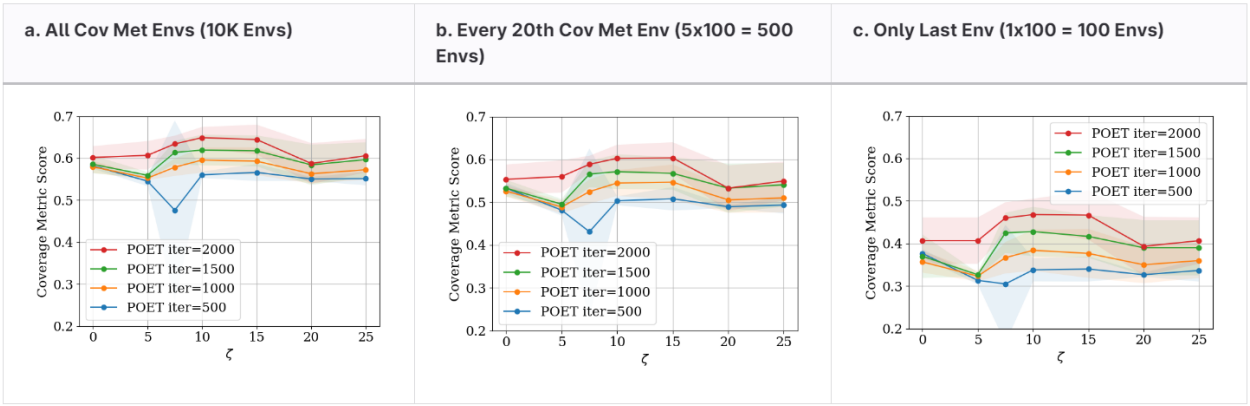
Contents

Appendix A Evaluation

A.1 Coverage Metric

Environment Set Selection



Supplemental Figure 1: Coverage metric scores vs ζ, the proportion of intrinsic reward added to the environment's extrinsic reward. From left to right, the covering set used for computing the coverage metric score is a.) the entire fixed set of $10,000$ environments, b.) a subset comprised of every 20^{th} environment in each freely evolved sequence, c.) a subset comprised of only the last environment in each freely evolved sequence.

The coverage metric requires a covering set of environments which represent a broader distribution of environments than are evolved in ePOET or Curious POET. A covering set must include both the distribution of evolved environments resulting from POET evolution, as well as distributions of environments rejected by the MCC filter, which includes those environments that are filtered out for being "too easy" or "too hard". I.e., we seek environments that span from easy (all agents solve), moderately difficult (some agents solve), and impossibly difficult, so that the coverage metric will be useful for comparing many different populations.

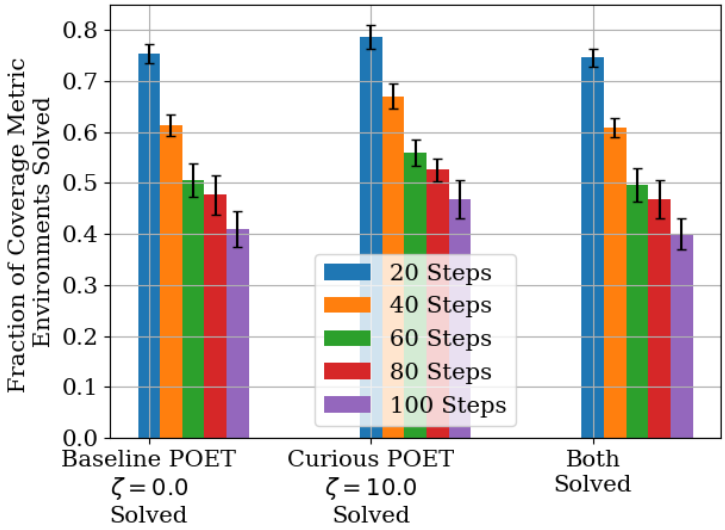
We argue that given a sufficiently large set of environments freely evolved via the CPPN environment evolution mechanism, the resulting distribution of environments includes those representative of the environments filtered by the MCC filter. Sampling of environments is organized as a set of free evolutionary sequences, or "threads" of length (in generations) significantly greater than the ePOET runs of interest in terms of ePOET iterations. In the bipedal walker domain, 100 steps often reaches an "impossible" level of difficulty as shown in Supplemental Figure 4. We restart this environment evolution sequence 100 times with unique seeds, and store the resulting set of $10,000$ environments as a fixed covering set.

