ± 0.02	-

Table 1: Results for Regularized Matching Pennies. Standard error is calculated across 512 seeds.

Regularized Matching Pennies: Matching Pennies [Gibbons, 1992] is a simple two-player zero-sum game similar to rock-paper-scissors in which each player can play "Heads" or "Tails" at each time step. If the first player matches the second player's action, they receive a score of 1, otherwise they receive a score of -1. In this paper, we investigate the continuous setting in which each player outputs a *probability* of playing Heads. To prevent the agents from instantly reaching the nash equilibria, we *regularise* the policies and introduce an L2 penalty for being near 0.5. To prevent the logits from saturating, we pass the agent parameter through the *sin* function and use *additive* mutations. Thus, the agent's learning rate determines the rate at which they cycle between "Heads" and "Tails". Table 1 shows the results of different orders of meta-evolution in head-to-head competition. In Figure 5, we show that the first-order agent learns to anticipate the evolution of a zeroth-order agent online.

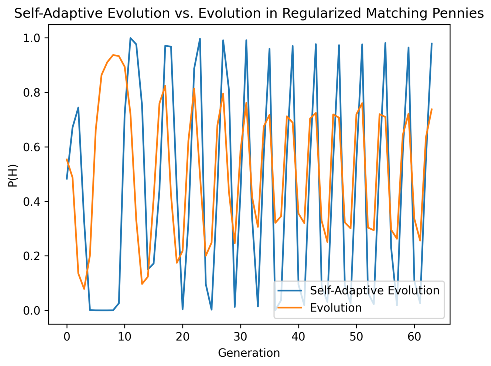


Figure 5: An analysis of the first 64 generations of a run of Regularized Matching Pennies of a first-order self-adaptive evolutionary agent (blue) against a zeroth-order evolutionary opponent (orange). y-axis corresponds to agent's probability of selecting heads. The first-order agent's objective is to match the output of the opponent, who is trying to do the opposite. Over the course of training, the self-adaptive agent learns the rate at which the zeroth-order opponent updates, eventually anticipating its evolution.

Kuhn Poker: Kuhn Poker [Kuhn, 1950] is a simplified version of poker in which there are only three playing cards. We use the implementation from Pgx [Koyamada et al., 2023]. Kuhn Poker has 16 possible states and 4 total actions, and we can thus represent each player's policy with a tabular representation. We show the results of higher-order meta-parameters in Table 2. In general, higher orders of learning allow the agents to anticipate their opponent's updates to exploit them.

	Order 0	**Order 1**	**Order 2**
Order 0	-	-0.21 $\pm$ 0.17	-0.56 $\pm$ 0.17
Order 1	0.21 $\pm$ 0.17	-	-0.31 $\pm$ 0.18
Order 2	0.56 $\pm$ 0.17	0.31 $\pm$ 0.18	-

Table 2: Results for Kuhn Poker. Standard error is calculated across 16 seeds.

8 Conclusion

We demonstrated that for nonstationary optimization problems, fitness of the population is sensitive to the choice of mutation operation: motivating the need for mutations themselves to adapt in anticipation of changes to the objective landscape. We showed that including higher order mutation parameters in the mutation process is sufficient to implicitly optimize for such cases, circumventing the need for expensive, multilevel optimization strategies. We have experimentally demonstrated that higher-order self-adaptive and self-referential mutations effectively overcome poor hyperparameter initialization in Population Based Training. Additionally, these mutations are effective at automatically tuning parameterized environment mutation operations for Unsupervised Environment Design in Reinforcement Learning.

Self-adaptive mutations are an important area of study for evolutionary dynamics. It is very clear that this happens in the natural world [Bäck et al., 1992], yet they are largely ineffective on our standard tasks and benchmarks [Kumar et al., 2022]. This paper answers this question by showing that these mutations are more effective in non-stationary, adversarial, and multi-agent tasks.

9 Future Work and Limitations

Future work could more directly investigate the impact of higher-order and self-referential adaptation in larger evolutionary [Chan, 2018] and multi-agent [Rutherford et al., 2023] systems. Specifically, studying the scalability of these adaptation mechanisms in more complex environments and over longer evolutionary timescales would provide deeper insights.

Our study has limitations that warrant consideration. For example, we do not investigate different mutation *strategies* and merely explore basic gaussian perturbations. Future work could investigate alternative strategies, such as crossover, and even discover ways we can evolve these more general mutation strategies as well.

References

Thomas Bäck et al. Self-adaptation in genetic algorithms. In *Proceedings of the first european conference on artificial life*, pages 263–271. MIT press Cambridge, 1992.

Colin F Camerer, Teck-Hua Ho, and Juin-Kuan Chong. A cognitive hierarchy model of games. *The Quarterly Journal of Economics*, 119(3):861–898, 2004.

Bert Wang-Chak Chan. Lenia-biology of artificial life. *arXiv preprint arXiv:1812.05433*, 2018.

Kartik Chandra, Audrey Xie, Jonathan Ragan-Kelley, and Erik Meijer. Gradient descent: The ultimate optimizer. *Advances in Neural Information Processing Systems*, 35:8214–8225, 2022.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo de Lazcano, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *CoRR*, abs/2306.13831, 2023.

