OPEN-ENDED DREAMER: AN UNSUPERVISED DIVERSITY-ORIENTED NEUROSYMBOLIC LEARNER

Anonymous authors
Paper under double-blind review

ABSTRACT

Harnessing *program induction*, coupling robustness, and expressivity, by combining some form of symbolic and procedural knowledge appears to be a promising direction towards more open-ended innovation, and extrapolative behavior. Building upon DreamCoder framework (Ellis et al., 2021), we present an unsupervised diversity-oriented neurosymbolic learner: *Open-Ended Dreamer* (OED). Balancing environmental, language and novelty pressures, OED aims to learn novel, and useful programmatic abstractions. As a first test-bed we experiment with a tower building environment, where we analyze the benefits of library learning, neural guidance, innate priors, or environmental pressures to guide the formation of symbolic knowledge and open-ended program discovery.

1 INTRODUCTION

Open-ended refers to a process that produces increasingly diverse and complex artifacts. An archetypal inspiration is evolutionary processes which, through iterations blending variation and selection mechanisms, have brought a stunningly diverse array of lifeforms and behaviors on Earth. Most of these lifeforms exhibit a complex nested and modular structure, whose primitive components can be found across numerous other species.

Similarly, a significant part of human knowledge and skills exhibit some form of hierarchical and compositional structure, as we relentlessly invent more complex artifacts and behaviors from previous stepping stones —explicit or implicit, collective or individual. This ability to re-use, combine, transfer or scale previous discoveries is crucial to both human learning and adaptation.

Methods combining symbolic component with neural learning have shown great potential in grasping forms of compositional generalisation, while providing interpretable representations Chen et al. (2020); De Raedt et al. (2019). For instance, a neuro-symbolic model like DreamCoder (Ellis et al., 2021) has proven to be versatile in tackling multiple domains via program induction[1], from creative tasks as logo generation to more classic programming inductive tasks like physical laws regression.

This work extends DreamCoder towards open-endedness, making it less dependent on supervision. Our diversity-oriented neuro-symbolic model —*Open-Ended Dreamer* (OED)— balances *efficiency*[2] and *novelty* pressures in order to learn a compact and diverse set of reusable programs, organized in a hierarchical library. Our main contributions can be summarized as follows:

- (i) A diversity-oriented neurosymbolic learner, whose learning process (cf. Figure 1) includes: (i1) Niche-conditioned neurally-guided generation, with controllable stochasticity; (i2) Novelty-guided selection mechanism;(i3) Abstraction across niches of elites: dynamic library learning with diversity-oriented filtering and pruning.
- (ii) Experiments both on biasing the formation of symbolic knowledge, with innate priors, or environmental pressures (cf. Section 4,B.3) and on the novelty-efficiency trade-off, with phenomenon of convergent evolution (cf. Section A.7,B.4).**CCL Stochastic**

Our ablation experiments in a 2D tower building domain suggest that both the symbolic library resp. the neural guidance favor a *continued* creation of increasingly diverse and innovative programs

[1]Programs are an adaptive, robust and expressive way to encode both artifacts and behaviors, compelling for their logical, interpretative and extrapolative qualities.

[2]Here *efficiency pressure* encompass both *environmental* pressures, and a *language*-related pressure.

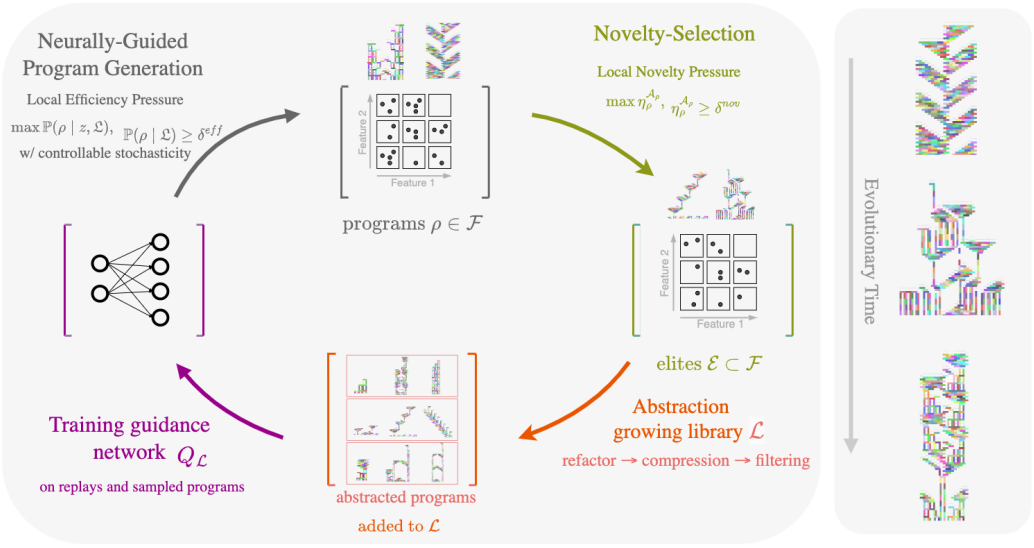

Figure 1: Open-Ended Dreamer follows an open-ended neuro-symbolic iterative process. (1) *Neurally Guided Generation*: programs are generated and distributed by niches in $\mathcal{F}$, by auto-regressively querying the neural model $\mathcal{Q}_\mathcal{L}$; (2) *Novelty-Guided Selection*: most novel programs are selected to form the elites $\mathcal{E} \subset \mathcal{F}$; (3) *Abstraction* across niches: through a EM algorithm with a Bayesian criteria, refactored programs are abstracted to grow a library $\mathfrak{L}$ of programmatic abstractions; (4) *Training* to consolidate the neural guidance on both replays and imagined programs, aiming to approximate a Bayesian posterior. *Right Side:* Illustration of programs generated through successive iterations.

(Section 4,B.2). The neural guidance is crucial in this discovery, and library learning improves the learning and support a more *sustained* generation, although considerably biasing the search space. Promisingly, the OED learner keeps finding novel solutions, although efficiency is favored over novelty in most of our experiments, as discussed Section A.7. Biasing the formation of symbolic knowledge through innate priors (e.g. bootstrapped library or metrics) also appears to help OED to create more diverse programs (Sections 4, B.2). Higher environmental pressures, seems to slow down this novelty search, and leads to more constrained, but also more consistent program discovery.

2 Related Works

This work touches upon *neuro-symbolic* models, *program synthesis*, and *library learning*. It is also rooted upon *open-endedness* considerations, as enacted in *novelty-search* and *quality-diversity* algorithms. We provide some related works in Section A.10, and only present in the main paper DreamCoder (Ellis et al., 2021), on which we built upon.

DreamCoder (Ellis et al., 2021) frames learning as program induction, and aims to jointly learn a hierarchical library of building blocks —called *primitives*, represented as programs—, and the intuition on how to compose them to solve a given task, via a neural network referred to as *recognition model*. DreamCoder takes place in a multi-task context, and leverages the variety of tasks to learn a library of useful units across these tasks. Programs may be either deterministic or probabilistic, and either act generatively (e.g. generating an artifact like a structure or image) or conditionally (e.g. input-output mapping), and are constructed upon a predefined *Domain Specific Language (DSL)*. DreamCoder follows a loose version of *wake-sleep algorithms* and a Bayesian framework: the library encodes a *prior* on program and the neural model aims to approximate a Bayesian posterior.

3 OPEN-ENDED DREAMER

As DreamCoder, OED consists of both a neural and a symbolic component bootstrapping each other: $\mathfrak{L}$, the *Generative Model*: a hierarchical symbolic library of concepts learning *useful* abstractions by capturing *regularities* across niches; $\mathfrak{Q}$, the *Recognition Model*: a neural model, trained to approximate the posterior distribution over elites programs given a niche, responsible for biasing the search space. As schematized in Figure 1, OED learns through an iterative process of *Neurally Guided Generation*, *Novelty-Guided Selection*, *Abstraction* and *Neural Training*, summarized below and detailed in Appendix A. A pseudo-Code is provided in Algorithm 1.

Supervision Relaxation By adopting a quality-diversity approach, OED relaxes DreamCoder's supervision needs by eliminating the requirement for handpicking tasks. Following MAP-Elites (Mouret & Clune, 2015), OED keep tracks of a multi-dimensional archive of phenotypic elites $\mathcal{A}$, cut into *niches* $\{\mathcal{A}_z\}$ along some predefined behavioral characteristics. Here, both the local competition and minimal criteria implemented blend *novelty* and *efficiency pressures*.

Bottom Up and Top Down generation Both a *top-down* generation following the generative model induced by the library $\mathfrak{L}$, and a *bottom-up* generation following the conditional model $\mathfrak{Q}_{\mathfrak{L}}$ takes place at different stages (cf. Algorithm 1). Programs ρ are first selected both from an efficiency-driven local competition on their posterior $(\max_{\rho \in z} \mathbb{P}(\rho \mid z, \mathfrak{L}) \propto \mathbb{P}(z \mid \rho)\mathbb{P}(\rho \mid \mathfrak{L}))$, given a niche z. To soften the efficiency pressure, we incorporated controllable stochasticity through sampling (instead of enumerating), as discussed Section A.7, A.3, with a minimal criteria on their prior $(\mathbb{P}(\rho \mid \mathfrak{L}) \geq \delta^{eff})$.

Novelty-Guided Selection The most novel programs are selected from previously generated programs, to form the elites $\mathcal{E}$, along two-steps of novelty-driven local competition ($\max_{\rho \in z} \eta_\rho^{\mathcal{A}}$ as defined in Equation 4) and novelty-driven minimal criteria ($\eta_\rho^{\mathcal{A}} \geq \delta^{nov}$). Section A.2 details the choice of the underlying metric space.

Abstraction The abstraction phase, first refactors the elites to find common program fragments, before abstracting these into new primitives[3], which are programs, of arity ≤ 3 in our experiments. It relies on the compression algorithm from Ellis et al. (2021) aiming to maximize a Bayesian criterion, through a generalised Expectation-Maximisation algorithm (Dempster et al. (1977)). To encourage a more evolving library, and soften the language-bias, we proposed a *stochastic pruning mechanism*, weighted by *innovation persistence* or *reach* defined Section A.8).

Neural intuition OED trains the recognition model to predict an approximate posterior over programs conditioned on the niche latent z, $\mathfrak{Q}(\rho \mid z) \approx \mathbb{P}(\rho \mid z, \mathfrak{L})$, both on *replays* from previous elites and *fantasies* –i.e programs sampled from the generative model. More details about the recognition model is detailed in Section A.6 and Figure 4.

4 EXPERIMENTS

We demonstrate our algorithm on the Tower Domain introduced in Ellis et al. (2021). Our approach relies on the same monadic, continuation-passing encoding of imperative programs. In this domain, the programs produce a sequence of actions, which encode the instruction of a hand moving over a $2D$ discretized canvas, which can drop horizontal or vertical blocks. The state of the hand encompasses both a $1D$ position and a boolean orientation. As behavioral characteristics, we use: center of gravity, width, and number of blocks of the tower generated from a program, and the *ecosystem minimal criteria* consists of lower and upper bounds to these values. Figures 10, 8, 5 show a selection of some of the programs produced, dreamed or abstracted.