However, as computation of the coverage metric score using the full covering set is rather compute expensive, we additionally select two subsets of the full set and recomputed coverage metric scores using these subsets. The first subset is comprised of the 20^{th}, 40^{th}, 60^{th}, 80^{th}, and 100^{th} environments from each free evolution sequence, resulting in 500 total environments. The second subset is comprised of only the last environment from each sequence, resulting in 100 environments. In practice, we find that even selecting the last environment in each sequence still yields a combination of easy, hard, and impossible environments. We propose using these 100 independent sequences of unconstrained environment evolution to approximate a covering set of the environments evolved in ePOET or Curious POET.

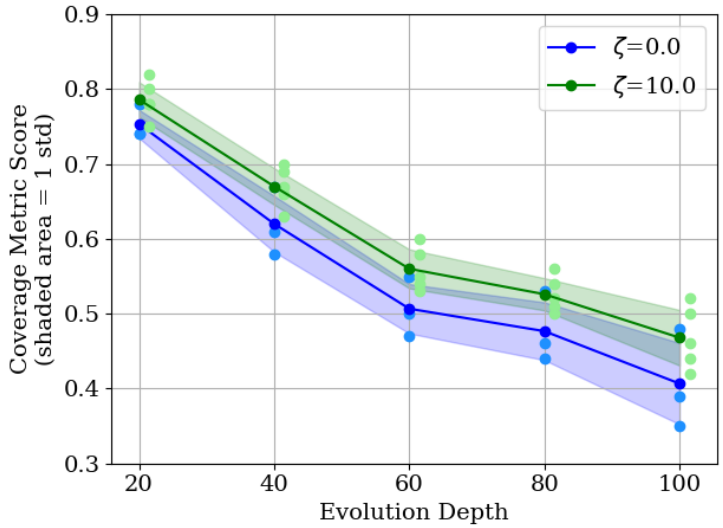
Coverage Metric Analysis

Supplemental Figure 1 shows that Coverage metric scores decline in columns b and c, due to the bias of these samples towards the end of the environment free evolution sequence where environments are more challenging. Notably, the overall score ordering with respect to ζ value and poet iterations is essentially the same in all three cases. Therefore, having determined that the entire fixed covering set of size $10,000$ is not needed for reasonable coverage metric behavior, in subsequent coverage metric analyses, we select the "Only Last Environment" sampling method as in column C in Supplemental Figure 1 and further reduce its size before utilization in the main experiment.

Supplemental Figure 2 indicates that thread length (evolution depth) does not seem to affect the relative solvability relationship between the environments solved by the baseline populations vs those solved by the curious populations.



Supplemental Figure 2: Coverage Metric Environment Solution Rate vs Sampling strategies. Note first the right column which shows how curious populations ($\zeta = 10$), seemingly independent of the coverage metric subset, also solved approximately 97% of the environments solved by the baseline ePOET populations. Further, in all cases, the curious populations went on to solve additional environments that were not solved by the baseline ePOET populations.

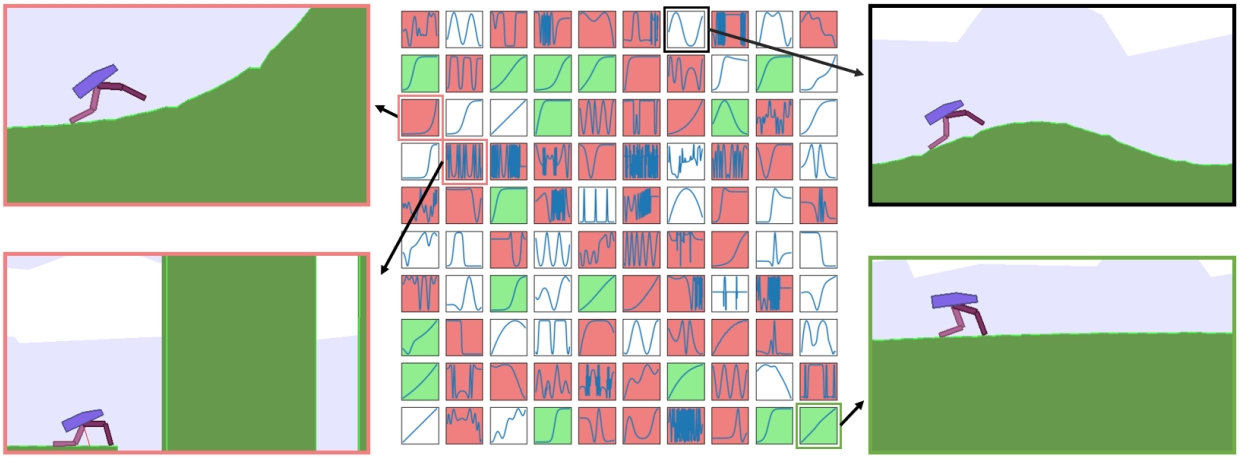


Supplemental Figure 3: Coverage Metric Scores vs Free Evolution Depth at POET iteration 2000, (points shifted for clarity)