Miguel A Costa-Gomes and Vincent P Crawford. Cognition and behavior in two-person guessing games: An experimental study. *American economic review*, 96(5):1737–1768, 2006.

Brandon Cui, Hengyuan Hu, Luis Pineda, and Jakob Foerster. K-level reasoning for zero-shot coordination in hanabi. *Advances in Neural Information Processing Systems*, 34:8215–8228, 2021.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in neural information processing systems*, 33:13049–13061, 2020.

Jakob N Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. *arXiv preprint arXiv:1709.04326*, 2017.

Kitty Fung, Qizhen Zhang, Chris Lu, Timon Willi, and Jakob Nicolaus Foerster. Analyzing the sample complexity of model-free opponent shaping. In *ICML Workshop on New Frontiers in Learning, Control, and Dynamical Systems*, 2023.

Robert S Gibbons. *Game theory for applied economists*. Princeton University Press, 1992.

Jonatan Gomez. Self adaptation of operator rates in evolutionary algorithms. In *Genetic and Evolutionary Computation Conference*, pages 1162–1173. Springer, 2004.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. A modern self-referential weight matrix that learns to modify itself. In *International Conference on Machine Learning*, pages 9660–9677. PMLR, 2022.

Matthew Thomas Jackson, Minqi Jiang, Jack Parker-Holder, Risto Vuorio, Chris Lu, Gregory Farquhar, Shimon Whiteson, and Jakob Nicolaus Foerster. Discovering general reinforcement learning algorithms with adversarial environment design. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023a. URL `https://openreview.net/forum?id=kAU6Cdq1gV`.

Matthew Thomas Jackson, Chris Lu, Louis Kirsch, Robert Tjarko Lange, Shimon Whiteson, and Jakob Nicolaus Foerster. Discovering temporally-aware reinforcement learning algorithms. In *Second Agent Learning in Open-Endedness Workshop*, 2023b.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Minqi Jiang, Michael Dennis, Jack Parker-Holder, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Replay-guided adversarial environment design. *Advances in Neural Information Processing Systems*, 34:1884–1897, 2021a.

Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In *International Conference on Machine Learning*, pages 4940–4950. PMLR, 2021b.

Minqi Jiang, Michael Dennis, Edward Grefenstette, and Tim Rocktäschel. minimax: Efficient baselines for autocurricula in jax. *arXiv preprint arXiv:2311.12716*, 2023.

Akbir Khan, Timon Willi, Newton Kwan, Andrea Tacchetti, Chris Lu, Edward Grefenstette, Tim Rocktäschel, and Jakob Foerster. Scaling opponent shaping to high dimensional games. *arXiv preprint arXiv:2312.12568*, 2023.

Louis Kirsch and Jürgen Schmidhuber. Eliminating meta optimization through self-referential meta learning. *arXiv preprint arXiv:2212.14392*, 2022.

Sotetsu Koyamada, Shinri Okano, Soichiro Nishimori, Yu Murata, Keigo Habara, Haruka Kita, and Shin Ishii. Pgx: Hardware-accelerated parallel game simulators for reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

Oliver Kramer. Evolutionary self-adaptation: a survey of operators and strategy parameters. *Evolutionary Intelligence*, 3:51–65, 2010.

Harold W Kuhn. A simplified two-person poker. *Contributions to the Theory of Games*, 1:97–103, 1950.

Akarsh Kumar, Bo Liu, Risto Miikkulainen, and Peter Stone. Effective mutation rate adaptation through group elite selection. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 721–729, 2022.

Robert Lange, Tom Schaul, Yutian Chen, Tom Zahavy, Valentin Dalibard, Chris Lu, Satinder Singh, and Sebastian Flennerhag. Discovering evolution strategies via meta-black-box optimization. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, pages 29–30, 2023.

Chris Lu, Jakub Kuba, Alistair Letcher, Luke Metz, Christian Schroeder de Witt, and Jakob Foerster. Discovered policy optimisation. *Advances in Neural Information Processing Systems*, 35:16455–16468, 2022a.

Chris Lu, Sebastian Towers, and Jakob Foerster. Arbitrary order meta-learning with simple population-based evolution. *arXiv preprint arXiv:2303.09478*, 2023.

Christopher Lu, Timon Willi, Christian A Schroeder De Witt, and Jakob Foerster. Model-free opponent shaping. In *International Conference on Machine Learning*, pages 14398–14411. PMLR, 2022b.

Luke Metz, C Daniel Freeman, Niru Maheswaranathan, and Jascha Sohl-Dickstein. Training learned optimizers with randomly initialized learned optimizers. *arXiv preprint arXiv:2101.07367*, 2021.

Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design. In *International Conference on Machine Learning*, pages 17473–17498. PMLR, 2022.

Günter Rudolph. Self-adaptive mutations may lead to premature convergence. *IEEE Transactions on Evolutionary Computation*, 5(4):410–414, 2001.