Figure 6 provides a quantitative results of OED model (BASE run) comparatively with ablation experiments detailed in B.2. OED manages to populate all niches after only a few iterations (11), with a search times drastically decreasing after a few iterations, and a ratio of legit programs increasing. As illustrated in our ablation studies, OED leverages both neural and symbolic component and seems to steadily keep finding both *efficient* (cf. log Priors (e)) resp. *novel* solutions (cf. novelty scores (f)), with even a slight increase of diversity throughout evolution. Being capable of maintaining a steady novelty-score constitutes a promising sign of the open-endedness potential of the algorithm.

[3]This sequestration of information into reusable units –abstractions– is a form of *knowledge compression/distillation*.

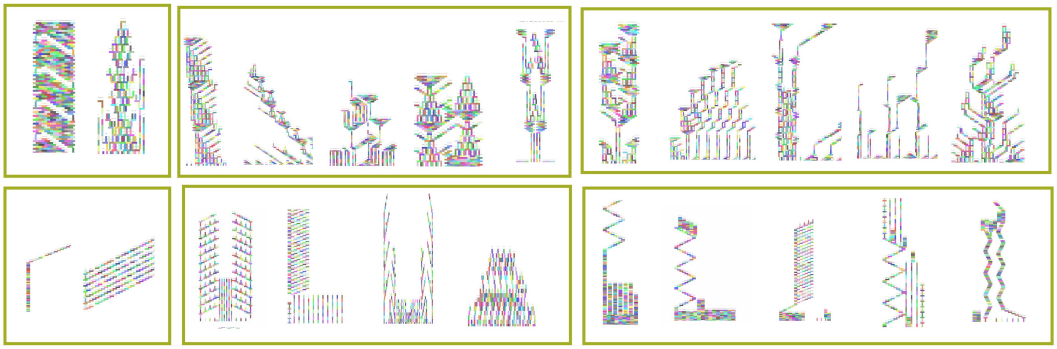

Figure 2: Extract of elites for INNATE (top) and BASE run (bottom), from early $(1, 2)$ mid $(4, 5)$ to late $(8, 9)$ iterations. OED produces progressively more complex artefacts.

It is important to note that the majority of experiments presented here are subjected to a strong efficiency (over novelty) pressure. However, as detailed Section A.7, the regulation of this balance via our hyperparameters could be investigated in future experiments.

Biasing Symbol Learning Since bias is a core component to direct symbol learning in social and natural systems, we examine different ways to bias evolution and guide the formation of symbolic knowledge, through either: (1) *innate knowledge*: the initial library is seeded with a few handpicked programmatic abstractions; (2) *stronger environmental pressure*: with a gravity-aware environment with $Box2d$ Physic Engine, or (3) *biased metrics*: guiding the novelty-search with a pre-trained CNN (default setting in most runs) instead of image-downsampling for the (RAW) Run. More details about these experiments can be found in Section B.3,7. Some main take-away: (1); Innate Priors help bootstrap the search at early stages, and the programs generated, dreamed or abstracted seems qualitatively more engaging and diverse (Figures 10, 8, 5). Yet it does not bring a strong quantitative advantage, which reflects a limited diversity metric and the bias ensuing from these priors;(2): the challenges brought by the physical constraints considerably slow down the program discovery, and the invented programs are more less diverse, yet consistent with the constraints (3): biasing the novelty-search through a more pertinent metric has, expectedly, a strong impact on the quality of the observed diversity.

5 Conclusion

While preliminary, our results suggest that neurally-guided library learning, enabling both more efficient program discovery and more sustained diversity is a promising framing towards open-endedness. By building a library of *programmatic abstractions* —instead of solutions—, our method preserves regularities in a more potent and versatile way than other diversity-oriented approaches. As we have seen, OED offers the possibility of guiding the formation of symbolic knowledge both with innate priors, biased metrics and environmental constraints. While putting the accent on efficiency pressures –both language-bias and environmental–, resulted in phenomenon of convergent evolution in our experiments, we expect that by loosening this pressure (cf. A.7) while allowing longer runs, we could witness more creative divergence. We leave this investigation to future work.

The simplicity of the Tower Domain task do not allow OED to reach its full potential for open-ended creation. Richer environments with stronger environmental coupling, and functional objective (such as in Minecraft Fan et al. (2022); Grbic et al. (2021)) could be key to further innovation. In future works, both the neural guidance, currently not tailored for diversity-enhancing, and the diversity-promoting algorithm could be refined towards an evolving and richer behavioral representation. A compelling development would be to use OED for the production of increasingly sophisticated behaviors, instead of simply artifacts.

REFERENCES

Xinyun Chen, Chen Liang, Adams Wei Yu, Dawn Song, and Denny Zhou. Compositional generalization via neural-symbolic stack machines. *Advances in Neural Information Processing Systems*, 33:1690–1701, 2020.

Luc De Raedt, Robin Manhaeve, Sebastijan Dumancic, Thomas Demeester, and Angelika Kimmig. Neuro-symbolic= neural+ logical+ probabilistic. In *NeSy'19@ IJCAI, the 14th International Workshop on Neural-Symbolic Learning and Reasoning*, 2019.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society: series B (methodological)*, 39(1): 1–22, 1977.

Jacob Devlin, Rudy R Bunel, Rishabh Singh, Matthew Hausknecht, and Pushmeet Kohli. Neural program meta-induction. *Advances in Neural Information Processing Systems*, 30, 2017a.

Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. Robustfill: Neural program learning under noisy i/o. In *International conference on machine learning*, pp. 990–998. PMLR, 2017b.

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B Tenenbaum. Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd acm sigplan international conference on programming language design and implementation*, pp. 835–850, 2021.

Kevin M Ellis, Lucas E Morales, Mathias Sablé-Meyer, Armando Solar Lezama, and Joshua B Tenenbaum. Library learning for neurally-guided bayesian program induction. 2018.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *arXiv preprint arXiv:2206.08853*, 2022.

Djordje Grbic, Rasmus Berg Palm, Elias Najarro, Claire Glanois, and Sebastian Risi. Evocraft: A new challenge for open-endedness. In *Applications of Evolutionary Computation: 24th International Conference, EvoApplications 2021, Held as Part of EvoStar 2021, Virtual Event, April 7–9, 2021, Proceedings 24*, pp. 325–340. Springer, 2021.

Rebecca M Henderson and Kim B Clark. Architectural innovation: The reconfiguration of existing product technologies and the failure of established firms. *Administrative science quarterly*, pp. 9–30, 1990.

Pascal Hitzler. Neuro-symbolic artificial intelligence: The state of the art. 2022.

Sreejan Kumar, Carlos G Correa, Ishita Dasgupta, Raja Marjieh, Michael Y Hu, Robert D Hawkins, Nathaniel D Daw, Jonathan D Cohen, Karthik Narasimhan, and Thomas L Griffiths. Using natural language and program abstractions to instill human inductive biases in machines. *arXiv preprint arXiv:2205.11558*, 2022.

Joel Lehman and Kenneth O Stanley. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pp. 211–218, 2011a.

Joel Lehman and Kenneth O Stanley. Novelty search and the problem with objectives. *Genetic programming theory and practice IX*, pp. 37–56, 2011b.

Percy Liang, Michael I Jordan, and Dan Klein. Learning programs: A hierarchical bayesian approach. In *ICML*, pp. 639–646. Citeseer, 2010.

Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. *arXiv preprint arXiv:1904.12584*, 2019.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. Innovation engines: Automated creativity and improved stochastic optimization via deep learning. In *Proceedings of the 2015 annual conference on genetic and evolutionary computation*, pp. 959–966, 2015.

Maxwell Nye, Armando Solar-Lezama, Josh Tenenbaum, and Brenden M Lake. Learning compositional rules via neural program synthesis. *Advances in Neural Information Processing Systems*, 33:10832–10842, 2020.

Justin K Pugh, Lisa B Soros, and Kenneth O Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, pp. 40, 2016.

Ronald E Rice and Everett M Rogers. Reinvention in the innovation process. *Knowledge*, 1(4): 499–514, 1980.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Tim Taylor, Mark Bedau, Alastair Channon, David Ackley, Wolfgang Banzhaf, Guillaume Beslon, Emily Dolson, Tom Froese, Simon Hickinbotham, Takashi Ikegami, et al. Open-ended evolution: Perspectives from the oee workshop in york. *Artificial life*, 22(3):408–423, 2016.

Lucas Tian, Kevin Ellis, Marta Kryven, and Josh Tenenbaum. Learning abstract structure for drawing by efficient motor program induction. *Advances in Neural Information Processing Systems*, 33:2686–2697, 2020.

A MODEL

A.1 PSEUDO-CODE

A simplified version of the learning loop is summarized in the pseudocode in Algorithm 1. Full code will be open-sourced on Github in the near-future.

A.2 METRIC SPACE

Both the *phenotypic distance* defined in Equation 1 and the *point cloud approximation* of the functional phenotypic distance defined Equation 2 are illustrated in Figure 3.

Phenotypic Distance To capture interesting variations between structures, we rely on a pre-trained visual network (VGG 16Simonyan & Zisserman (2014)). We expect that fine-tuning a visual feature extractor on the Tower Domain would make these high level features even more relevant to the data, and therefore the notion of distance more subtle and pertinent. It could be the object of future experiments. In our (RAW) experiment (cf. Section B.3), we intend to remove this bias, and simply (naively) rely on a downsampled version of the image as visual features.

To each program ρ, we can associate a black and white image I_ρ obtained after executing the program and rendering the block structure it corresponds to. The *phenotypic features* $f_\rho \in \mathbb{R}^d$ are computed by either downsampling the centered image to a fixed resolution of $r \times r$ (r being an hyperparameter; case of the (RAW) experiment only), or by computing its visual features using a pretrained CNN (most of experiments).

[1] At each iteration, niches are sampled following a mix of a random sampling and a novelty-biased sampling (where the sampling weights are the novelty score of the niche), with a ratio set by default at 0.5.