Expanding our analysis of the coverage metric, we generate environment subsets composed of only the last step of free evolution, and vary the free evolution depth. We see in Supplemental Figure 3 that the coverage metric's ability to order populations by performance is stable over environment free evolution depth. With fewer free evolution steps, the environments are "easier" to solve, resulting in higher absolute scores, but the coverage metric's value as a tool to compare population performance remains intact, as the ordering of results remain essentially unchanged.

Final "Interesting" Environment Set Selection

Having developed a covering set of environments, we employ multiple criteria (as in MCC) to "down-select" our covering set which further reduces evaluation complexity, and increases the utility of the coverage metric by improving



Supplemental Figure 4: Plots of Bipedal walker terrain from the last free-evolution step in 100 independent environment evolution sequences. Bipedal walker terrains are horizontally and vertically normalized to fill the extent of plots and are color coded according to "interestingness," i.e., whether they are too easy or hard for the superset of agent populations $\mathcal{A}$ in the main experiment. Green backgrounds indicate environments that are too easy (solvable by $> 95\%$ of agents in $\mathcal{A}$), red backgrounds indicate environments that are too hard (solvable by $< 5\%$ of agents in $\mathcal{A}$), leaving the neutral background environments which are used in the main experiment as the coverage metric covering set.

its ability to differentiate agent populations. We define a set $\mathcal{A}$ as the superset of all agents evolved in the course of our experiments. We now filter out those environments that are either "too hard" or "too easy" with respect to agent superset $\mathcal{A}$ for being "un-interesting." Specifically, we define two thresholds $\rho_h = 0.05$ and $\rho_e = 0.95$, and filter out members of the "only last environments" covering set that are solved by fractions of $\mathcal{A}$ greater than ρ_e and less than ρ_h. Members of the "only last environments" covering set that are too easy are color coded as green in Supplemental Figure 4. Environments that are deemed too hard are color coded red. Finally, the remaining "interesting" environments that are not filtered out (with white background) represent our final covering set. We select these values of ρ_e and ρ_h as the smallest symmetric interval that excludes members of the "Only Last Environment" covering set for being both "too-easy" and "too-hard."

A.2 Cross Evaluation

Cross evaluation of evolved agents and environments was previously performed in Wang et al. (2020), but only between co-evolved agents and environments from one POET run. We perform comprehensive cross evaluation of agents and environments across multiple POET runs in order to evaluate the aggregate effect of intrinsic motivation on population evolution in terms of agent capability and environment difficulty.

Each location in the confusion matrix in Supplemental Figure 5 represents cross evaluation of two ePOET populations of sizes M and N consisting of sets of agent-environment pairs $\{(\theta_m^{\zeta_a}, E_m^{\zeta_a})\}_{1 \leq m \leq M}$ and $\{(\theta_n^{\zeta_e}, E_n^{\zeta_e})\}_{1 \leq n \leq N}$, with intrinsic motivation coefficients ζ_a and ζ_e for agent and environment respectively. Setting aside $E_m^{\zeta_a}$ and $\theta_n^{\zeta_e}$, we are left with a population Θ^{ζ_a} of M ePOET agents and a set $\mathcal{E}^{\zeta_e}$ of N environments to cross evaluate. For brevity, we refer to these sets as Θ and $\mathcal{E}$. We compute