Alexander Rutherford, Benjamin Ellis, Matteo Gallici, Jonathan Cook, Andrei Lupu, Gardar Ingvarsson, Timon Willi, Akbir Khan, Christian Schroeder de Witt, Alexandra Souly, et al. Jaxmarl: Multi-agent rl environments in jax. *arXiv preprint arXiv:2311.10090*, 2023.

Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

Timon Willi, Alistair Hp Letcher, Johannes Treutlein, and Jakob Foerster. Cola: consistent learning with opponent-learning awareness. In *International Conference on Machine Learning*, pages 23804–23831. PMLR, 2022.

A Additional details on Experiment Settings

Experiments were performed on 8xNVIDIA A40's.

A.1 Population-based Training

Population-based Training (PBT) [Jaderberg et al., 2017] is an evolutionary approach to online hyperparameter tuning that involves training a population of models in parallel, each using its own set of hyperparameters. After a predefined number of learning steps the models are evaluated; the lowest-performing models then inherit the parameters and hyperparameters of the highest-performing members (*exploitation*) before mutating the hyperparameters they inherited (*exploration*). Model parameters θ are typically updated iteratively using stochastic gradient descent subject to these hyperparameters h, expressed as $\theta' \leftarrow \mathrm{step}(\theta \mid h)$. Each step may additionally include operations such as data collection, or rollouts in the case of reinforcement learning, and multiple gradient updates. Evaluation typically looks at model performance on a validation set, expressed as $\mathrm{eval}(\theta)$.

At its core, PBT is performing the following optimization problem in parallel to optimizing model parameters:

$$h^*(\theta) = \arg\max_{h \in \mathcal{H}} \left\{\mathrm{eval}(\mathrm{step}(\theta \mid h))\right\}, \tag{7}$$

where θ acts as the context. As such, PBT is performing non-stationary optimization.

A.2 Unsupervised Environment Design

One goal of UED is to learn a *minimax regret policy*, defined as

$$\pi_{minimax} \in \arg\min_{\pi} \left\{ \max_{\theta \in \Theta} \mathrm{Regret}_\theta(\pi) \right\}, \tag{8}$$

where $\mathrm{Regret}_\theta(\pi) = V^\theta(\pi^*) - V^\theta(\pi)$, $V^\theta(\pi) = \mathbb{E}_\pi[\sum_{t=0}^{T} r_t \gamma^t]$ is the expected discounted return when using policy π in level θ, π^* is the optimal policy.

Past work shows that such a policy is achieved at the Nash equilibrium of a game between a student agent and a teacher supplying the student with the current regret maximizing levels, which the student trains on [Dennis et al., 2020]. As such, the problem of learning a minimax regret policy reduces to solving the following optimization problem:

$$\theta^*(\pi) = \arg\max_{\theta \in \Theta} \left\{\mathrm{Regret}_\theta(\pi)\right\}, \tag{9}$$

which is non-stationary due to its dependence on the current policy π. Note that, in general, (9) is unavailable as the optimal policy is unknown during the course of training. Hence, in practise, the above objective is approximated heuristically [Dennis et al., 2020, Jiang et al., 2021b,a].

Prioritized Level Replay (PLR) [Jiang et al., 2021b] attempts to optimize (9) using random search. In PLR, training alternates between exploring the level space and exploiting previously discovered high-estimated regret levels. During exploration, new levels are sampled from an a priori level distribution. These new levels are then played by the currently learning agent, after which: their regret is estimated and, if the regret estimates exceed a threshold, are inserted into a rolling *level buffer*. During exploitation, levels are sampled from the level buffer using a distribution induced by each level's most recent regret estimate and time since the level was last played. These sampled levels are then used to update the model. The regret of a level θ is estimated using a trajectory τ sampled from the current learning agent π. Two common heuristics are:

$$\mathrm{pvl}(\tau, \pi; \theta) = \frac{1}{T} \sum_{t=1}^{T} \mathrm{relu}\left(\sum_{k=t}^{T} (\lambda\gamma)^{k-t} \delta_k \right), \tag{10}$$

$$\mathrm{MC}_{\mathrm{max}}(\tau, \pi; \theta) = \frac{1}{T} \sum_{t=1}^{T} \left(R_{\mathrm{max}} - V(s_t)\right), \tag{11}$$

where R_{max} is the largest cumulative return the learning agent has achieved on θ, δ_t is TD-error at timestep t, λ is the General Advantage Estimation constant, and $V(s_t)$ is the learning agent's estimated value of state s_t. We refer to these heuristics as *score* functions.

ACCEL [Parker-Holder et al., 2022] extends PLR by additionally exploring the level space using evolution, wherein the level buffer serves as the population. High-scoring levels are periodically mutated, and the resulting child-levels are attempted for insertion into the level buffer. Mutations are performed using a predefined operation.