Algorithm 1: Open-Ended Dreams

Input: Initial library $\mathfrak{L}_0$, behavioral space cut into niches $\mathcal{Z}$.
Output: Library $\mathfrak{L}$, Recognition model $\mathfrak{Q}$, Archive $\mathcal{A}$, Elite $\mathcal{E}$
Hyperparameters: Batch size bs, enumeration timeouts t^{TD}, t^{BU}, frontier bound $M_{\mathcal{F}}$, efficiency threshold δ^{eff}, novelty threshold δ^{nov}, library novelty threshold δ^{lib}, elite bound $topK$, number iterations $numIterations$, stochastic weight parameter α_{sto}, training steps T.

for i **from** 1 **to** $numIterations$ **do**

/* Weighted-sampling of a set of niches of size bs[1] */
$B \sim \mathcal{Z}$

/* (TopDownGENERATION) from $\mathcal{L}$ of high posterior programs */
$\mathcal{F}^{TD} \leftarrow \mathfrak{L}.enumerate(M_{\mathcal{F}}, t^{TD})$, where $\mid \mathcal{F}_z^{TD} \mid \leq M_{\mathcal{F}}$

/* (TRAINING) Train Recognition Model $\mathfrak{Q}$ on replays and dreams */
$\mathfrak{Q}.train(\mathcal{R}, \mathcal{D}, T)$ where $\mathcal{R} = \{\rho \mid \rho \sim \mathcal{A}\}$ and $\mathcal{D} = \{\rho \mid \rho \sim \mathfrak{L}\}$

/* (BottomUpGENERATION) from the Recognition Model,
niche-conditioned */
$\forall z \in B, \quad \mathcal{F}_z^{BU} \leftarrow \mathfrak{Q}.enumerate(z, M_{\mathcal{F}}, t^{BU})$

if

$$\alpha_{sto} > 0$$

then

/* (Bottom Up Sampling from the recognition model $\mathfrak{Q}$: */
$\forall z \in B, \mathcal{F}_z^{BU} \leftarrow \mathcal{F}_z^{BU} \cup \mathfrak{Q}.sample(z, \alpha_{sto}, \delta^{eff}, M_{\mathcal{F}})$

end if

/* (NoveltySELECTION) Update Elites selecting most novel solutions
(local competition and minimal condition) */
$\forall z \in B, \quad \mathcal{E}_z \leftarrow topKNovelty(\mathcal{F}_z, topK, \delta^{nov})$ where $\mathcal{F}_z = \mathcal{F}_z^{TD} \cup \mathcal{F}_z^{BU}$

/* (ABSTRACTION) */
/* Stochastic Pruning, weighted by innovation activity */
$\mathfrak{L} \leftarrow \mathfrak{L}.prune(\delta^{lib})$
/* Refactor these programs, i.e. find equivalent programs. */
$\forall \rho \in \mathcal{E}, \mathfrak{R}_\rho \leftarrow refactor(\rho)$
/* Update Library $\mathfrak{L}$ with an EM algorithm to maximise: */
$\mathfrak{L} \leftarrow \arg\max \mathbb{P}(\mathfrak{L}) \prod_{z \in B} \sum_{\rho \in \mathcal{E}_z} \mathbb{P}(z \mid \rho) \max_{\tilde{\rho} \in \mathfrak{R}_\rho} \mathbb{P}(\tilde{\rho} \mid \mathfrak{L})$
/* Novelty Filtering of the new primitives */
$\mathfrak{L} \leftarrow \mathfrak{L}.\text{noveltyFilter}(\delta^{lib})$

end for

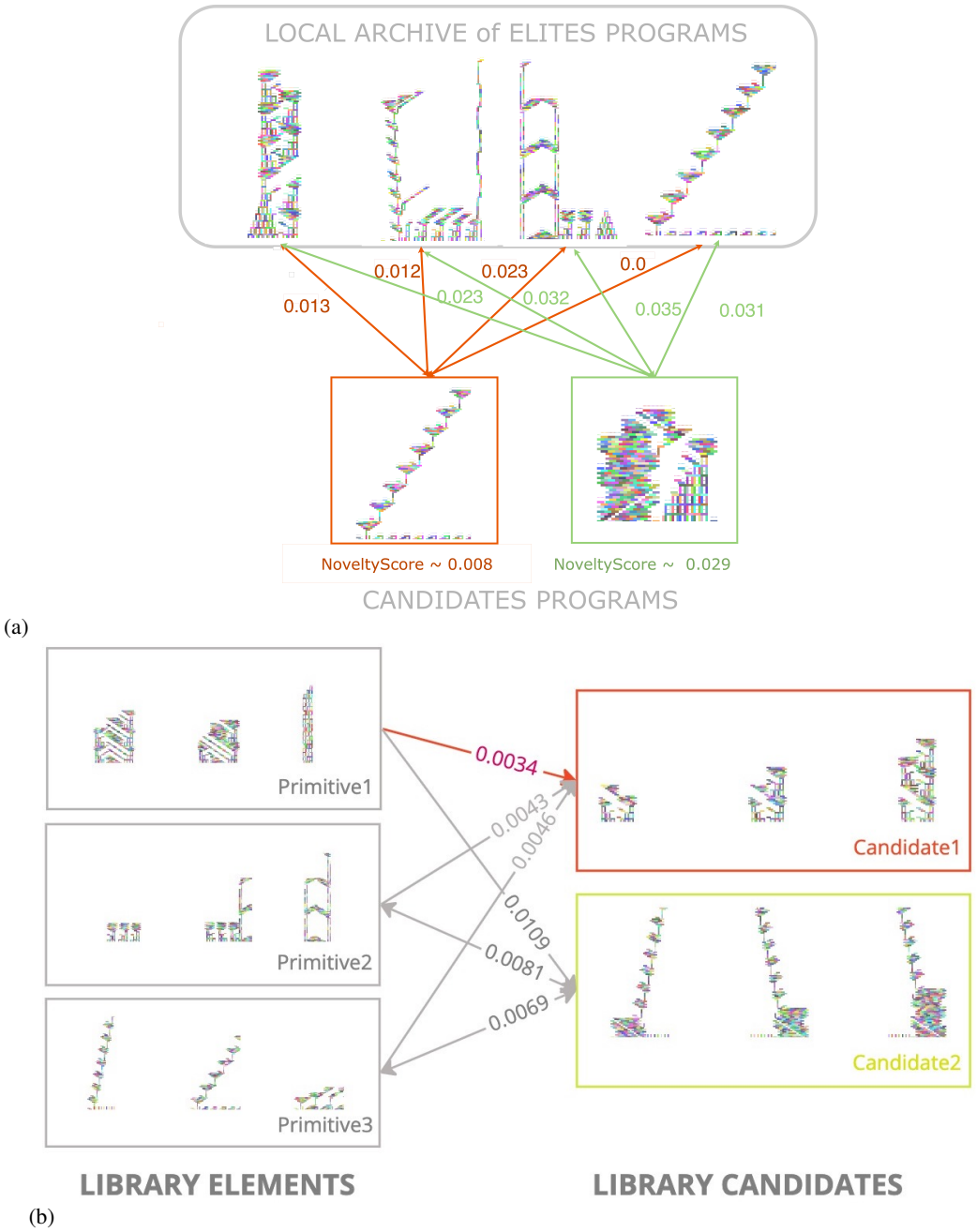

Figure 3: Examples of phenotypic distance between (a) programs (b) primitives, here relying on VGG 16 Simonyan & Zisserman (2014). (a) For each new candidate program generated, the novelty score is computed as the k-mean ($k = 3$) phenotypic distance elites programs in the archived (cf Equation 4). Top$K$($K = 4$) most novel solutions, which are above a threshold δ^{nov} ($= 0.015$ in our experiments) will be selected in the new Elites of this generation. (b) For each new primitive candidate to the library, we evaluate its distance (as defined Equation 2) to the other primitives already in the library $\mathfrak{L}$. If one of these distances is below a certain threshold (hyperparameter $\delta^{lib} \sim 0.35$ in our experiments), they are discarded. In this figure, red candidates are discarded because of the minimal novelty threshold required.

The *phenotypic distance* is then defined as the euclidean distance: [4]

$$\mathfrak{d}(\rho, \rho') \coloneqq \mathbf{d}(f_\rho, f_{\rho'}) \tag{1}$$

Point-based approximation for functional distance To evaluate the *functional distance* between primitives (functions of arity ≤ 3 in our experiments), we propose a point-based approximation. First, we sample for each primitive a set of $m_{\mathfrak{p}} \leq 20$ programs stemming from the primitive instantiation on arguments $\in \{0, 9\}$ referred to as a stochastic cloud of $\mathfrak{p}$, $\hat{C}_{\mathfrak{p}}$. The approximate functional distance between primitives is then defined as the average over $\mathfrak{p}$-clouds programs of the (min) distance to $\mathfrak{p}'$ cloud.

$$\hat{\mathfrak{d}(\mathfrak{p}, \mathfrak{p}')} \coloneqq \frac{1}{m_{\mathfrak{p}}} \sum_{\rho \in \hat{C}_{\mathfrak{p}}} \left(\min_{\rho' \in \hat{C}_{\mathfrak{p}'}} (\mathfrak{d}(\rho, \rho') \right). \tag{2}$$

where $\mathfrak{d}(\rho, \rho')$ is defined by Equation 1. In the abstraction phase, we incorporate a filtering mechanism, filtering out candidate primitive which are too close (given a threshold δ^{lib}) from primitives already in the library $\mathfrak{L}$, which relies on this distance.

A.3 Bottom Up and Top Down generation

Both, a *top-down* enumeration following the generative model induced by the library $\mathfrak{L}$, and a *bottom-up neurally-guided* enumeration following the conditional model $\mathfrak{Q}_{\mathfrak{L}}$, take place at different stages. In both case, we aim first to populate the sampled niches with the most efficient programs, selected according to their posterior:

$$\mathbb{P}(\rho \mid z, \mathfrak{L}) \propto \mathbb{P}(z \mid \rho)\mathbb{P}(\rho \mid \mathfrak{L}), \tag{3}$$

where z denotes a sampled niche feature, ρ a program, $\mathfrak{L}$ the library. The likelihood $\mathbb{P}(z \mid \rho)$ evaluate how a program ρ fit in a given niche (including some minimal criteria), and the prior reflects $\mathbb{P}(\rho \mid \mathfrak{L})$ how they fit in the library, i.e. how likely they are under the generative model. In these enumerations, programs are roughly enumerated in decreasing prior probability order,[5] and distributed between bounded frontiers $\mathcal{F}^{TD}$, resp. $\mathcal{F}^{BU}$.

Looking at programs with the higher posterior in each niche enacts a *efficiency-driven local competition* on the posterior.

Leveraging Stochasticity To lower the efficiency pressure (as discussed Section A.7), in contrast to DreamCoder, OED incorporate a *Bottom Up sampling* from the recognition model $\mathfrak{Q}$. This Bottom Up sampling includes an efficiency-guided minimal bound on the prior of the programs generated ($\mathbb{P}(\rho \mid \mathfrak{L}) \geq \delta^{eff}$, yet more loose than in the previous enumerations). By adjusting the number of sample, we can balance the stochasticity level[6], since the Bottom Up Frontiers obtained both through enumeration and sampling are merged before the novelty selection.

A.4 Novelty-Guided Selection

After merging top-down $\mathcal{F}^{TD}$ and bottom-up frontiers $\mathcal{F}^{BU}$, the most novel programs are selected, to form the elites $\mathcal{E}$ of the current iteration, coupling *local competition* and *minimal criteria*:

(i) **Local competition**: Selection of the top-k most novels programs per niche. A *local novelty score* is computed for each program ρ in a niche z as the K-mean of the *phenotypic distance* to the archive elements of the same niche $\mathcal{A}_z$:

$$\eta_\rho^{\mathcal{A}} \coloneqq \frac{1}{K} \sum_{\rho' \in \mathcal{N}_{z,\rho}} \mathfrak{d}(\rho, \rho'), \tag{4}$$

[4] We normalize the phenotypic features to ensure they are in the interval $[0, 1]$ (through a softmax) and that the distance between two programs similarly lies in the interval $[0, 1]$.

