Efficient adaptation of reinforcement learning agents : from model-free exploration to symbolic world models

Pierre-Alexandre Kamienny

Sorbonne Université
École Doctorale Informatique, Télécommunications et Électronique (Paris)

Thèse de Doctorat

Spécialité **Informatique**

présentée par

Pierre-Alexandre Kamienny

Efficient adaptation of reinforcement learning agents: from model-free exploration to symbolic world models

---

Adaptation efficace des agents appris par renforcement: de l'exploration *model-free* aux modèles symboliques

---

sous la direction de **Sylvain Lamprier** et de **Ludovic Denoyer**.

Présentée et soutenue le **4 octobre 2023**, à **Paris** devant le jury composé de

Prof Emmanuel **Rachelson**	ISAE-SUPAERO, Université de Toulouse	Rapporteur
Prof David **Filliat**	U2IS, ENSTA Paris	Rapporteur
Prof Olivier **Sigaud**	Sorbonne Université, ISIR	Président du jury
Prof Sara **Silva**	LASIGE, Universidade de Lisboa	Examinateur
Prof Olivier **Pietquin**	Google DeepMind	Examinateur
Prof Sylvain **Lamprier**	LERIA, Universite d'Angers	Directeur de thèse
Prof Ludovic **Denoyer**	Ubisoft	Encadrant industriel

Remerciements

First and foremost, I am deeply grateful to my supervisors, Sylvain and Ludovic. Your scientific guidance, unwavering support, and understanding have been instrumental in shaping this thesis. I am particularly thankful for your patience and openness, even during moments when I ventured into uncharted territories. I truly hope that our professional relationship will continue to flourish, and we will have the opportunity to share many more enjoyable moments over beers!

I consider myself incredibly fortunate to have collaborated with a brilliant group of researchers and fellow PhD students. To Jean, Matteo, and Alessandro, who contributed to the RL projects, I extend my warmest thanks. Your support, expertise, and camaraderie have been invaluable in pushing our papers to completion and boosting my confidence during moments of doubt. I have learned so much from each of you, both intellectually and on a personal level. To François, Stéphane, Guillaume, and Marco, who were involved in the SR projects, I express my deepest gratitude. Your dedication, insights, and teamwork have significantly enhanced the quality of our research. Working alongside you has been an absolute pleasure, and I am grateful for the positive impact you have had on my journey.

A special mention goes to the exceptional team of Cifre PhD students who have shared countless adventures (FIFA represents a non-negligible part) with me on a daily basis. Evrard, Jean, Jean-Baptiste, Paul-Ambroise, Lina and all others, you have made this PhD experience truly enjoyable. Your friendship and support have made the challenging moments more bearable and the successes even sweeter.

I would also like to extend my sincere gratitude to Meta for their confidence and financial support throughout my PhD. To the faculty and staff of Sorbonne University, I am sincerely thankful for creating an intellectually stimulating environment that has nurtured my academic development.

These past three years have been an intense challenge, and I would like to express my deepest appreciation to Hermione for her unwavering support. Your presence,

encouragement, and belief in me, especially during the difficult and uncertain times, have been invaluable. I am truly grateful to have you by my side. Lastly, none of this would have been possible without the immense support from my family. Maman, Papa, Clémence, Marie, Papi, and Mami, your love, encouragement, and unwavering belief in me have been the bedrock of my success. I am forever grateful to have such an incredible support system in my life.

Abstract

Reinforcement Learning (RL) encompasses a range of techniques employed to train autonomous agents to interact with environments with the purpose of maximizing their returns across various training tasks. To ensure successful deployment of RL agents in real-world scenarios, achieving generalization and adaptation to unfamiliar situations is crucial. Although neural networks have shown promise in facilitating in-domain generalization by enabling agents to interpolate desired behaviors, their limitations in generalizing beyond the training distribution often lead to suboptimal performance on out-of-distribution data. These challenges are further amplified in RL settings characterized by non-stationary environments and constant distribution shifts during deployment.

This thesis presents novel strategies within the framework of Meta-Reinforcement Learning, aiming to equip RL agents with the ability to adapt at test-time to out-of-domain tasks. The first part of the thesis focuses on model-free techniques to learn effective exploration strategies. We consider two scenarios: one where the agent is provided with a set of training tasks, enabling it to explicitly model and learn generalizable task representations; and another where the agent learns without rewards to maximize its state coverage. In the second part, we investigate into the application of symbolic regression, a powerful tool for developing predictive models that offer interpretability and exhibit enhanced robustness against distribution shifts. These models are subsequently integrated within model-based RL agents to improve their performance. Furthermore, this research contributes to the field of symbolic regression by introducing a collection of techniques that leverage Transformer models, enhancing their accuracy and effectiveness.

In summary, by addressing the challenges of adaptation and generalization in RL, this thesis focuses on the understanding and application of Meta-Reinforcement Learning strategies. It provides insights and techniques for enabling RL agents to adapt seamlessly to out-of-domain tasks, ultimately facilitating their successful deployment in real-world scenarios.

Résumé

L'apprentissage par renforcement (RL) est un ensemble de techniques utilisées pour former des agents autonomes à interagir avec des environnements de manière à maximiser leur récompense. Pour déployer avec succès ces agents dans des scénarios réels, il est crucial qu'ils puissent généraliser à des situations inconnues. Bien que les réseaux de neurones aient montré des résultats prometteurs en permettant aux agents d'interpoler des comportements souhaités, leurs limites en termes de généralisation au-delà de la distribution d'entraînement entraînent souvent des performances sous-optimales sur des données issue d'une distribution différente. Ces défis sont encore amplifiés dans les environnements de RL caractérisés par des situations non stationnaires et des changements constants de la distribution lors du déploiement.

Cette thèse présente de nouvelles stratégies dans le cadre du meta-RL visant à doter les agents RL de la capacité à s'adapter sur des tâches différentes du domaine d'entraînement. La première partie de la thèse se concentre sur les techniques *model-free*, c'est à dire qui ne modélisent pas explicitement l'environnement, pour apprendre des stratégies d'exploration efficaces. Nous examinons deux scénarios : dans le premier, l'agent dispose d'un ensemble de tâches d'entraînement, ce qui lui permet de modéliser explicitement les tâches et d'apprendre des représentations de tâches généralisables ; dans le second, l'agent apprend sans récompense à maximiser la couverture de l'espace des états.