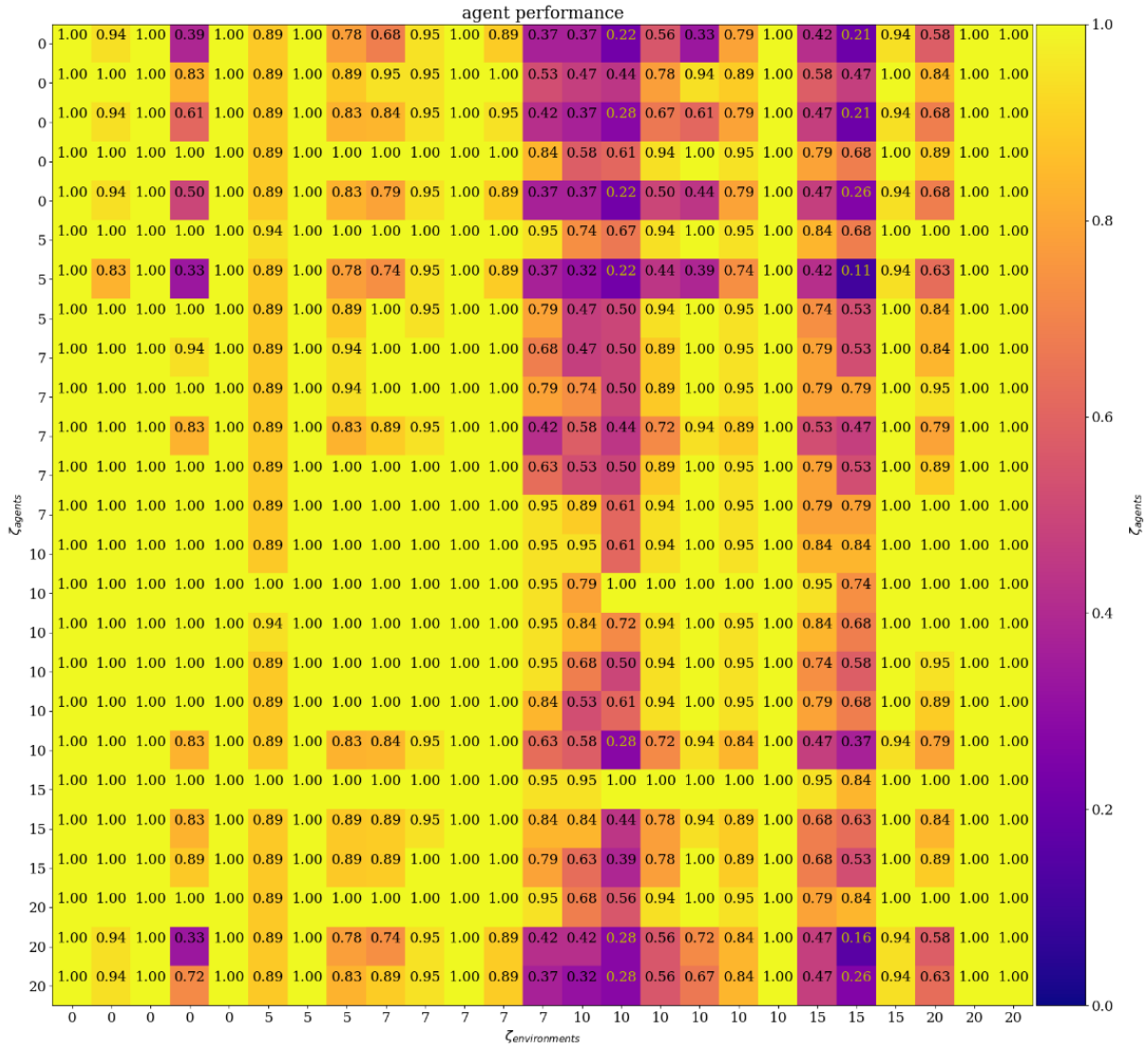
$$S_{\mathcal{E}(\Theta)} = \frac{|\{E \in \mathcal{E} \mid \exists \theta \in \Theta \text{ with } g_E(\theta) > \rho_{CE}\}|}{N}, \tag{1}$$

where

$$g_E(\theta) = \frac{\sum_{i=1}^{y} \begin{cases} 1 & f_E(\theta) > \rho_{solved} \\ 0 & \text{otherwise} \end{cases}}{y}, \tag{2}$$