[5] More specificity about this enumeration procedure, which mixes best-first search with depth-first may be found in Ellis et al. (2021), Algorithm 3.

[6] Additional, more minor, sampling mechanisms are incorporated in the Bottom Up generation (sampling from enumerated programs), or in the compression frontiers sent to the compressor (sampling from elites for the procedure to be tractable), or in the stochastic pruning mechanism of the library mentioned below.

where $\mathfrak{d}$ is the *phenotypic distance* defined in Section A.2) and $\mathcal{N}_{z,\rho}$ is defined as the local neighborhood of ρ in $\mathcal{A}_z$, i.e. the set of the K closer neighbors from ρ in a same niche z. Typically, K is set to 3 or 1 in our experiments. Top-K programs are selected one-by-one for the sampled niche, and at each step, the novelty score of each candidate is re-evaluated to adapt to the growing niche.

(ii) **Minimal Criteria**: from these selected programs, we retain only the programs whose novelty score is above a fixed novelty-threshold $\eta_\rho^{\mathcal{A}} \geq \delta^{nov}$.

A.5 ABSTRACTION

The abstraction phase aims to grow the library from useful and novel building blocks found across niches. It relies on DreamCoder self-supervised compression algorithm (detailed in Ellis et al. (2021), Appendix $S4.5$), which first refactors the elites in order to find common program fragments, before abstracting these fragments out into new functions, while aiming to maximize a Bayesian criterion. These abstracted programs –referred to as primitives– are functions of arity maximum 3 in our experiments.

The generative model $\mathfrak{L}$ consists of a set of primitives $\mathcal{P}$ (λ-calculus expressions) and a weight parameters θ (dimension $\#\mathfrak{L} + 1$), which are co-optimized following a form of generalised Expectation-Maximisation algorithm (Dempster et al., 1977). The consolidation steps aims to maximise a Bayesian objective: [7]

$$\mathbb{L}(\mathcal{P}, \theta, \mathcal{F}) = P[\mathfrak{L}] \prod_z \sum_{\rho \in \mathcal{F}_z^{COMP}} \max_{\tilde{\rho}} P[z \mid \rho] P[\rho \mid \mathfrak{L}], \tag{5}$$

where the max is taken over all the possible refactoring $\tilde{\rho}$ of a program ρ, and $\mathcal{F}^{COMP}$ is a bounded subset of elites programs sent to the compressor at one iteration.[8]

More precisely, each compression step includes a step of estimation of the weight vector $\theta^*(\mathcal{P})$[9], and a step of maximisation of the following library score over new candidate primitives $\mathfrak{p}'$:

$$\max_{\mathfrak{p}'} \texttt{score}(\mathcal{P} \cup \{\mathfrak{p}'\}) \quad \text{where} \quad \begin{cases} \texttt{score}(\mathcal{P}') & \coloneqq & \log \mathbb{P}[\mathcal{P}'] + \log L(\mathcal{P}', \theta^*(\mathcal{P})) - \|\theta^*(\mathcal{P})\|_0 \\ L(\mathcal{P}', \theta) & \coloneqq & \prod_z \sum_{\rho \in \mathcal{F}_z^{COMP}} \max_{\tilde{\rho}} P[z \mid \rho] P[\rho \mid \mathcal{P}', \theta] \\ \mathbb{P}[\mathcal{P}] & \propto & \exp\left(\lambda \sum_{p \in \mathfrak{L}} size(p)\right) \end{cases} \tag{6}$$

It includes a prior penalizing the syntactic complexity of the λ-calculus expressions in the library, to avoid unnecessary growth. This co-optimisation loop goes on until no further increase in the grammar score is possible, or after reaching a maximum of compression steps.

To enable further efficiency resp. adaptivity, we propose two additional mechanisms:

1. *a novelty-guided library minimal criteria*: new primitive $\mathfrak{p}'$ shall be distant enough from existing primitives in the library $\mathfrak{L}$:
$$\max_{\mathfrak{p} \in \mathfrak{L}} \mathfrak{d}(\mathfrak{p}, \mathfrak{p}') \geq \delta^{lib}$$
To evaluate such functional distance, we rely on the phenotypic metric space an a point-based approximation defined in Section A.2.
2. *stochastic pruning mechanism*: every iteration, we sample $\leq \beta$ primitives to prune (β being an hyperparameter, e.g. 2 in our experiments). This sampling is weighted by the *activity score* of each primitive at the current iteration which represents its frequency among the elites, as defined in Equation 7. Note that the *innovation reach score*, defined in Equation 8, could be another candidate to weight the pruning.

A.6 TRAINING THE NEURAL INTUITION

The recognition model, illustrated in Figure 4, aims to learn how to compose the programmatic abstraction to create novel programs. It plays the crucial role of biasing the search space, by drasti-

[7] $\mathbb{L}$ can be seen as a particle approximation marginalized over a finite set of programs.

[8] For this procedure to be tractable, the size of this set shall be constrained.

[9] The tractable MAP estimator adopted is detailed in Ellis et al. (2021), Appendix $S4.5.4$.

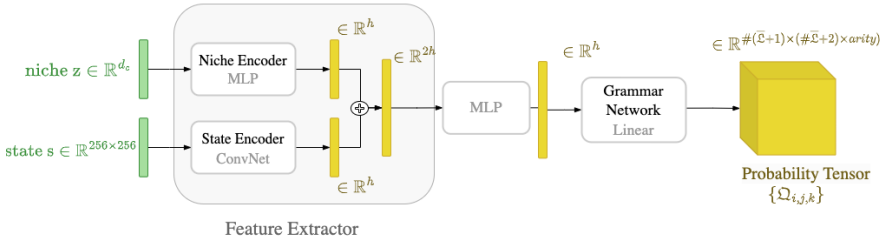

Figure 4: Architecture of the Recognition Model. It encompasses a feature extractor and a contextual Grammar Network and outputs a conditional probability matrix depending of the number of library elements and the maximum arity allowed.

cally reducing the breadth of search in the combinatorially exploding space of programs. It encodes a distribution over the elements in the library conditioned on the local context: taking as input a latent niche vector z, and the current state s passed as an image (e.g. image of the current tower being constructed), it outputs a probability tensor encoding the probability of a next token being j given the previous token in the growing language. Querying the recognition model auto-regressively produces therefore a program.

In the Dream phase, Dream Coder (Ellis et al. (2021), Section $S4.6$) trains the recognition model to predict an approximate posterior over programs conditioned on the niche latent z, $\mathfrak{Q}(\rho \mid z) \approx \mathbb{P}(\rho \mid z, \mathfrak{L})$. In OED, the novelty selection makes this bayesian objective more loose, since elites are selected as a subset of the highest posterior solutions found (non stochastic case). The neural network is trained both on *replays* from elites from previous iterations and *fantasies* –i.e programs sampled from the generative model–, with a dream ratio by default set at 0.5.

A.7 Efficiency-Novelty Trade-Off

At the core of OED search for diversity-enabling programmatic abstraction –and at the core of open-endedness– is a delicate balance between *efficiency* resp. *novelty*: for instance, most novel solutions compared to the archive commonly stands out as less likely given the generative model, and therefore could be discarded in the first efficiency-driven generation step. It is important to note that the *efficiency pressure* we refer to encompasses both (i) an *environmental pressure*, notably enclosed within the likelihood $\mathbb{P}(z \mid \rho)$ term in the posterior[10], and (ii) a *language-pressure* stemming from the learned generative model $\mathfrak{L}$, notably enacted by the prior $\mathbb{P}(\rho \mid \mathfrak{L})$ term.

Our model provide ways to balance efficiency versus novelty pressure. For instance, to loosen the efficiency pressure, we may increase the enumeration times, increase the first bound on each frontier $M_{\mathcal{F}}$ regulating the efficiency local competition, lower the efficiency threshold δ^{eff}, increase the Helmholtz sampling in the dream phase, lower the structure penalty (penalising longer primitives), but also, increase the proportion of sampling allowed within the generation, as explained A.3, A.9. Loosening up the efficiency pressure in favor of the novelty pressure would result in more diversity yet less regularity, and more arduous abstraction as early experiment suggests.

A.8 Innovation Metrics

'Innovation' is a relatively broad term [11] involving the production of 'novel' ideas or artifacts which many have attempted to define in various socio-economical contexts[12] Often innovation puts a strong emphasis on a successful implementation and adoption: *the production, assimilation, and exploitation of a value-added novelty.* By the fuzziness of its definition, quantifying or identifying innovation

[10] Both the ecosystem minimal criteria and the behavioral space are affecting this likelihood.

[11] We can witness innovation of different degrees and forms. For instance, Wolfgang Banzhaf Taylor et al. (2016) differentiate 'innovation within', 'model innovation', 'meta-model innovation', while Henderson and Clark differentiate 'incremental innovation', 'radical innovation', 'architectural innovation', 'modular innovation' Henderson & Clark (1990)

[12] Sociologist Everett Rogers defined it as: Rice & Rogers (1980): *"An idea, practice, or object that is perceived as new by an individual or other unit of adoption"*.

is a knowingly challenging task. However, *innovation persistence*[13], resp. *innovation reach*, judging the *extent* resp. the *diversity* of their offspring appear to be pertinent proxy metrics to examine.

(i) *Innovation persistence* is reflected by the frequency of its use among the elites throughout the evolutionary run. For $\mathfrak{p}$ a primitive, we define the activity score $\nu^i_{\mathfrak{p}}$ of $\mathfrak{p}$ at iteration i as its frequency among the elites of this generation:

$$\nu^i_{\mathfrak{p}} \coloneqq \frac{\sum_{\rho \in \mathcal{E}^i} \delta_{\mathfrak{p} \in \rho}}{\#\mathcal{E}^i}, \tag{7}$$

where δ_s is Kronecker symbol, being 1 if s is True, 0 else, and $\#\bullet$ denotes the size of a set.

(ii) *Innovation reach*, as mentioned in Taylor et al. (2016) reflects the idea that an invention has a diverse offspring. We adopt the following definition for the reach of $\mathfrak{p}$, measuring its offspring diversity:

$$\mu^i_{\mathfrak{p}} \coloneqq \sum_{\rho \in \mathcal{O}^i} \left(\delta_{\mathfrak{p} \in \rho}\right), \tag{8}$$

where $\mathcal{O}^i \subset \mathcal{E}^i$ is defined as the offspring of $\mathfrak{p}$ among iteration i. Similarly, we define a cumulative reach score $\mu_{\mathfrak{p}}$ for the primitive $\mathfrak{p}$ where $\mathcal{O}^i_{\mathfrak{p}}$ is replaced by the whole offspring of $\mathfrak{p}$ in the archive.