Dans la deuxième partie, nous explorons l'application de la régression symbolique, un outil puissant pour développer des modèles prédictifs offrant une interprétabilité et une meilleure robustesse face aux changements de distribution. Ces modèles sont ensuite intégrés aux agents *model-based* pour améliorer la modélisation de la dynamique. De plus, cette recherche contribue au domaine de la régression symbolique en introduisant une collection de techniques exploitant les modèles génératifs, en particulier le *Transformer*, ce qui améliore leur précision et leur efficacité.

En résumé, cette thèse aborde abordant le défi de la généralisation et adaptation dans le RL. Elle développe des techniques visant à permettre aux agents meta-RL de s'adapter à des tâches hors domaine, facilitant ainsi leur déploiement dans des scénarios du monde réel.

Contents

Chapter 1

Introduction

1.1 Success of Deep Learning and RL

Machine learning. Machine learning (ML) is a branch of artificial intelligence that utilizes data to improve computer performance by giving machines the ability to "learn". ML algorithms construct models based on sample data, referred to as training data, enabling them to make predictions or decisions without explicit programming. ML finds application in diverse domains, e.g. healthcare, fraud detection, image and speech recognition or recommendation systems, where developing algorithms for the required tasks is challenging.

ML encompasses various learning paradigms, such as supervised, unsupervised, self-supervised, meta, and reinforcement learning. In supervised learning (SL), algorithms encounter labeled data, where each data point comprises features and an associated label. The objective of supervised learning algorithms is to learn a function that maps feature vectors to labels. On the other hand, unsupervised learning algorithms do not have access to any labels; instead, they learn compact representations of the input data, which can be employed for tasks like data analysis, generating new data, or downstream applications. Both these techniques can be applied to one-step prediction problems or sequential problems where the prediction at a given time-step relies on previous predictions.

Reinforcement learning (RL) deals with multi-step prediction problems and occupies an intermediate position between supervised and unsupervised learning in terms of supervision. RL encompasses a range of techniques that model the learning process as a decision-making problem, in interaction with an environment. RL algorithms

typically have access to utility feedback instead of expert labels (optimal decisions). Agents are learned via a trial-and-error approach, where predictions (referred to as actions within the Markov decision process framework) and observed rewards serve as learning signals. The assumption of access to samples of the reward model is often weaker than in supervised learning, involving a factorization based on human knowledge of the problem, rather than human intervention for each example [1].

By optimizing an agent's actions to achieve specific objectives, RL facilitates the automation of intricate tasks that necessitate continuous adjustments. RL presents several challenges, including credit assignment, sparse or deceptive reward signals, and exploration. These difficulties have led to RL being described as *the cherry on the cake* by Yann Le Cun at NeurIPS 2016.

Deep Learning. Deep learning (DL) encompasses a set of techniques that use neural networks as function approximators across various learning paradigms mentioned previously. While the concept of deep learning has existed since the 1940s and 1950s, when researchers initially proposed neural networks as a means to emulate the human brain, its impact on machine learning has significantly transformed the field of ML, particularly in the early 2010s, resulting in a remarkable paradigm shift. Their application to RL, also named Deep Reinforcement Learning (DRL), has revolutionized problem-solving in complex scenarios. For instance, DRL enabled the achievement of remarkable milestones, such as AlphaGo defeating the world Go champion [Sil+17]. RL also plays a significant role in controlling physical systems like robots [Lil+15; Akk+19], vehicles [Boj+16], data center climate control [Laz+18], and even stratospheric balloons [Bel+20]. The recent architectural advancement of Transformer models [Vas+17] has garnered substantial attention from academic research and industry, presenting numerous applications.

The Generalization Problem. In a typical SL pipeline, ML algorithms minimize errors on a set of training observations, hoping to capture the underlying structure of the task being addressed. However, ML algorithms can encounter difficulties when faced with data points that significantly differ from the distribution of the training data, commonly known as out-of-distribution (OOD) samples. The ability of a learned predictor or agent to extrapolate from the training data to OOD samples is crucial and is referred to as *generalization*. When carefully tuned, NNs can obtain impressive generalization performance, however generalization becomes particularly challenging

[1] However, defining a reward function can be a complex task.

when the training data is limited in its representation of the overall task, due to limited data or a lack of diversity in the samples. The problem of generalization is also important in RL in two cases: i) when learning a single task, the model (can be a policy or a value-function) should be accurate on unvisited states, ii) when trained on multiple training tasks, an agent should perform well on unseen but similar tasks.

Synergies between Learning Paradigms. When ML practitioners have access to labeled datasets, they typically employ supervised learning (SL). However, SL operates on fixed datasets, meaning that once the training phase is completed, the model lacks the capability to improve its performance on OOD samples during testing. This limitation becomes apparent in cases such as GPT-3 [Bro+20], a conversational agent trained on a large corpus of textual data, which may exhibit significant errors when encountering OOD samples, including displaying toxic behavior or generating factually incorrect claims. In that case, RL can help improve generalization of SL models by augmenting them with the ability to self-improve at test-time. RL addresses this issue by providing tools to actively seek new information while leveraging existing knowledge, in order to cope with the distribution-shift problem. This process assumes the formulation or learning of a reward function, which can be derived, for example, from human preferences. The combination of RL and SL has demonstrated its efficacy in various applications. For instance, the enhancement of GPT-3 involved incorporating feedback from humans using RL and preference models, leading to the development of ChatGPT. As ChatGPT collects more conversational data across a broader range of topics and engages in meaningful interactions with users, its performance can improve over time.

The "cherry on the cake" is today widely recognized as an essential tool for improving the learning process. Jiang et al. [JRG23] manifesto concludes that solving general intelligence necessitates RL exploration as an integral component of the learning process. Relying solely on fixed datasets would limit our ability to achieve this goal, and exploration is essential to augment datasets and gather as much information as possible about the task at hand.