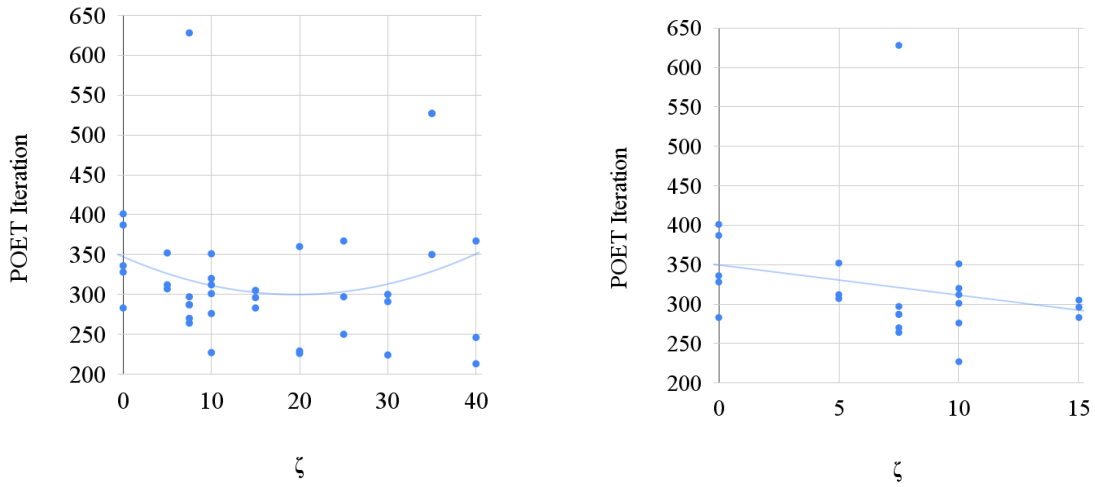
$|\cdot|$ denotes the size of the set, and $\rho_{CE} = 0.9$, and $\rho_{solved} = 230.0$, and $y = 10$ evaluation seeds.

As shown in Figure 3, we compute and plot the means and standard deviations over the sets of scores $S_{\mathcal{E}(\Theta)}$ represented by each pair of ζ_{agent} and $\zeta_{environment}$.



Supplemental Figure 5: Comprehensive cross evaluation metric of POET agent populations vs co-evolved environment populations, over our experiment's values of intrinsic motivation coefficient ζ. Indicated values are the fraction of environments solvable by any agent in at least 90% of evaluation seeds.

Appendix B Learning to Walk - a Learning Bottleneck or Reward Plateau



(a) Learning to Walk vs POET iterations

(b) Learning to Walk vs POET iterations (Inset)

Supplemental Figure 6: Number of ePOET iterations wherein the first ("flat") ePOET agent learns to solve the bipedal walker environment (with score $>= 230$) plotted across values of ζ, the intrinsic motivation coefficient. Increasing ζ, to a point, results in solving bipedal walker in fewer ePOET iterations. Performance suffers for $\zeta > 15$. This early stage of POET evolution, with the POET population comprised of only a single individual, is directly comparable to the ICM work in (Pathak et al., 2017), which showed that there is some optimal combination of intrinsic and extrinsic rewards that optimizes learning.

The design of the bipedal walker simulation environment seems to impose a structure in the learning progression of agents which typically proceeds through three phases: first an agent learns to stand up, i.e., not immediately fall over for which the agent receives a large penalty. Next, the agent learns to "jiggle" the walker legs in such a way as to slowly move forward and start earning extrinsic reward, which then provides a learning gradient resulting in continual refinement of the "jiggle" into a walking gait. The point immediately after learning to stand can be viewed as a learning bottleneck as there are many ways to "jiggle" the walker legs that don't result in horizontal translation, which is the source of reward gradient. Therefore, behavioral exploration is the only way to progress past standing up to a successful walking gait that succeeds in the bipedal walker environment.

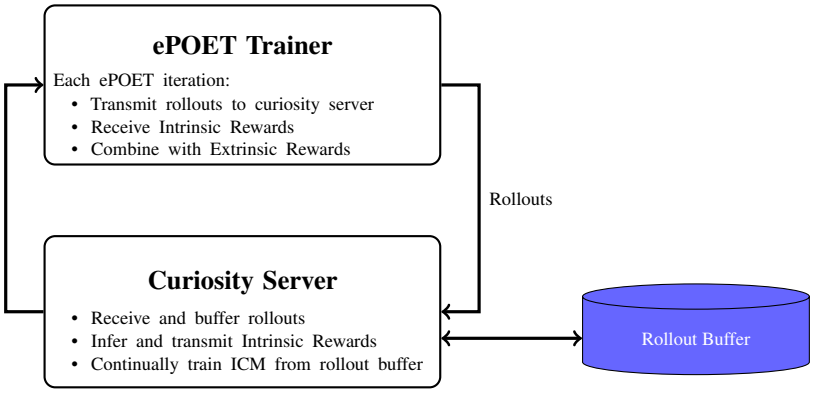
We measure the effect of intrinsic motivation on the first agent in a population in its paired "flat" environment. We count the number of POET iterations required for the first agent to learn to walk, indicated by a bipedal walker environment score of 230, a threshold previously discussed in (Wang et al., 2020). This analysis indicates that the curiosity oracle's intrinsic curiosity module is performing as expected from previous ICM works (Pathak et al., 2017; Burda et al., 2019). Supplemental Figure 6a and Supplemental Figure 6b show that increasing intrinsic motivation coefficient ζ results in learning to walk in fewer POET iterations on average. This is expected as single agents generally learn faster with some intrinsic motivation as previously identified in (Pathak et al., 2017).