A.9 Role of Stochasticity in the Emergence of Innovation

Stochasticity, instabilities, and the possibility of historical contingencies [14] seems to be strongly tied to the emergence of Innovation throughout history[15]. In that regard, but also to lower the efficiency pressure (cf. Section A.7) we incorporated more stochasticity in OED learning, alongside hyperparameters to balance the desired stochasticity level: e.g. Bottom Up Sampling as explained in Section A.3, stochastic pruning mechanism of the library as detailed in A.5, or from the replays in the recognition model training and from the compression frontiers sent to the compressor. An illustration of an early run (iteration 2) is shown Figure 13, yet we leave further study about benefit and drawbacks of stochasticity for future work.

A.10 Related Works

Neuro-Symbolic Models Despite certain compelling advantages of robustness or interpretability, pure symbolic system, relying on logic and rule-based systems for instance, are often brittle to noise, uncertainty, expensive, and often neither scalable, nor adaptive. In that regard, *neuro-symbolic* models De Raedt et al. (2019); Hitzler (2022),are combining the strengths of both neural networks and symbolic reasoning. Leveraging the ability of neural networks to process large amounts of data, and borrowing certain symbolic reasoning capability from traditional symbolic systems, they can perform both perceptual and reasoning tasks. They can better adapt to new information and handle ambiguity and uncertainty, which is essential to tackle complex, and noisy real-world problems.

Program Synthesis *Program synthesis* refers to the automatic generation of computer programs commonly from either input/outputs examples(as Dream Coder

Language/Library Learning *Symbol learning* focus on learning symbolic representations from data and knowledge, and use these newly learned representations to perform tasks such as prediction, classification, and reasoning. Learning new abstraction (symbol learning, or more specifically even library learning) is a promising research question, which researchers have approached from different perspectives: some have looked at learning symbol from visual clues Mao et al. (2019), some from human demonstration, closer to this work, are learning DSLs or libraries by inferring reusable pieces

[13]"If the innovation is adaptive, it persists in the population with a beneficial effect on the survival potential of the components that have it. It persists not only in the component which first receives the innovation, but in all subsequent components that inherit the innovation."**?**

[14]I.e. "chance-influenced events with substantial long-term effects, that is, events which clearly take history down a different path than it otherwise would have followed". Taylor et al. (2016)

[15]In Taylor et al. (2016), Norman Packard suggest instabilities may lead to the emergence of innovation.

of code (e.g. Ellis et al. (2021); Devlin et al. (2017a); Nye et al. (2020); Tian et al. (2020); Liang et al. (2010); Devlin et al. (2017b); Ellis et al. (2018)), etc. Collateral to the idea of bias within programmatic symbolic knowledge, a recent work Kumar et al. (2022) look at how language – and therefore programs– are repositories of abstract prior knowledge, –and human bias– and can be used to elicit human-like bias in other tasks.

Novelty Search and Quality Diversity *Novelty Search* (NS) and *Quality Diversity* (QD)are approaches in machine learning that focus on finding novel and diverse solutions to complex problems, rather than directly focusing on finding the global optimum. Traditional NS approaches are purely novelty-driven, and have proven to be competitive on certain tasks, despite never having specified the objective Eysenbach et al. (2018). Quality Diversity algorithms Pugh et al. (2016) (e.g., novelty search with local competition Lehman & Stanley (2011a) and MAP-Elites Mouret & Clune (2015)) have adopted a more nuanced approaches of evaluating solutions both based on their performance and their novelty. To preserve diversity, often approaches enact forms of local competition, where solutions are only competing when being close enough in a behavioral space; cf. the 'bins' in MAP Elites which are inspired by ecological niches. Nguyen et al. (2015) is another interesting work in that line of novelty-guided search, where they learn investigate a 'deep notion of distance', learned with a neural model. These approach are particularly well-suited for problems that have multiple, often conflicting, objectives, but also as promising paths to tackle complex or insoluble tasks. Indeed, they are stemming from the belief than intelligent and complex behaviors would emerge rather from bottom-up approaches, focusing on gathering diverse skills rather than more top-down approaches, aiming directly to face a sometimes insoluble task Lehman & Stanley (2011b).

B EXPERIMENTS

B.1 TOWER BUILDING EXPERIMENTS

We refer to DreamCoder for more details about the Tower Building Environment. By default, the initial Library is seeded with two basic control flow primitives: `for` (for loop), and `get/set` which gets and saves the current state of the agent's hand. We also provide movement-related and construction-related primitives: `moveHand` (advancing the hand), `reverseHand` (flipping the orientation), `dropVerticalBlock` and `dropHorizontalBlock`. Equation 9 displays examples of λ calculus expression turned into programs.

To define the niches, we adopt a three dimensional behavioral characteristic, encompassing the center of gravity g_ρ, width w_ρ, and number of blocks b_ρ of the artifact generated from a program ρ. The *ecosystem minimal criteria* implemented consists of a minimum resp. maximum number of blocks (e.g. 5 resp. 500), a maximum width and height of the structure (e.g. 300). The *behavioral characteristics* are defining the ecological niches, and are therefore defining the locality of the competition in our Quality Diversity Approach. Meanwhile, the *phenotypic features* defined in Section A.2 are controlling the local novelty search within each niche. The niches can be seen as a relaxation of the supervised tasks in DreamCoder, as higher-dimensional subspaces of the behavioral space.

Figures 10, 8, and 5 give examples of some of the programs produced, dreamed or abstracted across all experiments. The elite programs, first selected based on an efficiency-compression criteria, typically present stronger regularity than the dreams, more widely sampled and therefore resulting in more 'diversity', notably in early stages. The niche distribution is displayed in Figure 11 for some of our experiments, and the growing hierarchical Library for an Innate run can be seen in Figure 12. It is important to note that the majority of experiments presented here are subjected to a significant pressure for efficiency (notably a strong language bias), strongly biasing the search towards the most efficient solutions to the detriment sometimes of novelty. However, as detailed Section A.7, OED enable to regulate this balance, which could be further investigated in future experiments.

Quantitative evaluation metrics presented in Figures 6 & 7 include:

(a) Ratio of populated niches across iterations, where the dashed line is the number of sampled niches.

(b) Number of new elites found in each iteration.

(c) Search Times for the Bottom Up enumeration procedure.

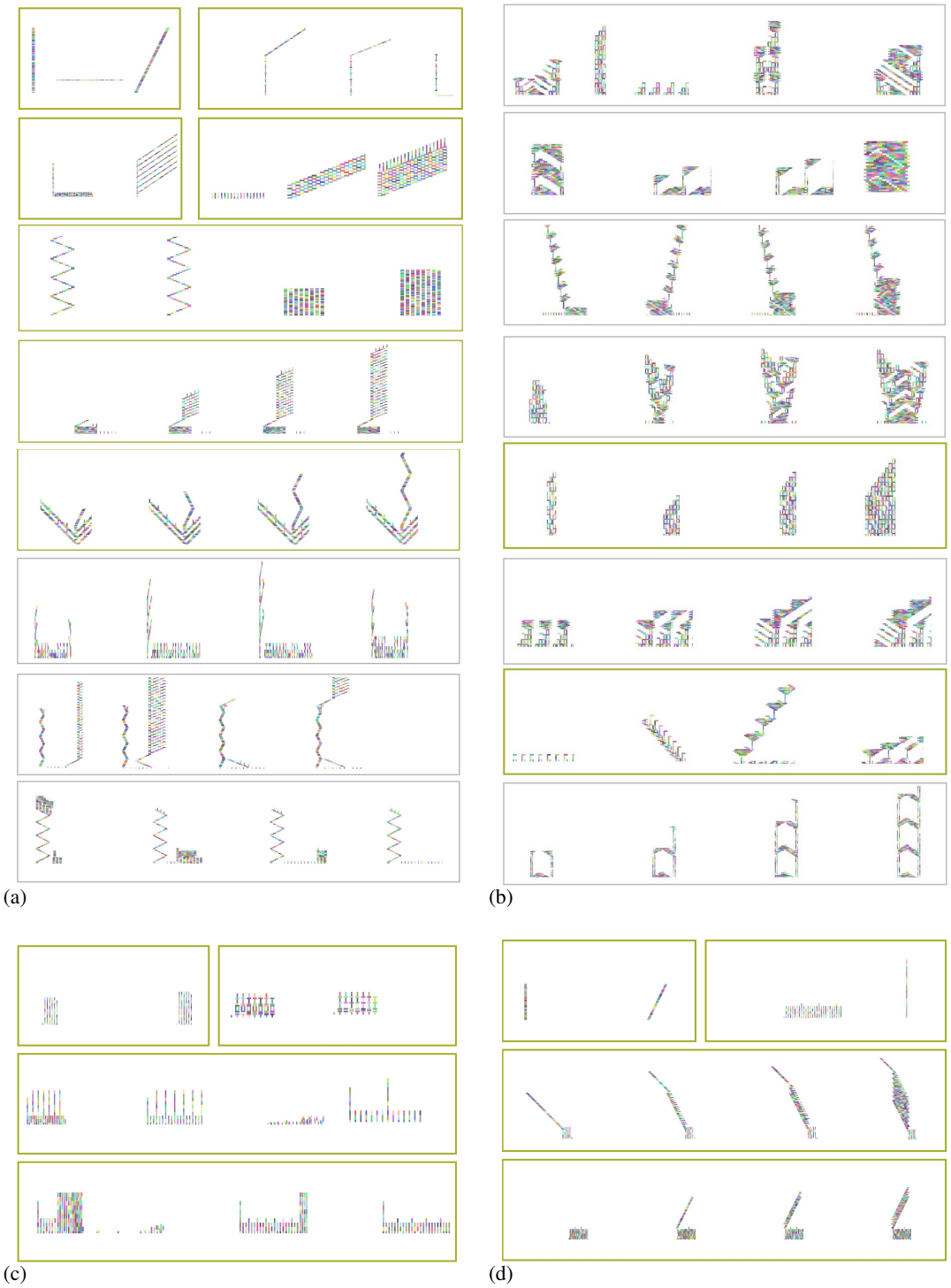

Figure 5: Primitives (programmatic abstractions), added to the library $\mathfrak{L}$, from early (top rows) to late (bottom rows) stages of evolution, across different run. Each box represent a primitive (programmatic function, of arity ≤ 3), instantiated with different arguments (e.g. numbers from 0 to 9). (a) BASE Run (CNN), (b) INNATE (c) PHYSICS (d) RAW (downsampling). Instance of Convergent Evolution, i.e. primitives which are been reinvented across run, are highlighted in green.

(d) Ratio of legit programs found, either through Bottom Up or Top Down (dashed). Note that missing parts in the Top Down ratio indicates the Top Down enumeration had been skipped, since no new element in the library had been added in the previous iteration.

(e) Log Priors ($P(\rho \mid \mathfrak{L})$) of the Elites, which are a measure of how compressed are the programs found relatively to the generative model $\mathfrak{L}$, therefore reflecting how 'efficient' are the elites.

(f) Local Novelty Scores, either historical (i.e the local novelty scores of the new elites at a given iteration, computed against the current existing archive as explained in A.4) or final (i.e. $KMean$ of the distance of computed at each iteration considering t) (dashed line).