1.2 Motivation and challenges

As we aim to deploy intelligent agents in the physical world that seamlessly integrate into our daily routines, it is essential for these agents to possess the ability to adapt to

unfamiliar scenarios. Adaptability distinguishes genuinely intelligent systems from mere executors of pre-programmed instructions. Adaptibability is all the more a critical characteristic as existing research in the field of RL has primarily focused on learning from simulated environments agents whose purpose is deployment in the physical world (*"sim2real"* transfer [Ope+18]); this challenge is often referred to as the *reality gap*. What defines adaptability? To meet these expectations, the agents must possess the following attributes:

- General and Useful: Agents should demonstrate versatility and practicality across a range of situations.
- Truly Autonomous: It is crucial that agents can rapidly acquire new skills and knowledge without relying heavily on human supervision or intervention.

What defines efficiency in terms of adaptability? To be efficient, we want adaptation to happen as fast as possible, more precisely with the minimal number of interactions possible, as these are generally responsible for larger running times or human interventions.

"To achieve efficient adaptation, you must skillfully harmonize with life's ceaseless currents."
ChatGPT as if it was Yoda

1.3 Approach

Meta-Reinforcement Learning (meta-RL) is a subfield of RL that aims at developing agents that can learn to solve new tasks with minimal interactions. Taking aside regret considerations, an intelligent agent should adapt to a new task by following the two phases: i) task understanding, also called exploration, where the agent collects information about the dynamics and rewards of the environment, ii) task solving, or exploitation, where the agent tries to maximize the expected cumulative reward based on the knowledge acquired during exploration. When the number of environmental interactions is limited, agents must naturally trade-off exploration and exploitation. There exists a large diversity of Meta-RL approach families, some consider learning how to tackle both phases separately (also called decoupling) whereas some do not. These choices can be largely influenced by the considered evaluation metric/setting which defines what adaptation should look like; an agent can be evaluated in a zero-shot manner, on a larger number of interactions or even being given a budget of

non-penalized exploration steps. In this manuscript, we will focus our attention on Meta-RL methods that explicitly consider task understanding.

The present research explores two ways to learn adaptable agents:

1. model-free: prepare agents for a distribution of task arriving at test time, through adaptation mechanisms that let agents learn at test-time.

2. model-based: build predictive dynamics models that are both robust, accurate and learn from minimal environment interactions.

1.4 Organisation of the manuscript

The present manuscript is structured into two distinct parts.

The first part is dedicated to methods that focus on the learning of adaptability mechanisms (i) by addressing the following challenges of meta-RL:

(a) When to switch exploration for exploitation?

(b) How to learn efficient exploration strategies that bring as much information as possible?

(c) How can the exploitation phase best utilize the exploration information?

We explore various algorithmic approaches, evaluation settings, and the impact of prior knowledge availability on the task distribution.

We start with a background chapter Chapter 2 to equip the reader with the necessary knowledge to grasp the concepts and principles that underpin Meta-RL. In particular, we examine variations between different meta-RL algorithms and aim to provide insights into their respective strengths and limitations.

In Chapter 3, we address (a) and (c) in the setting of "online adaptation" where the agent needs to adapt to unseen tasks in a single episode. It is given prior knowledge on the test tasks under the form of a set of training tasks. This scenario requires the agent to strike a balance between understanding the tasks and maximizing rewards. We train a recurrent agent that embeds the rolling trajectory into a representation that is well-prepared to generalize to test tasks.

In Chapter 4, we investigate (b) learning exploration strategies that maximize coverage of the state space and prepare the agent to be readily fine-tunable on the test tasks. The agent never observes extrinsic rewards during training and its adaptation is

evaluated on a set of downstream tasks. In practice, we tackle (c) by first imitating the behaviors that led to the highest rewards during exploration then fine-tuning using RL.

In the second part of the manuscript, we explore learning better predictive models under a small data regime. Leveraging those with planning algorithms, as in model-based reinforcement learning, limits the need for explicitly learning adaptation strategies. As NNs are bad approximators in small data regimes, they do not lend themselves to meta-RL. This part will explore the use of symbolic regression (SR) to find accurate and interpretable (in the form of small symbolic expressions) predictive models as they have been shown to be more robust to overfitting. Note that learning better world models is a way to tackle the challenge of trajectory representation described in (c). We will cover a set of methods that tackle the task of SR using recent advances in Transformer architectures.

In Chapter 5, we start with a background section that introduces formally SR and covers the related work. We introduce a class of SR algorithms called Deep Generative Symbolic Regression (DGSR) and show how they fit in an englobing framework for SR learning.

In Chapter 6, we explain how to train Transformers with SL to tackle SR on procedurally-generated synthetic datasets and evaluate the model's performance on standard SR benchmarks. We show the model achieves competitive performance for order of magnitudes less inference time.

In Chapter 7, we enable a pre-trained Transformer to learn from unsupervised datasets by augmenting it with search (Monte-Carlo Tree Search) at train and/or test time.

In Chapter 8, we propose to explicitly represent the dynamic and reward models (englobed under the name of world model) using SR and applying a planning algorithm on the learned models. We show in toy environments that the symbolic representation largely outperforms neural networks approaches.

Finally, we conclude in Chapter 9 by discussing potential future research directions and areas that warrant further exploration, fostering ongoing advancements in the field. Particularly, i) how to relax the asssumptions of human interventions in methods introduced in Part I, ii) the prospects of having interpretable symbolic world models, particularly how they can help improve meta-RL and iii) avenues to improve deep generative symbolic regression methods.