A.3 Multi-Agent Learning and Opponent Shaping

Multi-Agent learning, especially non-cooperative multi-agent learning, introduces non-stationarity because player parameters can change throughout training. In many cases, this leads to convergence to sub-optimal solutions in general-sum learning. Learning with Opponent-Learning Awareness (LOLA) [Foerster et al., 2017] introduces the notion of *Opponent Shaping* (OS), which tries to account for the fact that other players are learning at the same time. While LOLA uses higher-order gradient-based methods to perform shaping, more recent works, such as Model-Free Opponent Shaping [Lu et al., 2022b, Khan et al., 2023], cast OS as a generic meta-learning problem because gradient-based methods require unrealistic access to opponent parameters. However, they use bi-level optimisation schemes to repeatedly train against a learning opponent. Performing higher-order opponent shaping in this setting quickly becomes computationally infeasible [Fung et al., 2023]. Ideally, we would be able to perform effective opponent shaping, without computationally intractable multi-level optimisation or unrealistic access to the opponents' gradients. In this paper, we investigate this by opponent shaping a population of evolutionary agents with varying levels of self-adaptation and self-reference.

PBT and ACCEL can be seen as multi-agent systems where one agent follows a standard learning algorithm, while the other employs evolutionary strategies to suggest new hyperparameters or levels. However, in these systems, higher-order learners are not directly competing against each other. In our multi-agent learning experiments, as the level of self-adaptation increases, the complexity and non-stationarity also continually increase. This contrasts with the PBT and ACCEL experiments, where the complexity reaches a limit due to the presence of a non-self-adaptive component.

B Proof of Theorem 1

Let each member of the population be given by $\mathbf{x} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \in \mathbb{R}^d$ with $\mathbf{x}_i \in \mathbb{R}^{d_i}$ and $\sum_{i=1}^n d_i = d$, and for mutation operations of the form:

$$\mathbf{x}_i^{t+1} \leftarrow \mathbf{x}_i^t + G_i(\mathbf{x}_{i+1}^t) + B_i$$

For $B_i \sim \mathcal{N}(0; \beta)$, for some predetermined β, $G_i : \mathbb{R}^{d_{i+1}} \to \mathbb{R}^{d_i}$ is some deterministic function. We denote the resulting member after applying all order mutations by $M_B : \mathbb{R}^d \to \mathbb{R}^d$, where B represents a noise vector.

For example, in [Lu et al., 2023] we have $d_i = 1$, G_i is the identity function, and $B_i \sim \mathcal{N}(0, \beta)$ for each i. For a sequence of t mutation vectors $\mathcal{B} = (B^1, \dots, B^t)$, we write $M_{\mathcal{B}}^t(\mathbf{x})$ or $M^t(\mathbf{x}, \mathcal{B})$ for the resulting individual.

Let P denote our population of solutions, where $|P| = N$. Suppose our objective is a vector-valued function of the form $f : \mathbb{R}^{d_0} \to \mathbb{R}$ **which depends only on $\mathbf{x}_0$**. Further, suppose we have a top-k algorithm $A : \mathbb{R}^{d \times N} \to \mathbb{R}^{d \times N}$ which takes in a population and outputs the next generation depending on f, as specified below. We write $A^t = A^t(P, \mathcal{C})$ for the population at time t, obtained by applying t times the algorithm A, to the initial population P, with a collection $\mathcal{C} = \{\mathcal{B}_i\}_{i=0}^{N-1}$ of mutation vectors for each individual. Similarly we write $A^t(\mathbf{x}, P, \mathcal{C})$ for the population descending from a specific individual $\mathbf{x} \in P$, so that $A^t(P, \mathcal{C}) = \cup_{\mathbf{x} \in P} A^t(\mathbf{x}, P, \mathcal{C})$.

We abuse notation and write $M_{\mathcal{C}}^t(\mathbf{x})$ to mean $M_{\mathcal{B}_i}^t(\mathbf{x})$, where i is the index corresponding to $\mathbf{x}$, whenever convenient. The top-k algorithm A is now formally specified as

$$A(\mathbf{x}, P, \mathcal{C}) = \begin{cases} \left\{ M_{\mathcal{C}}(\mathbf{x}) \mid j \in \left[\frac{N}{k}\right] \right\} & \text{if } \mathbf{x} \in \operatorname*{kmax}_{\phi \in P}\{f(M_{\mathcal{C}}(\phi))\} \\ \emptyset & \text{otherwise.} \end{cases}$$

The set $\mathrm{kmax}_{\phi \in P}\{f(M_{\mathcal{C}}(\phi))\}$ is defined to contain the k individuals ϕ with largest fitness $f(M_{\mathcal{C}}(\phi))$. Note also that we 'stop the clock' after selection of the k best individuals and their duplication into N individuals, but before mutation.

Finally, recall the definition of statewise dominance: $X \succ Y$ iff $\mathbb{P}(X \geq Y) = 1$ and $\mathbb{P}(X > Y) > 0$.