B.2 Ablation Experiments

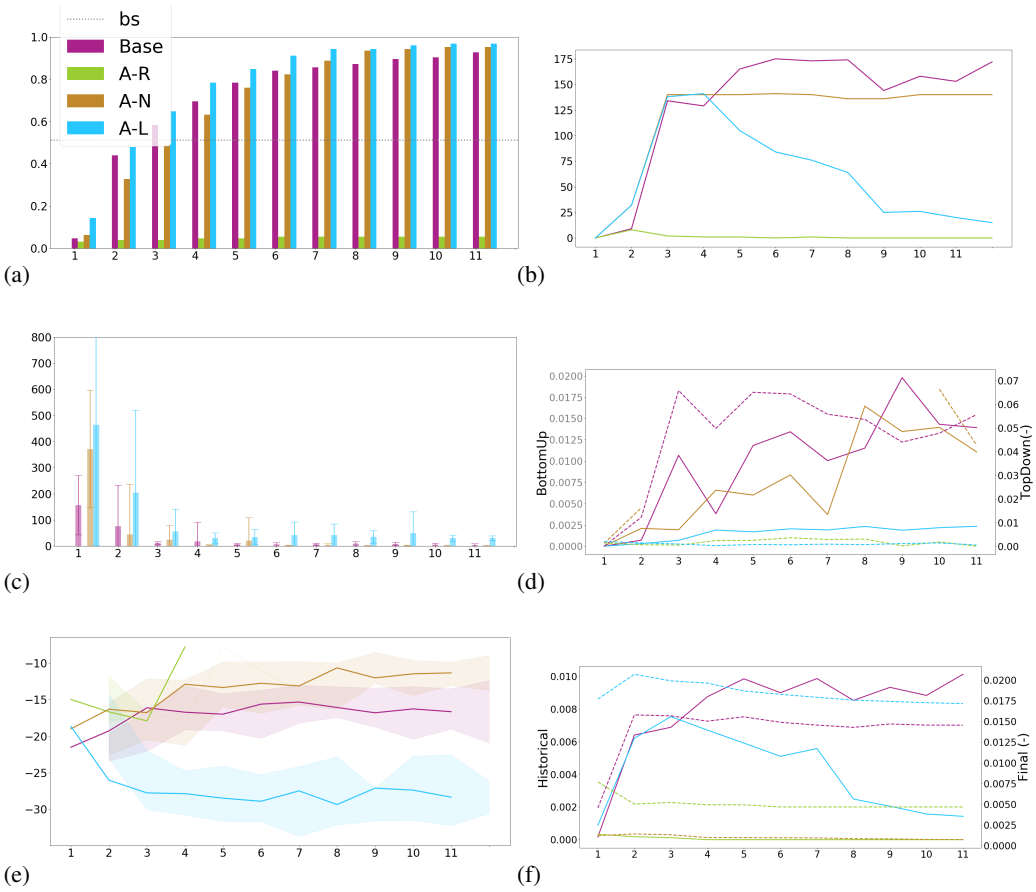

Figure 6: Comparative ablation experiments on (a) Niches Population (b) Number of Elites (c) Bottom Up Enumeration Search Times (d) Ratio Legit Programs found, either Bottom Up or Top Down (dashed) (e) Log Priors (f) Novelty Scores, historical and final (dashed).

To assess the benefit of the different components of our model in the open-ended creation of diverse programs, we run several ablation studies both (A-L) without the library, (A-R) without the recognition model, (A-N) without the novelty-Selection. Comparative results are presented in Figure 6. Qualitative results in Figure 10, 5 confirms quantitative results in terms of the overall impoverishment of found solutions in these ablation experiments. Some main take-away from these experiments:

(A-L) As reflected in Figure 6, Library learning helps to sustain a flow of novel and diverse programs (as the number of found solutions is drastically decreasing after the 4th iteration, cf. (b)) but also in terms of efficiency, both in terms of the programs created (log priors

becomes much smaller for (A-L), cf. (e)), and in terms of the learning itself (the search time is an order of magnitude higher in (A-L), cf.(c), or the ratio of legit programs is poor, cf. (d)). The diversity exhibited from (A-L) run, as reflected Figure 10 seems much wider than in a base run, notably at start. In this first non-challenging environment, niches are still mostly populated under library ablation, as reflected in Figure 6, (a) or Figure 11, albeit distinctly less populated than in a (BASE) Run. We expect that in challenging environment, *library learning* would be a pivotal point to populate the ecological niches and overcome more complex challenges.

(A-R) The neural guidance seems an essential component to the discovery programs, both in terms of efficiency and novelty. The non-guided learner in the [(A-R) experiments do not manage to find solutions above the novelty threshold after a few iterations nor to populate most of niches, as reflected in Figure 6 (a), (b), (d), (f). The neural guidance is crucial in such neuro-symbolic model since it helps facing the issue of the intractability of the symbolic search space due to its combinatorial nature.

(A-N) Without the novelty pressure, the learner manage to populate niches (since the novelty threshold is removed, it becomes a much easier task), yet, the archive exhibit a very low final diversity score. The elites found seems to enact efficient ways to populate the niches (e.g. the widening parallelogram type of structure present in Figure 10 to fit all the niches), yet crucially, lack of diversity.

B.3 Biasing Symbol Learning

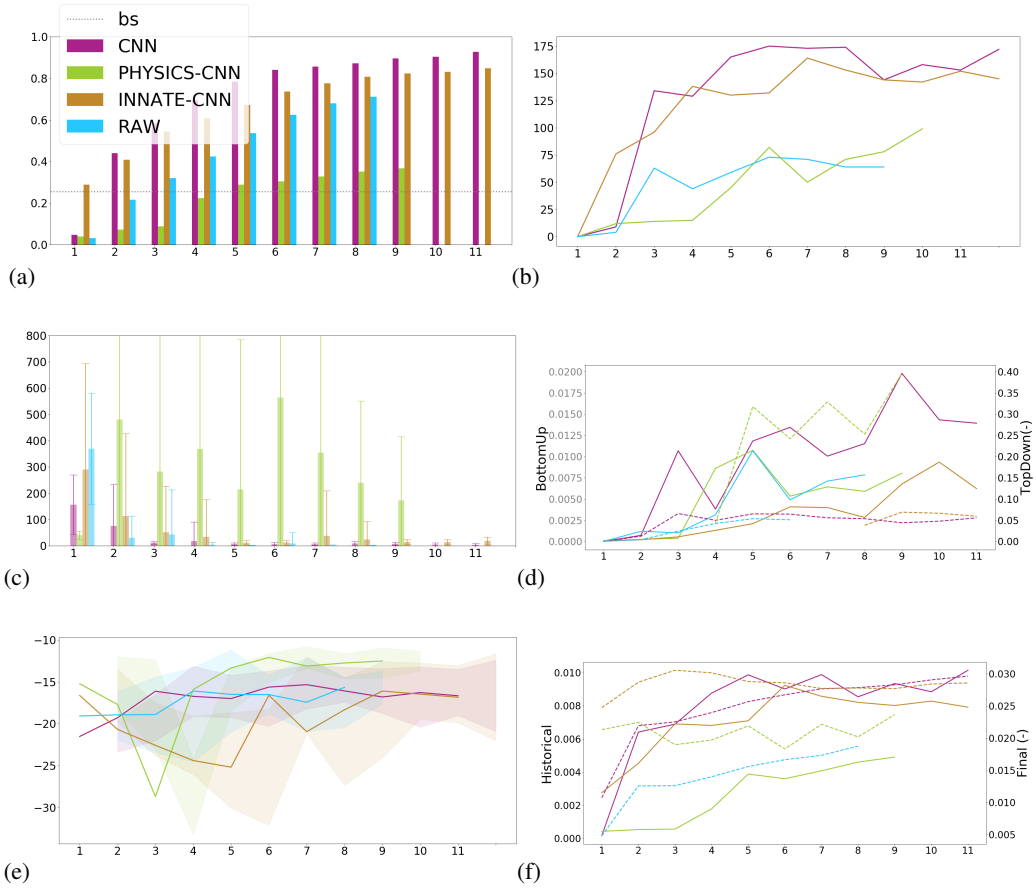

Figure 7: Bias Study, with (INNATE), (CNN) and (PHYSICS) biased run, and the minimally biased (RAW) run. (a) Niches Population (b) Number of Elites (c) Bottom Up Enumeration Search Times (d) Ratio Legit Programs found, either Bottom Up or Top Down (dashed) (e) Log Priors (f) Novelty Scores, historical and final (dashed).

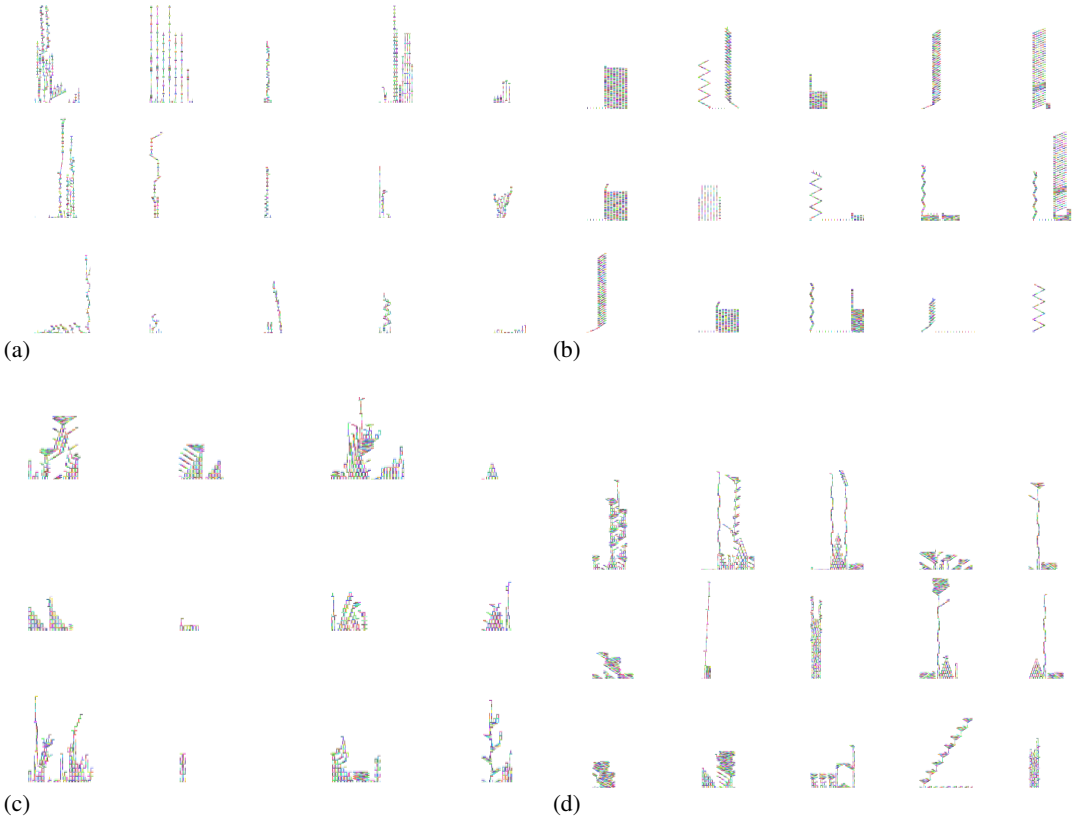

Figure 8: Evolution of Dreams through Iterations. (a) Base Early Dreams ($it = 1$) , (b) Base Late Dreams ($it = 9$) (c) Innate Early Dreams ($it = 1$) (d) Innate Late Dreams ($it = 9$). Progressively, dreams displays more structure, and more bias.