List of publications

Publications

- Pierre-Alexandre Kamienny, Guillaume Lample, Sylvain Lamprier, Marco Virgolin. **Deep Generative Symbolic Regression with Monte-Carlo-Tree-Search**. In *International Conference on Machine Learning* (ICML), 2023 (presented in Chapter 7)
- Tommaso Bendinelli, Luca Biggio, Pierre-Alexandre Kamienny. **Controllable Deep Symbolic Regression**. In *International Conference on Machine Learning* (ICML), 2023 (presented in Chapter G)
- Pierre-Alexandre Kamienny*, Stéphane d'Ascoli*, Guillaume Lample, Francois Charton. **End-to-end symbolic regression with transformers**. In *Neural Information Processing Systems* (NeurIPS), 2022 (presented in Chapter 6)
- Stéphane d'Ascoli*, Pierre-Alexandre Kamienny*, Guillaume Lample, Francois Charton. **Deep symbolic regression for recurrence prediction**. In *International Conference on Machine Learning* (ICML), 2022 (presented in Chapter 6)
- Pierre-Alexandre Kamienny*, Jean Tarbouriech*, Sylvain Lamprier, Alessandro Lazaric, Ludovic Denoyer. **Direct then Diffuse: Incremental Unsupervised Skill Discovery for State Covering and Goal Reaching**. In *International Conference on Learning Representations* (ICLR), 2022 (discussed in Chapter 4)

Workshop papers

- Pierre-Alexandre Kamienny, Sylvain Lamprier. **Symbolic Model-Based Reinforcement Learning**. In *Neural Information Processing Systems, AI For Science workshop* (NeurIPS), 2022 (presented in Chapter 8)
- Pierre-Alexandre Kamienny, Matteo Pirotta, Alessandro Lazaric, Thibault Lavril, Nicolas Usunier, Ludovic Denoyer. **Meta-Reinforcement Learning With Informed Policy Regularization**. In *International Conference on Machine Learning, BIG workshop* (ICML), 2023 (presented in Chapter 3)

Works not mentioned in this thesis

- Bei Peng, Tabish Rashid, Christian Schroeder de Witt, Pierre-Alexandre Kamienny, Philip Torr, Wendelin Böhmer, Shimon Whiteson. **FACMAC: Factored**

multi-agent centralised policy gradients. In *Neural Information Processing Systems* (NeurIPS), 2021

- Pierre-Alexandre Kamienny, Kai Arulkumaran, Feryal Behbahani, Wendelin Boehmer, Shimon Whiteson. **Privileged information dropout in reinforcement learning**. In *International Conference on Learning Representations, Beyond Tabula Rasa in RL workshop* (ICLR), 2020

Part I

Learning model-free exploration strategies

The progress in the field of learning agents has been significantly propelled by the rise and widespread adoption of Reinforcement Learning (RL) as a unifed framework, accompanied by its own array of methodologies and algorithms. Meta-Reinforcement Learning (meta-RL) is one such research direction that has emerged, focusing on endowing agents with the ability to learn during their deployment.

In this part of the manuscript, we provide a comprehensive overview of the RL formalism, followed by a detailed exploration of the distinctive characteristics and nuances of meta-RL. We then focus on the development of model-free adaptation mechanisms that enable learning at the time of testing. We particularly investigate two settings: one where the agent has access to a set of training tasks, and the other where it operates without observing any extrinsic rewards prior to test-time. In both scenarios, this is realized differently: in Chapter 3, IMPORT devises implicit exploration strategies that extract information about the task, and in Chapter 4 UPSIDE employs explicit strategies aimed at maximizing state-space coverage.

Chapter 2

Background

2.1 The Reinforcement Learning problem

2.1.1 Definitions

Markov Decision Processes (MDP). The agent's sequential decision-making process is usually modeled by a MDP defined as a tuple $(\mathcal{S}, \mathcal{A}, P, P_0, r, \gamma)$. At every time step t, the agent observes the *state* of the environment, $s_t \in \mathcal{S}$ and has too choose an *action* $a_t \in \mathcal{A}$. $\mathcal{S}$ and $\mathcal{A}$ are respectively the state and action spaces, i.e. the set of possible values they can take, which can either be continuous or discrete. This action has two effects: i) it changes the state of the environment from s_t to s_{t+1} and ii) it rewards the agent with a reward r_t. This change can be stochastic and modelled by the *transition* distribution $P(s_{t+1}|s_t, a_t)$. As the state environment is considered *Markovian*, the transition only conditions on the current state and not earlier states $s_{<t}$. The subscript $<$ in $x_{<t}$ (resp. $\leq$) denote the tuple of all values of x before t (resp. including the current). The reward $r_t : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ can also be stochastic and depends on (s_t, a_t, s_{t+1}). We denote trajectory $\tau_{\leq t}$, the sequence of state-action-reward tuples up to the step t, i.e. $\tau_{\leq t} = (s_0, a_0, r_0, s_1, \ldots, s_{t+1})$. In this manuscript, we will always assume an *episodic* setting. The agent starts an episode in state $s_0 \sim P_0(s_0)$ and the interaction with the environment stops after a fixed horizon T or when the agent reaches a *terminal* state. In what follows, if t is omitted from τ, it means the trajectory express $\tau_{\leq T}$, i.e. a full episode trajectory.

We call the *return function* $R(\tau)$ the discounted sum of rewards collected by the environment on trajectory

$$R(\tau) = \sum_{t \leq T-1} \gamma^t r_t(s_t, a_t, s_{t+1}) \tag{2.1}$$

The *discount factor* $\gamma \in [0, 1]$ accounts for the fact the agent should care more about immediate rewards than long-term ones. The smaller γ, the greedier the agent will aspire to be.

We represent by a policy $\pi(a_t|s_t) : \mathcal{S} \times \mathcal{A} \to [0, 1]$ the action probability distribution in a given state s_t. We can describe the dynamics of the agent-environment interaction as follows (omitting the stochasticity of rewards).

$$P(\tau) = P_0(s_0)\Pi_{0 \leq t \leq T-1}\pi(a_t|s_t)P(s_{t+1}|a_t, s_t) \tag{2.2}$$

In the manuscript, we will use the word *task* to qualify a MDP. Analogoulsy to SL where tasks can be datasets from different distribution, i.e. different datasets in computer vision or natural language processing, tasks will differ in what follows with their transition and reward functions. Furthermore, transition and reward functions are not known, so the agent gain information about them via trial and error through environment interactions.

Partially Observable Markov Decision Processes. A POMDP relaxes the assumption of MDPs that the full state s_t is observed by the agent. The agent observes $o_t \in \mathcal{O}$ from the *observation* distribution $O(o_t|s_t)$. To behave optimally, the agent must now condition on the entire trajectory $\tau_{\leq t} = (o_{\leq t}, a_{<t}, r_{<t})$. We will assume in what follows that the environment is fully observable, however approaches to solve MDPs can be extended to POMDPs by replacing the policy conditioning on the state s_t by $\tau_{\leq t}$.