Theorem 1. Fix any $t < n$. Let $P = Q \cup \{\mathbf{x}\}$ and $\bar{P} = Q \cup \{\bar{\mathbf{x}}\}$ be initial populations such that $\mathbf{x}_i = \bar{\mathbf{x}}_i$ for all $i < t$, and let $\mathcal{B} = (B^1, \ldots, B^t)$ be a vector of random mutations up to time t. If the fitness of $\bar{\mathbf{x}}$ is statewise dominant over that of $\mathbf{x}$ after t mutations, that is,

$$f(M^t_{\mathcal{B}}(\bar{\mathbf{x}})) \succ f(M^t_{\mathcal{B}}(\mathbf{x}))\,,$$

then the number of descendants of $\bar{\mathbf{x}}$ is statewise dominant over that of $\mathbf{x}$ at time t,

$$\left|A^t(\bar{\mathbf{x}}, \bar{P}, \mathcal{C})\right| \succ \left|A^t(\mathbf{x}, P, \mathcal{C})\right|,$$

where $\mathcal{C} = \{\mathcal{B}_i \mid i \in [N]\}$. In particular, $\bar{\mathbf{x}}$ has a larger expected number of descendents than $\mathbf{x}$:

$$\mathbb{E}\left[\left|A^t(\bar{\mathbf{x}}, \bar{P}, \mathcal{C})\right|\right] > \mathbb{E}\left[\left|A^t(\mathbf{x}, P, \mathcal{C})\right|\right].$$

Proof. By assumption, there is a set $\Omega \subset \mathbb{R}^{t\times n}$ of measure 1 such that $f(M^t_b(\bar{\mathbf{x}})) \geq f(M^t_b(\mathbf{x}))$ for all $\mathcal{B} = b \in \Omega$. In particular, the set Ω^N also has measure 1 in $\mathbb{R}^{N\times t\times n}$, and for any $\mathcal{C} = c \in \Omega^N$,

$$f(M^t_c(\bar{\mathbf{x}})) \geq f(M^t_c(\mathbf{x}))\,. \tag{12}$$

For convenience, let $D, \bar{D} = A^{t-1}(\mathbf{x}, P, c), A^{t-1}(\bar{\mathbf{x}}, \bar{P}, c)$ for the descendents of $\mathbf{x}, \bar{\mathbf{x}}$ at time $t-1$, and $D^{\mathsf{c}}, \bar{D}^{\mathsf{c}}$ for the complement (individuals which are not descendents of $\mathbf{x}, \bar{\mathbf{x}}$ respectively). Since the fitness function depends only on θ_0, and by assumption that $\theta_i = \bar{\mathbf{x}}_i$ for all $i < t$, note that $|D| = \left|\bar{D}\right|$, and since other individuals are identical, $D^{\mathsf{c}} = \bar{D}^{\mathsf{c}}$. In particular, there is a bijection $h : D \cup D^{\mathsf{c}} \to \bar{D} \cup \bar{D}^{\mathsf{c}}$ satisfying, by Equation (12),

$$f(M_c(h(\phi))) = f(M^t_c(\bar{\mathbf{x}})) \geq f(M^t_c(\mathbf{x})) = f(M_c(\phi)) \tag{13}$$

for all $\phi \in D$, with equality for $\phi \in D^{\mathsf{c}}$. Now the number of descendents of $\mathbf{x}$ at time t, by definition of the top-k algorithm A, is given by N/k times the number of values $\{f(M_c(\phi))\}_{\phi\in D}$ which are among the largest k fitness values, namely,

$$\left|A^t(\mathbf{x}, P, c)\right| = \frac{N}{k} \times \left| D \cap \operatorname*{kmax}_{\phi \in D \cup D^{\mathsf{c}}} \{f(M_c(\phi))\}\right|.$$

Similarly,

$$\left|A^t(\bar{\mathbf{x}}, \bar{P}, c)\right| = \frac{N}{k} \times \left| D \cap \operatorname*{kmax}_{\phi \in D \cup D^{\mathsf{c}}} \{f(M_c(h(\phi)))\}\right|.$$

Let us now compare the two sets,

$$\begin{aligned}\{f(M_c(\phi))\} &= \{f(M_c(\phi)) \mid \phi \in D\} \cup \{f(M_c(\phi)) \mid \phi \in D^{\mathsf{c}}\}\\ \{f(M_c(h(\phi)))\} &= \{f(M_c(h(\phi))) \mid \phi \in D\} \cup \{f(M_c(h(\phi))) \mid \phi \in D^{\mathsf{c}}\}\,.\end{aligned}$$

By Equation (13), the RHS sets are identical, while the second LHS set contains values which are no smaller than their corresponding entry in the first LHS set. In particular, it can contain no fewer values which are among the k largest, and we thus obtain

$$\left|A^t(\bar{\mathbf{x}}, \bar{P}, c)\right| \geq \left|A^t(\mathbf{x}, P, c)\right|.$$