Appendix C Curious Oracle

C.1 Curiosity Server

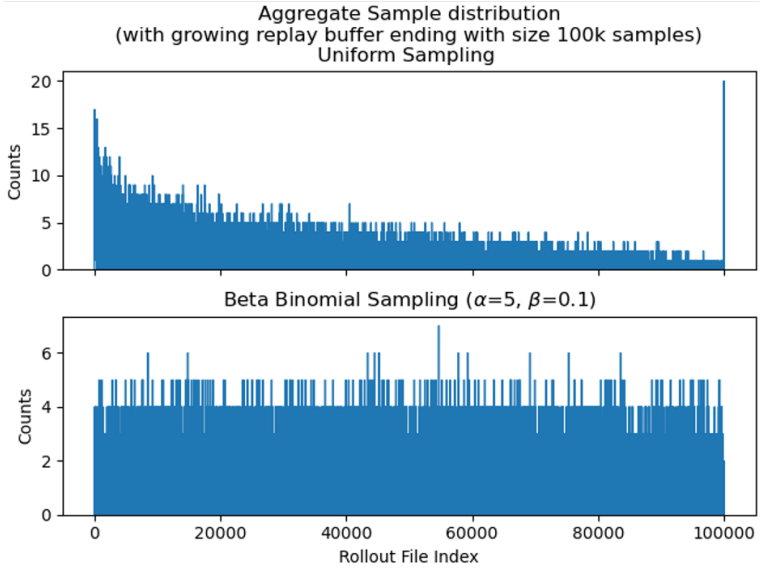
As seen in Supplemental Figure 7, the curiosity oracle is comprised of an Intrinsic Curiosity Module (Pathak et al., 2017) situated in a separate computational node as the POET trainer. The curiosity oracle receives complete rollouts from an ePOET trainer which it aggregates into a rollout buffer. They are stored in a collection of HDF5 files specific to a particular Curious POET run, each limited to 1GB in size. The rollout buffer ultimately will retain the entirety of the population's experience. The curiosity server alternately performs two main functions: 1.) Infer intrinsic rewards from incoming rollouts and return them to the ePOET trainer for inclusion in the agents' weight update. 2.) Continuously train the ICM module from the rollout buffer as in (Pathak et al., 2017).

The ePOET Trainer and Curiosity Server compute nodes share a file system and network subnet in our implementation.



Supplemental Figure 7: Our experiment distributes computation into specialized nodes. The ePOET trainer code runs on a 96 CPU core node, while the curious oracle code runs concurrently on "GPU node" with an Nvidia V100 used to train the ICM as well as run inference on rollouts for Curious POET. The ePOET Trainer sends batches of rollouts via ethernet to the Curiousity Server, whereupon, the ePOET Trainer is blocked until it receives an associated batch of intrinsic rewards. After receiveing the latest batch of rollouts, the Curiosity Server infers intrinsic rewards from the rollouts using the ICM and transmits the intrinsic rewards back to the ePOET Trainer where they are ζ proportionally combined with the extrinsic rewards from the bipedal walker environment. The Curiosity server alternates between running inference and training the ICM on rollout samples from the rollout buffer.

C.2 Curious Oracle Sampling



Supplemental Figure 8: In order to mitigate catastrophic forgetting of behaviors demonstrated in early ePOET iterations, we approximate a uniform sample distribution over a growing support of rollout files used for self-supervised training of the ICM. We sample from a Binomial Beta distribution (top), which biases sampling toward more recently written files. The resulting effective distribution (bottom) approximates a uniform distribution.

While training Curious POET, the ICM learns the behaviors of all agents as captured in rollouts which are continuously accumulated in a rollout buffer. This presents a sampling challenge, as naive uniform sampling on a growing collection results in a non-uniform sample distribution over the course of training as shown in Supplement Figure 8 (top). We utilize a beta binomial distribution to induce a recency bias in sampling, the effect of which is to approximate a uniform sampling distribution (bottom) over the growing rollout buffer.