Value functions. The state value $V^\pi(s)$ of a policy π is the expected return of the policy when starting in state $s \in \mathcal{S}$, i.e.

$$V^\pi(s) = \mathbb{E}[R(\tau)|s_0 = s] \tag{2.3}$$

where the expectation is with respect to the random trajectory generated by executing π starting from $s \in \mathcal{S}$. Similarly, the state-action value $Q^\pi(s, a)$ of a policy is the

expected return of the policy when starting in state $s \in \mathcal{S}$ and taking action $a \in \mathcal{A}$

$$Q^{\pi}(s,a) = \mathbb{E}[R(\tau)|s_0 = s, a_0 = a] \tag{2.4}$$

The advantage function A is modified version of Q with lower variance obtained by removing the state value as the baseline.

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{2.5}$$

The value functions at different states are connected with the Bellman equation:

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{a\sim\pi(.|s),s'\sim P(.|s,a)}[R(s,a,s') + V^{\pi}(s')] \\ Q^{\pi}(s,a) &= \mathbb{E}_{a'\sim\pi(.|s'),s'\sim P(.|s,a)}[R(s,a,s') + Q^{\pi}(s',a')] \end{aligned} \tag{2.6}$$

Goal. The objective of the agent is to find an *optimal* policy, i.e.

$$\pi^* = \arg\max_{\pi} J(\pi) \text{ where } J(\pi) = \mathbb{E}_{P(\tau)}[R(\tau)] \tag{2.7}$$

In terms of value functions, a policy π^* is said to be optimal, if it exists, if it maximizes the value functions $V^{\pi}(s)$ and $Q^{\pi}(s,a)$ in every state and action, i.e.

$$\begin{aligned} &\forall s \in \mathcal{S}, V^*(s) = V^{\pi^*}(s) = \max_{\pi} V^{\pi}(s) \\ &\forall (s,a) \in \mathcal{S} \times \mathcal{A}, Q^*(s,a) = Q^{\pi^*}(s,a) = \max_{\pi} Q^{\pi}(s,a) \end{aligned} \tag{2.8}$$

2.1.2 Challenges.

Exploration/Exploitation trade-off. The inherent RL challenge primarily arises from the inherent uncertainty associated with the environment. In situations where the environment is fully understood, the pursuit of optimal behavior essentially becomes a task of dynamic programming or planning. However, in the absence of such complete knowledge, the learner encounters a dilemma between the imperative to explore the environment to unravel its underlying structure (e.g., reward distribution and state transitions) and the necessity to exploit the information accumulated thus far. This trade-off is further complicated by the interdependency between the agent's current actions and future information. Consequently, it becomes imperative to strike an appropriate balance between exploration and exploitation to facilitate efficient learning.

Learning from a bad reward signal. In its original formulation, agents perceive upon executing actions within the environment in the MDP framework. The rewards, commonly qualified as *extrinsic* or *external*, are generated by the environment itself. In general, it means humans have carefully designed the reward function to capture the desired behavioral patterns of an optimal agent across various states. However, the process of designing these rewards can be impractical for some tasks. For instance, consider goal-oriented tasks wherein the agent's objective is to reach a specific state. Using the L2-distance between states as a reward metric may lead to the emergence of sub-optimal agents, particularly when obstacles such as walls impede the agent's progress towards the goal. Alternatively, a reward formulation can be established using an indicator function that determines whether the agent has successfully attained the goal state. In this case, the reward design becomes trivial but relying on such *sparse* rewards often poses challenges for agents to effectively learn solely from extrinsic rewards alone [Sut84]. This issue is further exacerbated in long-horizon tasks, which demand the agent to execute an increasingly protracted and flawless sequence of actions before encountering a positive reward. The agent must then extract pertinent information from this reward signal and utilize it in the most efficient manner.

Unsupervised RL. Unsupervised RL encompasses the utilization of an intrinsic reward function, such as curiosity-driven exploration [Pat+17], to autonomously generate training signals. This approach facilitates the acquisition of a diverse range of behaviors that are not explicitly tied to specific tasks. Despite the recent surge in research efforts towards various methodologies for unsupervised RL, the problem at hand remains significantly under-constrained. The absence of extrinsic reward signals poses a considerable challenge in acquiring behaviors that possess practical utility.

2.2 Some algorithms

RL algorithms can be cast in two classes: model-free and model-based. In short, model-free algorithms learn a policy (directly or via value functions), whereas model-based explicitly model transition and reward function of the environment and leverage planning tools on the learnt environment model. In this section, we will mention a few model-free approaches that were used in Chapters 3 and 4 and leave a small introduction to the model-based approach in Chapter 8. Figure 2.1 showcases a non-exhaustive list of algorithms in modern RL.