The emphasis on *implementation* nested in the notion of innovation implies the candidate novel artifacts-ideas will likely undergo strong environmental pressures (e.g. market pressure, functional pressures, social pressures etc.) when being put at test in the real world, heavily biasing the search space. It seems therefore legitimate to wonder if strong inductive biases and environmental coupling are key to guide the emergence of abstract structured knowledge and innovation. Generating and retaining 'stepping stones to everywhere' seems indeed too naive -ambitious in such a combinatorially exploding space of programs in the absence of any environmental pressure (what survive, what functions, etc.). Coupling such quest with some form of inductive bias seems both more realistic to how symbolic knowledge formation takes place in social and natural systems but also more promising to lead to more consistent innovations. Such bias may take the forms of innate knowledge or environmental pressure.

This leads us to investigate the effect of bias on the formation of symbolic knowledge for our learner across different experiments: (INNATE) relying on a bootstrapped library, (PHYSICS) enacting environmental pressures and (CNN) using biased metrics, discussed in paragraphs below. Qualitative results are displayed Figure 7, while quantitative results are reflected through the selected, dreamed or abstracted programs in Figure 10, 8, 5. We also discuss below the *language bias* carried on by the library learning, omitting here the bias carried by the Domain Specific Language itself on which we ground our experiment.

We observe throughout all these experiments that strong bias has the effect of narrowing the search space, which in some case can be detrimental to diversity, despite being a necessary part to constraint symbol learning.

Strong bias also have led to phenomenon of convergent evolution, discussed Section **??**, and displayed in the learned primitives Figure 5.

(INNATE)-Boostrapped Library In the Innate Experiment, to bootstrap evolution, we seed the initial library with a set of 5 handpicked primitives [16] –bridge, staircase, pyramid, towerArch, and bricks, displayed in Equation 9.

$$
\begin{array}{ll}
\textit{bricks}: & \lambda(\lambda(\texttt{for}\ \ \$0(\lambda(\lambda(\texttt{moveHand3}(\texttt{reverse}(\texttt{for}\$3(\lambda(\lambda(\texttt{move6}(\texttt{H}\$0))))\$0)))))))) \\
\textit{staircase}: & \lambda(\texttt{for}\ \ \$0(\lambda(\lambda((\lambda(\lambda(\texttt{for}\$1(\lambda(\lambda(\texttt{get.set}(\lambda((\lambda(\texttt{V}(\texttt{move4}(\texttt{V}(\texttt{reverse}(\texttt{move2} \\
 & (\texttt{H}\$0)))))))\$0))\$0)))\$0)))\$1(\texttt{move6}\$0)))))) \\
\textit{bridge}: & \lambda(\lambda(\texttt{for}\$0(\lambda(\lambda((\lambda((\lambda(\lambda(\texttt{for}\$0(\lambda(\lambda(\texttt{V}(\texttt{move4}(\$3\$0)))))(\texttt{move2}(\texttt{H}\$2))))))\$0 \\
 & (\lambda(\texttt{reverse}\$0))))(\texttt{move4}\$0)\$3))))) \\
\textit{towerArch}: & (\lambda(\lambda((\lambda((\lambda(\lambda(\texttt{for}\ \ \$0(\lambda(\lambda(\texttt{V}(\texttt{move4}(\$3\$0)))))(\texttt{move2}(\texttt{H}\$2))))))\$0(\lambda \\
 & (\texttt{reverse}(\texttt{V}\$0)))))\$0\$1))) \\
\textit{pyramid}: & (\lambda(\texttt{for}\ \ \$0(\lambda(\lambda(\texttt{move6}(\texttt{get.set}(\lambda(\texttt{reverse}((\lambda(\lambda(\texttt{for}\ \ \$1(\lambda(\lambda(\texttt{move}\$2(\texttt{V} \\
 & (\texttt{move2}((\texttt{get.set}(\lambda(\texttt{move2}(\texttt{V}\$0)))(\texttt{H}\$0)))))))))\$21\$0)))\$0)))))))
\end{array}
\tag{9}
$$

where `V`, `H`, `reverse` , `move` are shortcuts for `dropVerticalBlock` resp. `dropHorizontalBlock`, `reverseHand`, `moveHand`

Innate Priors helps bootstrap the search for novel programs as the programs generated, dreamed or abstracted are qualitatively consistently more diverse (Figures 10, 8, 5). First, we observe that first invented abstractions are happening typically much later than for a BASE run. This is likely due to the fact that the bootstrapped library helps to face the environmental pressure (i.e. populate the niches) from early stage on, there is therefore less pressure on abstracting new primitives on early stages. Secondly, it seems it does not bring a strong quantitative advantage, which reflects a relatively poor diversity metric, but also the possible downsides of library bootstrapping: too specific primitive priors, carrying too much bias, may affect the final diversity: e.g. structures carried on similar building blocks (e.g. pyramid) would easily be judged similar.

(PHYSICS) - Environmental Pressure Experiment Stronger environmental pressures is another way to bias the formation of symbolic knowledge towards a more consistent path. As a first example, we experiment with a gravity-aware environment, using Box2D Physics Engine (gravity being set at 9.8), with an optional penalty on the ratio of fallen blocks. Since the Physics Engine noticeably slow down the simulation, and because of the stronger challenges brought by this constrained environment, it considerably slows down the search for diverse and complex artifacts, It explains the lower diversity of artifacts displayed in Figure 10 and the poorer niche distribution in Figure 11. Enabling much longer evolutionary run could open up ultimately for more diversity. Yet, in counterpart, because of stronger environmental pressures, the programmatic abstractions seems more consistent, and display phenomenon of convergent evolution, as discussed Section B.4.

(CNN) - Biased Metrics Underlying the novelty selection mechanism is a crucial choice of a notion of distance. As mentioned in Section A.2, we experiment with two ways to define the phenotypic distance: either relying on a downsampled version of the image (referred to as (RAW) in our experiments), or relying on a visual features obtained from a pretrained CNN, referred to as (CNN) or (Base) in our experiments). We use VGG 16 Simonyan & Zisserman (2014) for our experiments, and obverse that this model has not be fine-tuned nor berry-picked. Extracting high-level and hierarchical visual features through a pre-trained visual neural model, enacts a bias in the diversity quest. Comparing (CNN) run with the (RAW) run in Figure 7 suggest that biasing the novelty-search via a more pertinent metric has, expectedly, a strong impact on the observed diversity (cf (f)), and the amount of novel (cf. (b)) programs found. We also suspect that fine-tuning this metric to relevant data, would help enrich the diversity of found solutions.

B.4 Efficiency Pressure

Convergent Evolution Strong efficiency pressures, either from environmental constraints (as in the (PHYSICS) run), or via stronger language-pressure (cf. (BASE) run), or conversely lower novelty pressure (as in the (A-N) run), seem to favors phenomenon of *convergent evolution* and the repeatability of certain evolutionary outcomes, as we observed in our experiments. Indeed, certain programs and primitives were (approximately) reinvented across different evolutionary runs, as they

[16] These primitives are extracted from supervised examples in DreamCoder, Figure $5.B$, Ellis et al. (2021)

appeared to be some of the most efficient way to inhabit certain ecological niches and fit the minimal criteria (notably when this one is more demanding as in the gravity-aware environment): e.g. slanted lines, cranes-type structure, parallelograms, comb-like artifact, and combinations of them as displayed in Figure 5 were commonly reinvented across these runs. Loosening up the efficiency pressure, and increasing the novelty pressure would encourage more divergent evolution, desirable in the open-endedness quest.

Language Bias As reflected in the elites invented or dreamed (figure 10, 8), learning a language also implies learning a strong inductive bias towards the search space. This language bias is notably enacted through the prior $\mathbb{P}(\rho \mid \mathfrak{L})$ pressure in OED learning, stemming from the generation step detailed in Section A.3. Under a strong language bias –as in most of our experiments–, early primitives incorporated in the library (e.g. the noticeable "zigzag" artifact in the Base run, which has been added to the library at the second iteration, cf Figure 5) have a strong effect on the generation of new programs in the later run. Such bias is also reflected in the evolution of dreams between the initial and last iteration in Figure 8, in the (BASE) run, the dynamic generative model resulting from the learned library displays throughout evolution a shift from a more chaotic diversity to a more structured yet narrow diversity. However, this language bias can be moderated by lowering the efficiency pressure as discussed in Section A.7, or thanks to the stochastic pruning mechanism we propose Section A.5.

B.5 Library Learning and Emergence of Innovation

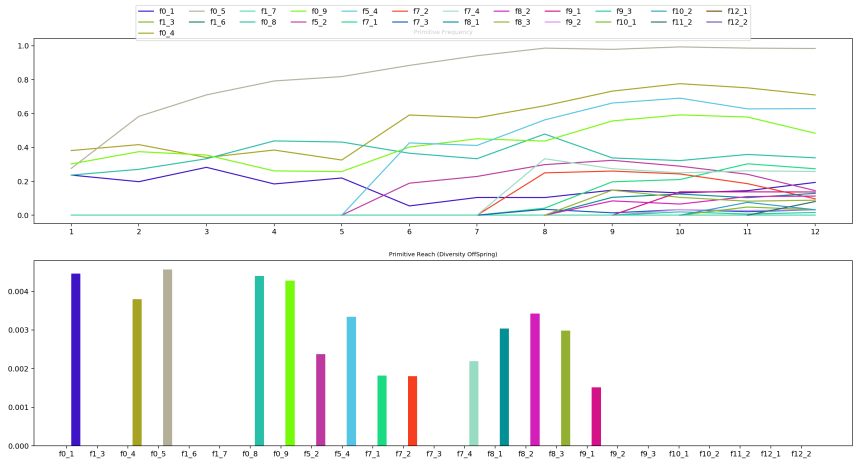

Figure 9: Primitive Metrics, both (top) innovation persistence and (bottom) innovation reach, for the Innate Run.

The *innovation persistence* and *innovation reach* as defined in Section A.8 from some experiments is displayed in Figure 9. There, we relied on a sample of the offspring to evaluate the innovation reach scores $\hat{\mu^i_{\mathfrak{p}}}$, resp. $\hat{\mu_{\mathfrak{p}}}$. Although the evolutionary scale is too restricted to comment on long-term effect, we can observe some interesting fluctuations with certain primitives activity rising, or exhibiting persistence, while others are almost vanishing throughout evolution. Despite being generally favored, some initial prior or primitive may fade out (such as the pyramid $f0_1$ in the Innate run), while others early or late primitives may rise (cf. the wall $f0_5$, or the promising $f5_4$, in the Innate Run displayed in Figure 9).