Since Ω^N is a set of measure 1, we conclude $\mathbb{P}(\left|A^t(\bar{\mathbf{x}}, \bar{P}, \mathcal{C})\right| \geq |A^t(\mathbf{x}, P, \mathcal{C})|) = 1$. It remains only to show that $\mathbb{P}(\left|A^t(\bar{\mathbf{x}}, \bar{P}, \mathcal{C})\right| > |A^t(\mathbf{x}, P, \mathcal{C})|) > 0$. By assumption of statewise dominance, there is a set $\Phi \subset \mathbb{R}^{t\times n}$ of non-zero measure such that $f(M^t_b(\bar{\mathbf{x}})) > f(M^t_b(\mathbf{x}))$ for all $b \in \Phi$. In particular, there is a compact subset $\Phi' \subset \Phi$ of non-zero measure and a real number $\delta > 0$ such that

$$f(M^t_b(\bar{\mathbf{x}})) > f(M^t_b(\mathbf{x})) + \delta \tag{14}$$

for all $b \in \Phi'$. Moreover, since $\mathcal{B}$ has full support in $\mathbb{R}^{t\times n}$, and f is continuous, the mutations $\mathcal{B}_\phi$ of individuals $\phi \in Q$ can be chosen such that $\mathbf{x}, \bar{\mathbf{x}}$ have a positive (and equal, as proven in the first half of the proof) number of descendents $|D| = \left|\bar{D}\right| > 0$ at time $t-1$. Similarly, mutations at time t can be chosen such that exactly k descendents in $D^{\mathsf{c}} = \bar{D}^{\mathsf{c}}$ have fitnesses satisfying

$$\max_{\psi\in D} M_c(\psi) < f(M_c(\phi)) < \max_{\psi \in D} M_c(\psi) + \delta \tag{15}$$

and all other descendents satisfying

$$f(M_c(\phi)) < \max_{\psi\in D} M_c(\psi)\,. \tag{16}$$

Formally, Φ' can be extended to a set $\Psi \subset \mathbb{R}^{N\times t\times n}$ of non-zero measure such that these equations hold for all $c \in \Psi$. In particular, by Equation (14),

$$f(M_c(\phi)) < \max_{\phi \in D} f(M_c(h(\phi)))$$

for all $\phi \in D$, and thus, combined with Equation (15), $|A^t(\mathbf{x}, P, c)| = 0$. On the other hand, combined with Equation (16), we have

$$\left|A^t(\bar{\mathbf{x}}, \bar{P}, c)\right| \geq \frac{N}{k} > 0 = \left|A^t(\mathbf{x}, \mathbb{P}, c)\right|.$$

Since Ψ is a set of non-zero measure, and this holds for all $c \in \Psi$, we obtain

$$\mathbb{P}(\left|A^t(\bar{\mathbf{x}}, \bar{P}, \mathcal{C})\right| > \left|A^t(\mathbf{x}, P, \mathcal{C})\right|) > 0\,.$$

Finally, we conclude $\left|A^t(\bar{\mathbf{x}}, \bar{P}, \mathcal{C})\right| \succ |A^t(\mathbf{x}, P, \mathcal{C})|$. It is now difficult not to prove that $\bar{\mathbf{x}}$ has a larger expected number of descendents:

$$\begin{aligned}\mathbb{E}\left[\left|\bar{A}^t\right| - \left|A^t\right|\right] &= \mathbb{E}\left[\left|\bar{A}^t\right| - \left|A^t\right| \mid \left|\bar{A}^t\right| \geq \left|A^t\right|\right] \\ &= \mathbb{E}\left[\left|\bar{A}^t\right| - \left|A^t\right| \mid \left|\bar{A}^t\right| > \left|A^t\right|\right] \mathbb{P}(\left|\bar{A}^t\right| > \left|A^t\right|) > 0\,.\end{aligned}$$