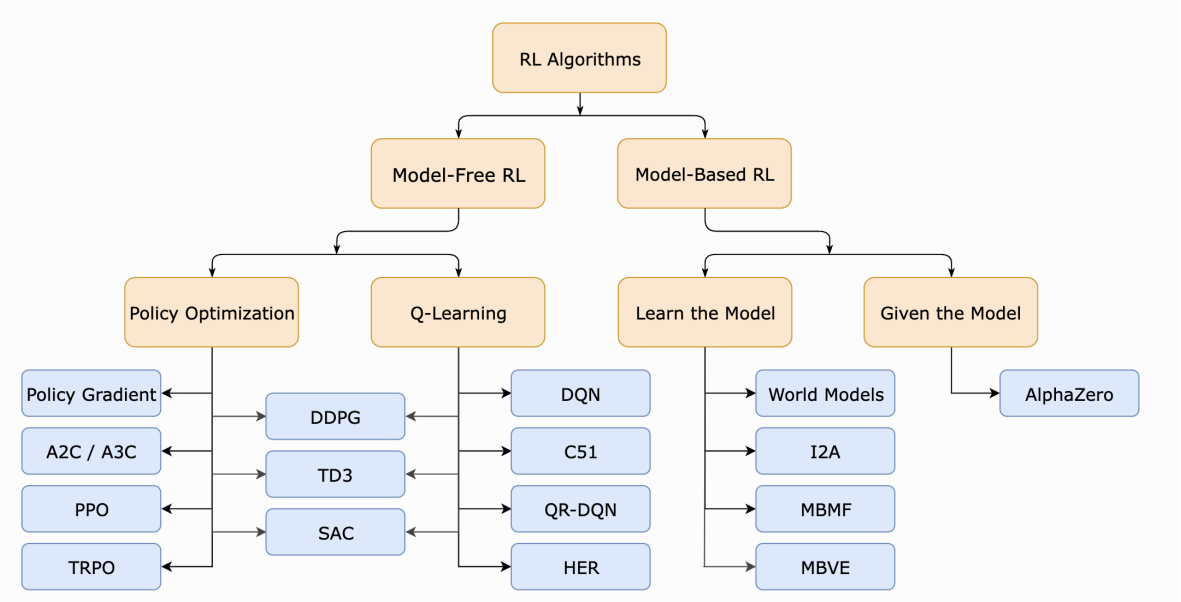


Figure 2.1 – Non-exhaustive list of modern RL algorithms from [Ach18]

Policy gradient. A policy π can be parametrized by a parameter θ_p and learned following the policy gradient $\nabla_{\theta_p} J(\pi_{\theta_p})$ with J defined in Equation (2.7). Using the policy gradient theorem [Sut+99], the gradient can be computed using:

$$\nabla_{\theta_p} J(\pi_{\theta_p}) = \mathbb{E}_\tau \Big[R(\tau) \nabla_{\theta_p} \log \pi_{\theta_p} \Big] \tag{2.9}$$

REINFORCE [Wil92] estimate the objective by using a Monte-Carlo estimate by rolling out the policy in the environment.

Actor-critic approaches. Actor-critic algorithms propose smaller variance estimates of returns Equation (2.9) by relying on value functions introduced in Section 2.1. We can learn a policy's value function V_{θ_v} by minimizing the error between the current value and a target value estimate.

$$\begin{aligned} \mathcal{L}^V &= \mathbb{E}_\tau [V_{\theta_v}(s_t) - V^{\text{target}}(s_t)]^2 \\ \mathcal{L}^Q &= \mathbb{E}_\tau [Q_{\theta_q}(s_t, a_t) - Q^{\text{target}}_{\theta_q}(s_t, a_t)]^2 \end{aligned} \tag{2.10}$$

For instance, Advantage Actor-Critic (A2C) [Mni+16] uses one-step Sarsa bootstrap as the target state-action value $Q^{\text{target}}_{\theta_q}(s_t, a_t) = r_t + (1 - d_t)\gamma Q^-(s_{t+1}, a_{t+1})$ where d_t is a boolean that indicates where the episode is finished after timestep t, Q^- is a target network updated slowly, either periodically or smoothly using Polyak averaging.

Proximal Policy Optimization (PPO) [Sch+17b] improves over A2C by adding a regularization term that penalizes the policy for deviating too much in between iterations. The overall objective is given by $\mathcal{L}^{PPO} = \mathbb{E}_\tau\Big[\min(A_{\theta_q}(s_t, a_t)c_t(\theta_p), A_{\theta_q}(s_t, a_t)clip(c_t, 1-\varepsilon, 1+\varepsilon)\Big]$, with $c_t = \frac{\pi_{\theta_p}(a_t|s_t)}{\pi^{old}_{\theta_p}(a_t|s_t)}$. The algorithms are on-policy algorithms which means examples considered to compute losses were generated by the current policy. They work for both discrete and continuous action spaces.

The following algorithms are off-policy and are designed for continuous action spaces. They store accumulated experience into a replay buffer $\mathcal{B}$ a collection of transition tuples (s, a, s', r, d). Deep Deterministic Policy Gradient (DDPG) learns a deterministic policy $\mu_{\theta_p}(s)$ that maximizes the state-action value function by differentiating through Q, i.e.:

$$\max_{\theta_p} \mathbb{E}_{s\in\mathcal{B}}\Big[Q_{\theta_q}(s, \mu_{\theta_p}(s))\Big] \tag{2.11}$$

Exploration is encouraged in DDPG by adding a Gaussian noise to actions. Twin-Delayed DDPG (TD3) [FHM18] improves DDPG by considering clipped Double-Q learning [Has10], delayed policy updates and target policy smoothing. Soft Actor-Critic (SAC) [Haa+18] uses stochastic policies where the variance is learnt. It places a higher emphasis on exploration with a policy entropy regularization term as well as a modified reward function that encourages to visit states with high policy-entropy.

2.3 Meta-Reinforcement Learning

2.3.1 Motivation

Within a standard supervised learning (SL) pipeline, the algorithm acquires knowledge by learning a predictor from a collection of training observations. The key challenge lies in the predictor's ability to identify abstract patterns from the training set and apply them effectively to new and unseen contexts. Given that the training set is limited and cannot encompass all potential inputs the model may encounter during deployment, the predictor must extrapolate from the available data in order to generate meaningful predictions. Two distinct forms of **generalization** emerge in this context: (a) in-distribution generalization and (b) out-of-distribution (OOD) generalization. These forms diverge in terms of their underlying assumptions regarding the generation of data.

In-distribution generalization pertains to evaluating the performance of the predictor on test data generated from the same data distribution as the training set [BM02; Vap06]. Conversely, out-of-distribution generalization focuses on evaluating the predictor's performance on test data generated from a different data distribution [Koh+21]. The disparity between these two scenarios is commonly referred to as distribution shift. While in-domain assumptions can provide provable guarantees on the generalization performance of specific classes of learning algorithms, these assumptions often do not align with the practical settings encountered in many real-world applications, such as the ones encounted in RL.

Deep Reinforcement Learning has been used to successfully train agents on a range of challenging environments such as Atari games [Mni+13; Bel+13; Hes+17] or continuous control [Pen+18; Sch+17a]. Nonetheless, in these problems, RL agents perform exploration strategies to discover the environment and implement algorithms to learn a policy that is tailored to solving a *single task*. Whenever the task changes, RL agents generalize poorly and the whole process of exploration and learning restarts from scratch. On the other hand, we expect an intelligent agent to fully master a *problem* when it is able to generalize from a few instances (tasks) and achieve the objective of the problem under many variations of the environment. For instance, children *know* how to ride a bike (i.e., the problem) when they can reach their destination irrespective of the specific bike they are riding, which requires to adapt to the weight of the bike, the friction of the brakes and tires, and the road conditions (i.e., the tasks).