C.3 Curiosity Server

The Curiosity Oracle, pictured in Supplemental Figure 7(bottom), is realized on a compute node with Nvidia V100 GPU (32GB), 50GB of system ram, and Linux operating system.

C.4 ICM Hyperparameters

ICM hyperparameters used in all experiments are recorded in Supplemental Table 1

Supplemental Table 1: ICM hyperparameters

Hyperparameter	Value
batch-size	100
num-frames	4
lr	0.0001
input-size	96
feature-size	20
hidden-size	512
output-size	4
β binomial α coef.	5
β binomial β coef.	0.1

C.5 ICM models

The neural network models utilized in the ICM are composed as follows:

feature-embedding:

Linear(input-size, hidden-size),
LeakyReLU(),
BatchNorm1d(num-features=hidden-size),
Linear(hidden-size, hidden-size),
LeakyReLU(),
BatchNorm1d(num-features=hidden-size),
Linear(hidden-size, feature-size),
BatchNorm1d(num-features=feature-size),

inverse-model:

Linear(feature-size * 2, hidden-size),
ReLU(),
Linear(hidden-size, hidden-size),
ReLU(),
Linear(hidden-size, output-size),

forward-model:

Linear(output-size + feature-size, hidden-size),
LeakyReLU(),
Linear(hidden-size, hidden-size),
LeakyReLU(),
Linear(hidden-size, feature-size)

Appendix D Bipedal Walker

To evaluate our approach, we utilize the enhanced bipedal walker environment as in Wang et al. (2020) where terrain height is parameterized by the output of a CPPN. Extrinsic reward $R^e(t)$ is computed at each timestep in the bipedal walker environment where

$$R^e(t) = \begin{cases} -100, \text{ if robot falls} \\ 130 \times \ \Delta x - 5 \times \Delta H_a - 0.00035 \times T_a, \text{ ot} \end{cases} \tag{3}$$

where H_a is the hull angle and T_a is applied torque. The bipedal walker simulation environment terminates due to encountering one of three terminating conditions:

1. The agent reaches the obstacle course finish line;
2. The agent falls over (realized as the agent's hull making contact with the ground as determined by the bipedal walker simulation);
3. The episode maximum length of 2000 time-steps is reached.

Appendix E ePOET Trainer Node

The ePOET Trainer, pictured in Supplemental Figure 7(top), is realized on a Linux compute node with 96 CPU cores and 400GB of system ram.

Appendix F ePOET Hyperparameters

Enhanced POET hyperparameters used in all runs are recorded in Supplemental Table 2.

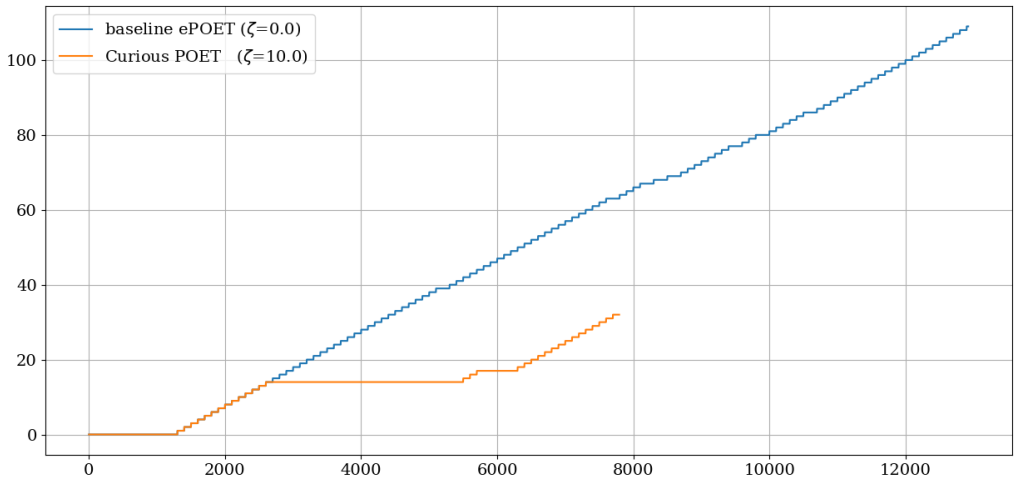
Supplemental Table 2: ePOET hyperparameters (Common to ePOET and Curious POET runs)