□

C Additional Results

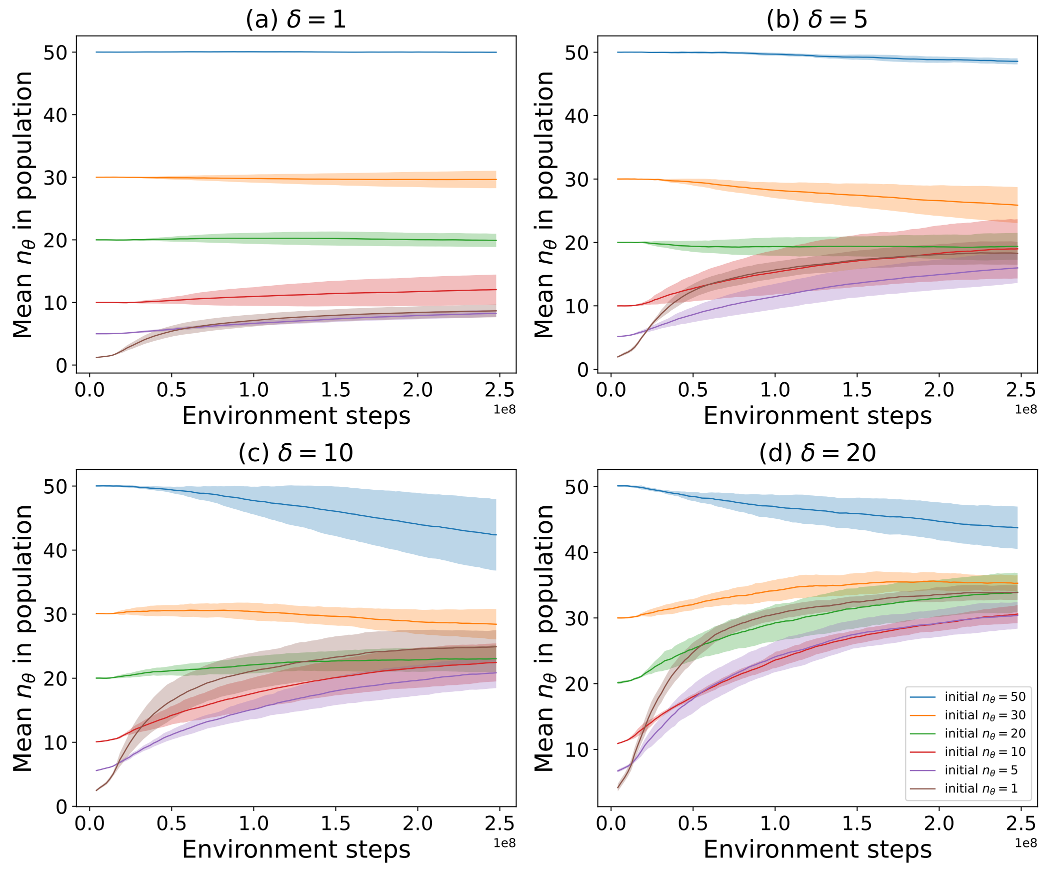


Figure 6: Mean performance over holdout levels displayed in Figure 8. Each plot corresponds to performance over different n_{init}, measured over 5 random seeds. Error bars correspond to 1 standard deviation. We see that allowing the number of edits to adapt enables ACCEL to achieve similar performance, regardless of initialization.

	$n_{\text{init}} = 1$	$n_{\text{init}} = 5$	$n_{\text{init}} = 10$	$n_{\text{init}} = 20$	$n_{\text{init}} = 30$	$n_{\text{init}} = 50$
$\delta_\theta = 1$	0.511 ± 0.021	0.600 ± 0.031	0.537 ± 0.032	0.654 ± 0.025	0.774 ± 0.025	0.708 ± 0.021
$\delta_\theta = 5$	0.506 ± 0.026	0.666 ± 0.019	0.543 ± 0.031	0.703 ± 0.024	0.682 ± 0.025	0.731 ± 0.021
$\delta_\theta = 10$	0.601 ± 0.033	0.565 ± 0.028	0.653 ± 0.022	0.646 ± 0.021	0.651 ± 0.024	0.778 ± 0.017
$\delta_\theta = 20$	0.706 ± 0.029	0.657 ± 0.027	0.702 ± 0.021	0.724 ± 0.024	0.682 ± 0.027	0.706 ± 0.013

Figure 7: Mean solve rate on maze navigation tasks for higher order self-referential mutations. In this setup, we extend each member of the population to include an additional δ_θ parameter corresponding to the maximum step size applied to $n\theta$. δ_θ is itself mutated by sampling from $\mathcal{U}(1, 2\delta_\theta)$. We observe improved performance robustness to poor initial mutation rates, however for small initial δs, performance improvements are less pronounced. Experiments were measured over 5 random seeds; standard error is reported.

D Visualizing Held-Out Minigrid Levels

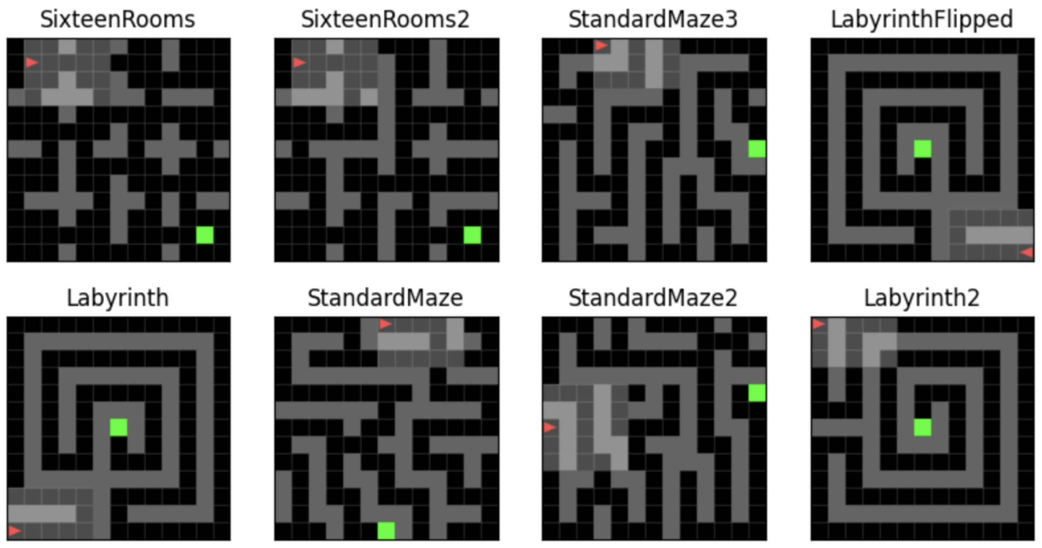


Figure 8: Set of 13×13 partially observable navigation environments. Red triangle represents the agent's current position and direction. At each timestep, the agent can only seen 5×5 tiles in front of them, and must choose whether to move forward 1 tile or rotate $90°$ (anti-)clockwise. The aim is to reach the goal represented by the green tile within a fixed budget of 250 timesteps.