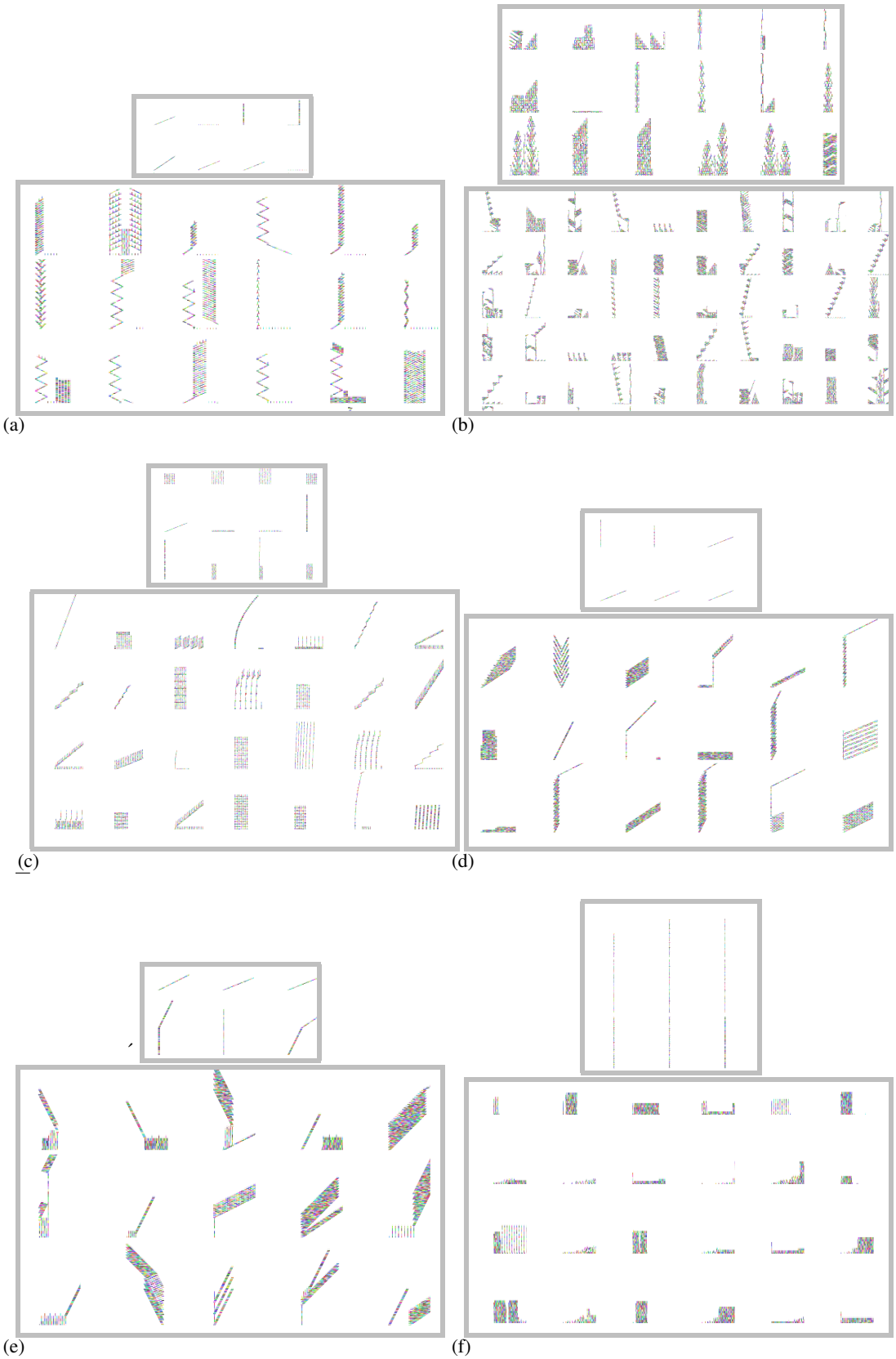

Figure 10: Evolution of the Elites artifacts across experiments, for (top) early ($i = 1$) resp. (bottom) late ($i = 9$) iteration. (a) BASE, (b) Innate (c) Library Ablation (d) Novelty Ablation (e) Raw (f-top) Recognition Ablation (f-bottom) Physics. Note that the fallen blocks in (f) are not properly rendered.

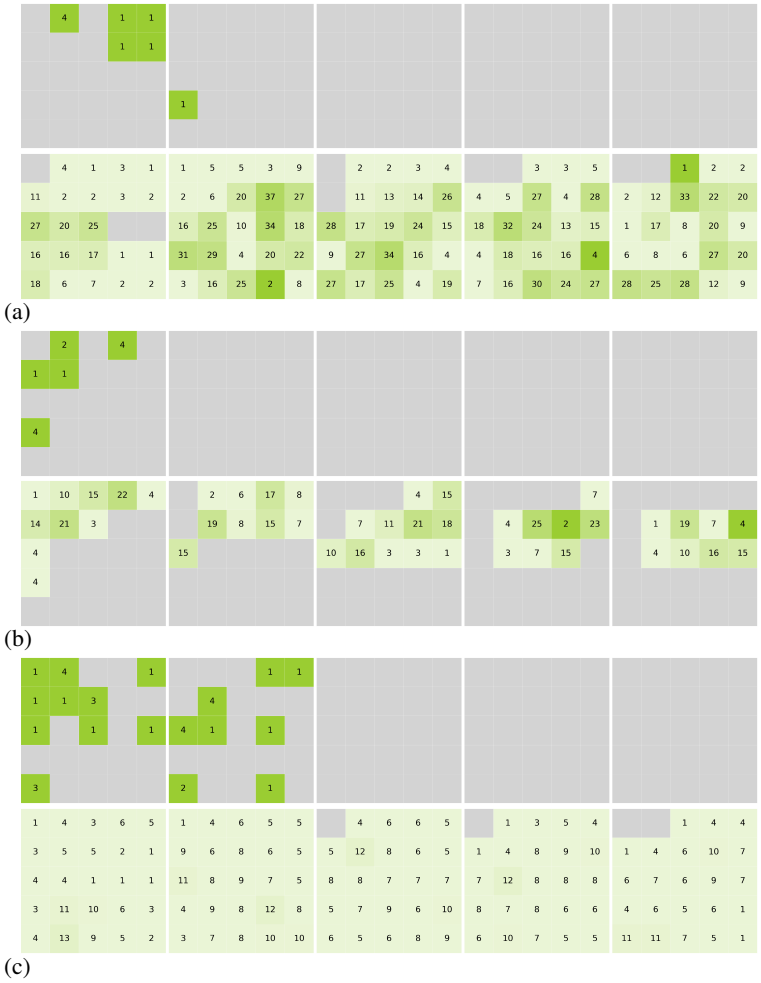

Figure 11: Population distribution across niches at different iterations (top rows 1st iteration, bottom last iteration) for different runs. Niches freshly populated are highlighted in green. Unpopulated niches are displayed in grey. Numbers indicate the population size. The axis stands for a behavioral feature: X axis represent the Width, Y axis the Gravity, and the Number of Block are increasing from left to right. (a) BASE environment $it = 1$, $it = 11$ (b) (A-L) Library Ablation (c) PHYSICS

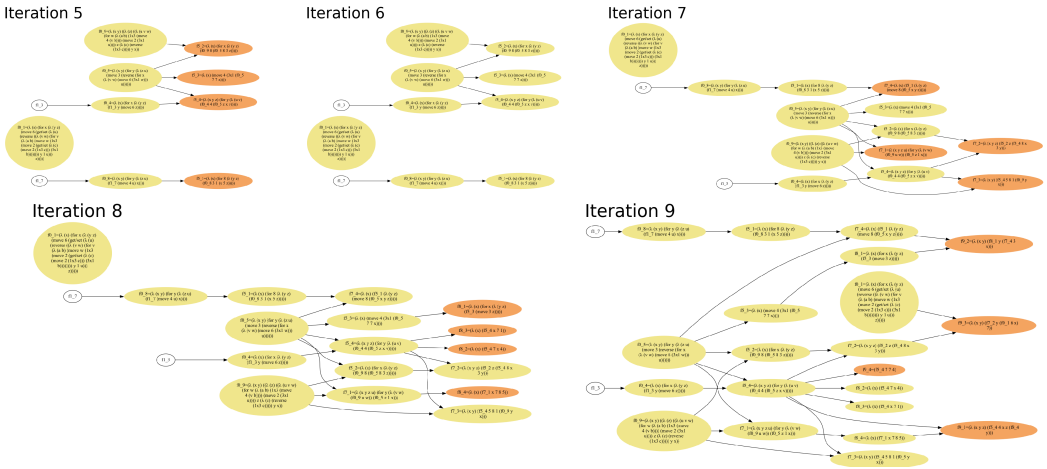

Figure 12: Extract of a Growing hierarchical Library, across iterations for the Innate Run. Lower Iterations are not displayed as presenting no new primitives. Orange primitives indicates novel primitives.

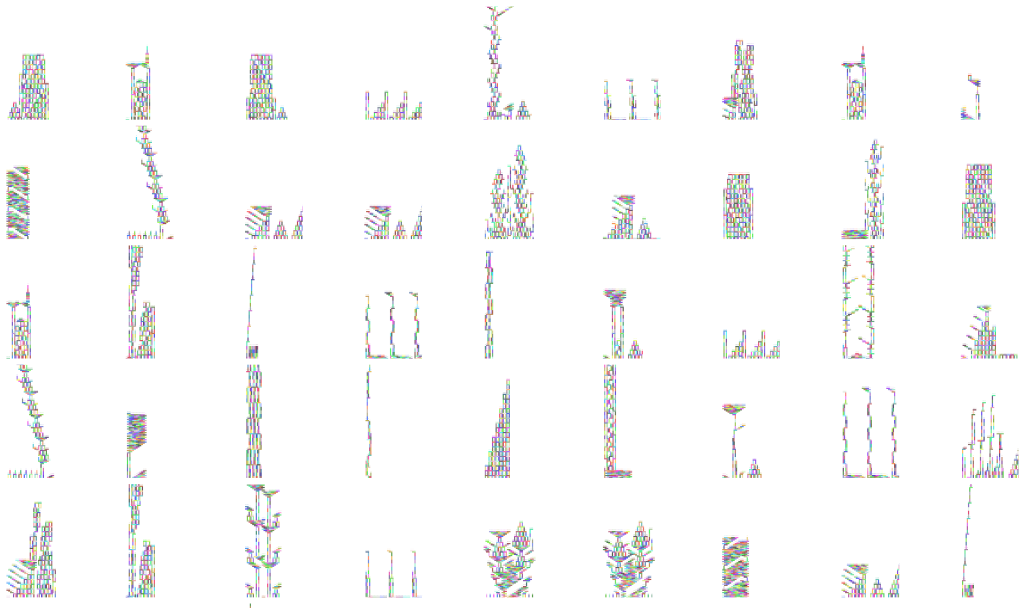

Figure 13: Extract of Early Elites ($it = 2$) from a Stochastic run (Stochastic INNATE).