Hyperparameter	Symbol	Value
init		random
visualize-freq		50000
learning-rate	α	0.01
lr-decay		0.9999
lr-limit		0.001
batch-size		1
batches-per-chunk	S	512
eval-batch-size		1
eval-batches-per-step		1
noise-std	σ	0.1
noise-decay		0.999
noise-limit		0.01
normalize-grads-by-noise-std		True
returns-normalization		centered-ranks
max-num-envs		10
num-start-envs		1
start-envs-type		"randAgent"
adjust-interval		2
propose-with-adam		True
steps-before-transfer		50
mc-lower	ρ_h	100
mc-upper	ρ_e	340
num-workers		80
n-iterations		4000
icm-stable-iteration		200
intrinsic-reward-coefficient	ζ	0.0 (default for baseline)

Appendix G ANNECS

A recovering-stall phenomenon is observed in a minority of Curious POET runs with $\zeta > 7.5$, wherein population growth flattens out for significant periods of training time. Supplemental Figure 9 shows an example of this behavior in terms of the ANNECS measure, beside a normal ePOET baseline.

We hypothesize that there may be loss surface configurations such that our fixed intrinsic reward coefficient ζ is locally too high, resulting in decoupling of agent behavior exploration from extrinsic reward. This situation is then comparable to (Burda et al., 2019) where agents are trained in a variety of environments using only intrinsic motivation. Their results show that some environments are solvable and others are not. Our experience while testing our ICM implementation is that bipedal walker is not solvable using only intrinsic reward. It may be possible to better understand this effect by exploring the use of active or passive scheduling of the intrinsic motivation coefficient ζ.



Supplemental Figure 9: ANNECS vs Iterations. Note "stall" period followed by recovery and comparison to long baseline

Appendix H Reproducibility Checklist

This Paper:

- Includes a conceptual outline and/or pseudocode description of AI methods introduced (yes/partial/no/NA)
- Clearly delineates statements that are opinions, hypothesis, and speculation from objective facts and results (yes/no)
- Provides well marked pedagogical references for less-familiare readers to gain background necessary to replicate the paper (yes/no)

Does this paper make theoretical contributions? (yes/no)

Does this paper rely on one or more datasets? (yes/no)

Does this paper include computational experiments? (yes/no)

If yes, please complete the list below.

- Any code required for pre-processing data is included in the appendix. (yes/partial/no).
- All source code required for conducting and analyzing the experiments is included in a code appendix. (yes/partial/no) Note: The entire code-base used to realize this experiment will be released with the final paper in coming months, but our internal review process precludes release of code for this submission.
- All source code required for conducting and analyzing the experiments will be made publicly available upon publication of the paper with a license that allows free usage for research purposes. (yes/partial/no)
- All source code implementing new methods have comments detailing the implementation, with references to the paper where each step comes from (yes/partial/no)
- If an algorithm depends on randomness, then the method used for setting seeds is described in a way sufficient to allow replication of results. (yes/partial/no/NA)
- This paper specifies the computing infrastructure used for running experiments (hardware and software), including GPU/CPU models; amount of memory; operating system; names and versions of relevant software libraries and frameworks. (yes/partial/no)
- This paper formally describes evaluation metrics used and explains the motivation for choosing these metrics. (yes/partial/no)
- This paper states the number of algorithm runs used to compute each reported result. (yes/no)
- Analysis of experiments goes beyond single-dimensional summaries of performance (e.g., average; median) to include measures of variation, confidence, or other distributional information. (yes/no)
- The significance of any improvement or decrease in performance is judged using appropriate statistical tests (e.g., Wilcoxon signed-rank). (yes/partial/no)
- This paper lists all final (hyper-)parameters used for each model/algorithm in the paper's experiments. (yes/partial/no/NA)
- This paper states the number and range of values tried per (hyper-) parameter during development of the paper, along with the criterion used for selecting the final parameter setting. (yes/partial/no/NA)

References

Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros. Large-scale study of curiosity-driven learning. In *Seventh International Conference on Learning Representations*, Appleton, WI, 2019. ICLR.

D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, New York, NY, 2017. Association for Computing Machinery. URL `https://dl.acm.org/doi/10.5555/3305890.3305968`.

R. Wang, J. Lehman, A. Rawal, J. Zhi, Y. Li, J. Clune, and K. O. Stanley. Enhanced poet: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions, 2020.