E Details and Hyperparameters for PBT

We used the same convolutional architecture for MNIST and CIFAR. It consists of two convolutional layers and a dense layer.

Hyperparameter	Value
Conv 1 Kernel Size	(3, 3)
Conv 1 Strides	(2, 2)
Conv 1 Features	32
Conv 2 Kernel Size	(3, 3)
Conv 2 Strides	(2, 2)
Conv 2 Features	64
Dropout	0.5
Activation	ReLU
Max Pool Window	(2, 2)
Dense	10
Population Size	30
Top-K	6
Generations	500
Batch Size	32

Table 3: Hyperparameters for MNIST and CIFAR10

F Details and Hyperparameters for ACCEL

Hyperparameter	Value
Conv 1 Kernel Size	(3, 3)
Conv 1 Strides	(1, 1)
Dense Direction Embedding Size	5
LSTM Size	256
Actor Dense Layer Width	32
Critic Dense Layer Width	32
Learning Rate (lr)	1e-4
Max Grad Norm (max_grad_norm)	0.5
Number of Updates (num_updates)	30000
Number of Steps (num_steps)	256
Number of Train Environments (num_train_envs)	32
Number of Minibatches (num_minibatches)	1
Gamma (gamma)	0.995
Epoch PPO (epoch_ppo)	5
Clip Epsilon (clip_eps)	0.2
GAE Lambda (gae_lambda)	0.98
Entropy Coefficient (entropy_coeff)	1e-3
Critic Coefficient (critic_coeff)	0.5
Agent View Size (agent_view_size)	5
Initial Number of Walls (n_walls)	0
Evaluation Frequency (eval_freq)	250
Number of Evaluation Attempts (eval_num_attempts)	10
Evaluation Levels (eval_levels)	SixteenRooms, Labyrinth, StandardMaze
Score Function (score_function)	MaxMC
Exploratory Grad Updates (exploratory_grad_updates)	False
Level Buffer Capacity (level_buffer_capacity)	4000
Replay Probability (replay_prob)	0.8
Staleness Coefficient (staleness_coeff)	0.3
Temperature (temperature)	0.3
Minimum Fill Ratio (minimum_fill_ratio)	0.5

Table 4: Hyperparameters for ACCEL

	$n_{\text{init}} = 1$	$n_{\text{init}} = 5$	$n_{\text{init}} = 10$	$n_{\text{init}} = 20$	$n_{\text{init}} = 30$	$n_{\text{init}} = 50$
ACCEL	0.321 ± 0.106	0.597 ± 0.104	0.580 ± 0.107	0.685 ± 0.125	0.754 ± 0.092	0.660 ± 0.105
$\delta = 1$	0.625 ± 0.069	0.631 ± 0.069	0.615 ± 0.085	0.690 ± 0.060	0.742 ± 0.059	0.766 ± 0.056
$\delta = 5$	0.670 ± 0.068	0.671 ± 0.068	0.650 ± 0.073	0.620 ± 0.066	0.672 ± 0.052	0.652 ± 0.066
$\delta = 10$	0.725 ± 0.056	0.727 ± 0.056	0.602 ± 0.075	0.666 ± 0.053	0.728 ± 0.080	0.666 ± 0.052
$\delta = 20$	0.743 ± 0.057	0.755 ± 0.057	0.747 ± 0.051	0.720 ± 0.057	0.746 ± 0.054	0.726 ± 0.050
self-referential	0.613 ± 0.133	0.659 ± 0.107	0.636 ± 0.100	0.711 ± 0.073	0.672 ± 0.095	0.671 ± 0.088

Figure 9: Mean solve rate on maze navigation tasks. Each column corresponds to the number of edits that empty levels are initialized with. For ACCEL (first row), the number of edits is fixed throughout training. When the number of edits is small, performance on holdout levels suffers as ACCEL struggles to effectively search level space. Subsequent rows correspond to experiments where the number of edits are mutable (i.e. with self-adaptation). Rows 2-5 correspond to ACCEL experiments where the number of edits can adapt by at most a step size specified by δ. Self-referential corresponds to ACCEL experiments where the number of edits is its own mutation parameter. Performance is measured over 5 random seeds, where errors reported correspond to standard error.

G Details and Hyperparameters for Opponent Shaping

Hyperparameter	Value
Number of Generations	2048
Initial Std	0.1
Smoothing	0.9
Regularization	1.0
Population Size	1024
Top-K	512

Table 5: Hyperparameters for Regularized Matching Pennies.

Hyperparameter	Value
Number of Generations	1024
Initial Std	0.001
Smoothing	0.25
Number of Evaluations	3
Population Size	1024
Top-K	512

Table 6: Hyperparameters for Kuhn Poker.