The generalization problem is particularly apparent in the field of reinforcement learning (RL) due to two primary factors:

(a) Large state and action spaces: In cases where the state and action spaces are substantial, simple look-up tables are inadequate. Most RL applications involving real-world data necessitate in-domain generalization to account for the immense scale of these spaces.

(b) Non-stationarities: Non-stationarities manifest as the state distribution evolves during agent training, resulting in the agent encountering previously unseen state-action pairs. These non-stationarities commonly occur in non-stationary or multi-task environments characterized by changes in the underlying dynamics. For instance, if a robot is subject to an additional force, such as wind, it is unlikely to perform the desired task effectively unless it has been trained to be robust to such environmental changes.

Overcoming distribution shifts in generalization poses technical challenges that depend on the nature and magnitude of the shifts themselves. Consequently, endeavors focused on developing models resilient to distribution shifts typically necessitate explicit assumptions regarding the specific type of shift under consideration. In the following section, our investigation will concentrate on exploring out-of-domain generalization by examining the impact of changes in dynamics. Specifically, we will center our attention on scenarios where the agent has access to multiple training tasks and is evaluated on unseen but conceptually similar tasks.

In the context of SL, the assessment of generalization involves training the model using a set of training observations and subsequently measuring the error on the test data. Similarly, in reinforcement learning (RL), generalization can be evaluated by quantifying the cumulative reward obtained when deploying an agent on test tasks. However, this definition is limiting as it solely captures the correlation in behavior across different tasks. It fails to consider the agent's ability to learn and update its behavior through interactions. If there exists a *transfer* of knowledge from training tasks to test tasks, the agent should exhibit accelerated learning capabilities. In what follows, we develop on the subfield of meta-RL; during evaluation, i.e. deployment on test tasks, the agent can still update its behavior by considering the knowledge acquired from interactions on the test task. We will explain the evaluation metric as well as learning strategies in greater detail in Section 2.3.2.

2.3.2 Formalism

Definition The fundamental principle underlying this component is Meta-Learning, alternatively referred to as *"Learning to Learn"*. Its primary goal is to enhance the efficiency of acquiring new tasks, a concept that can be traced back to 1987 [Sch87] and that experienced renewed interest and extensive investigation thanks to the advancements in NNs [HVP21; Hos+20]. Its application to RL, Meta-Reinforcement Learning (meta-RL), are methods that *learn how to reinforcement learn*, i.e. prepare agents to best adapt to unseen tasks. Meta-RL is closely related to *Multi-task Reinforcement Learning* [Wil+07; Teh+17], *Transfer Learning* [TS11; Laz12]. During preparation, i.e. learning on train tasks, sample efficiency is not important, however it is important during testing.

Evaluation of Meta-RL agents We usually frame the evaluation of a meta-RL agent in a single task $\mathcal{M}$ in two phases:

1. an *exploration* phase in which the agent can freely explore the environment efficiently to gather information about the task. Since the exploration interactions are not included in the return, the agent can focus on collecting information about the task (dynamics and rewards) rather than strictly maximizing the reward, including risk-taking actions.

2. an *execution* phase in which the agent needs to drive its behaviour to maximize the return, using the knowledge acquired during exploration.

The length of the exploration phase is what we call the exploration **budget** (also called *burn-in* in [Bec+23]). In what follows, we will consider the length of both phases to be measures in number of episodes, K_{exp} and K_{exec} respectively for the exploration and execution phases. The agent interacts with the same environment during trials [Dua+16], defined by K_{exp} exploration episodes then K_{exec} execution episodes.

Under this framing, the objective can be formulated as follows:

$$J(\pi, \mathcal{M}, K_{\text{exp}}, K_{\text{exec}}) = \mathbb{E}_{\tau_{\text{exp}}^{1:K_{\text{exp}}} \sim \pi} \mathbb{E}_{\tau_{\text{exec}}^{1:K_{\text{exec}}} \sim \pi(.|\tau_{\text{exp}}^{1:K_{\text{exp}}})} R_{\mathcal{M}}(\tau_{\text{exec}}^{1:K_{\text{exec}}}) \tag{2.12}$$

where $R_{\mathcal{M}}(\tau_{\text{exec}}^{1:K_{\text{exec}}}) = \frac{1}{K_{\text{exec}}} \sum_{1 \leq k \leq K_{\text{exec}}} R_{\mathcal{M}}(\tau_{\text{exec}}^{k})$. $R_{\mathcal{M}}$ is the return function of environment $\mathcal{M}$ as defined in Section 2.1. The agent's conditioning on the exploration trajectories $\pi(.|\tau_{\text{exp}}^{1:K_{\text{exp}}})$ suggests that it needs to leverage the information acquired in the exploration phase as well as possible to maximize its return.

One typically evaluated meta-RL agents on a distribution of tasks $q(\mathcal{M})$ using the per-task objective Equation (2.12):

$$J(\pi, K_{\text{exp}}, K_{\text{exec}}) = \mathbb{E}_{\mathcal{M} \sim q(\mathcal{M})} J(\pi, \mathcal{M}, K_{\text{exp}}, K_{\text{exec}}) \tag{2.13}$$

Example algorithms. The meta-RL algorithm typically has access to a set of J_{train} training tasks $\mathcal{M}_{\text{train}} = \{\mathcal{M}_j\}_{1 \leq j \leq J_{\text{train}}}$ and is evaluated on $\mathcal{M}_{\text{test}} = \{\mathcal{M}_j\}_{1 \leq j \leq J_{\text{test}}}$. Approaches to meta-RL differ with the way i) they model the policy π, ii) the policy update rule and iii) the reward signal. We will summarize a non-exhaustive list of recent meta-RL approaches that rely on NNs.

(i) During the exploration and execution phases, the behavior can be represented by a single policy [Hau+18; Rak+19; Hum+19; Kam+20], usually parametrized by a NN, or two policies that decouple explicitly the way the agent should act in exploration π_{exp} and execution π_{exec} [Liu+21]. Though the agent is decoupled into two policies,

π_{exp} and π_{exec} can share information, e.g. through parameters or experience. In the latter, the execution policy π_{exec} leverages the information acquired in the exploration phase by conditioning on the exploration trajectories $\pi(.|\tau_{\text{exp}}^{1:K_{\text{exp}}})$. Decoupling behaviors comes at the cost of learning when to switch in cases when K_{exp} exploration episodes are not enough to fully grasp the task.

(ii) Two policy update rules have been considered. Model-Agnostic Meta-Learning (MAML) [Nag+18; Al-+17; Rag+19] approaches pre-train the model to provide the best weight initialization for test-time gradients. Recurrent approaches [Hum+19; Kam+22; Liu+21] generally treat the adaptation problem as a POMDP problem, and therefore accumulate knowledge on the task using recurrent NNs where the state of the RNN encodes the experiences.

(iii) Strictly maximizing the meta-RL objective (Equation (2.12)), i.e. considering the return on execution episodes only, has the drawback of introducing sparse rewards during exploration. Different surrogate objectives have been developed to provide more supervision to the exploration policy, for the purpose of better sample efficiency and often at the cost of theoretical optimality. Considering extrinsic rewards during the exploration phase as in RL2 [Dua+16] is suboptimal and privileges actions that maximize rewards rather than information gathering actions. Other works have considered using intrinsic rewards. They range curiosity-driven [Pat+17], novelty-driven [Nai+18; Pon+20; CLS22], goal-driven [Suk+17], diversity-driven [Eys+19; Kam+21] rewards. When the agent has access to a task descriptor at train time, it can learn to infer it from interactions; for instance, [Liu+21] introduces an intrinsic reward that quantify the information gain. [Hum+19; Kam+20] use auxiliary losses to learn task representations using $\mathcal{M}_{\text{train}}$.

Parametrized MDPs. In this manuscript, we will consider $q(\mathcal{M})$ to be a distribution over tasks (MDPs) in which the structure of the MDP is determined by a parameter with only a few degrees of freedom. We consider state and action spaces to be the same. Formally, tasks are thus fully determined by their parameter μ and induce variations in part of environments. We define in what follows a μ-MDP as $\mathcal{M}_\mu = (\mathcal{S}, \mathcal{A}, P^\mu, P_0^\mu, r^\mu, \gamma)$. As μ is not observed, solving the multi-task objective Equation (2.13) accounts for solving a partially observable MDP (POMDP), where the hidden variable is the descriptor μ of the μ-MDP.

2.3.3 Opening to Chapter 3 and Chapter 4

In Chapter 3, we will consider the setting of *online adaptation* where a trial consists of a single episode of execution, i.e. $K_{\text{exec}} = 1$ and $K_{\text{exp}} = 0$. The agent will have access to a set of training tasks which it can clearly identify by $\{\mu_j\}_{j \leq J_{\text{train}}}$. This setting has been largely studied in the literature of task identification.

In Chapter 4, we learn exploration strategies that will be help the agent adapt to test-time tasks. We study the setting where the task distribution $q(M)$ consists of a single dynamic function but multiple reward functions. In that case, $\mathcal{M}_\mu = (\mathcal{S}, \mathcal{A}, P, P_0, r^\mu, \gamma)$. During training, no extrinsic reward will be observed and agents will be evaluated by their adaptation to downstream tasks at test-time with a large K_{exec} and $K_{\text{exp}} = 1$.

Chapter 3

Online adaptation with Informed Policy Regularization

3.1 Introduction

Meta-reinforcement learning aims at finding a policy able to generalize to new tasks. When facing a new task, the policy must then balance *exploration* (or probing), i.e. identify its particular characteristics or alternatively reduce the uncertainty about the current task, and *exploitation* to maximize the cumulative reward of the task.

We consider the *online adaptation* setting where the agent needs to trade-off between the two types of behaviour within a **single episode** of the same task. The online adaptation setting is a special case of a partially observed markov decision problem, where the unobserved variables are the descriptors of the current task. It is thus possible to rely on recurrent neural networks (RNNs) [Bak01; Hee+15], since they can theoretically represent optimal policies in POMDPs if given enough capacity. Unfortunately, the training of RNN policies has often prohibitive sample complexity and it may converge to suboptimal local minima. Even though policies based on recurrent neural networks can be used in this setting by training them on multiple environments, they often fail to model this trade-off, or solve it at a very high computational cost. To overcome this drawback, efficient online adaptation methods can leverage *privileged knowledge* in the form of a task descriptor at training time. The main approach is to pair an exploration strategy with the training of *informed policies*, i.e. policies taking the description of the current task as input.

In this chapter, we introduce IMPORT (*InforMed POlicy RegularizaTion*), a novel policy architecture for efficient online adaptation that combines the rich expressivity of RNNs with the efficient learning of informed policies. At train time, a shared policy head receives as input the current observation, together with either a (learned) embedding of the current task, or the hidden state of an RNN such that the informed policy and the RNN policy are learned simultaneously. At test time, the hidden state of the RNN replaces the task embedding, and the agent acts without having access to the current task. This leads to several advantages: **1**) IMPORT benefits from informed policy to speed up learning; **2**) it avoids to reconstruct features of the task descriptor that are irrelevant for learning; and as a consequence, **3**) it adapts faster to unknown environments, showing better generalization capabilities.

We evaluate IMPORT against the main approaches to online adaptation on environments that require sophisticated exploration/exploitation strategies. We confirm that Thompson-Sampling **??** (TS) suffers from its limited expressivity, and show that the policy regularization of IMPORT significantly speeds up learning compared to Task Inference (`TI`) [Hum+19]. Moreover, the learnt task embeddings of IMPORT make it robust to irrelevant or minimally informative task descriptors, and able to generalize when learning on few training tasks.

3.2 Setting

Let $\mathcal{M}$ be the space of possible tasks. Each $\mu \in \mathcal{M}$ is associated to an episodic μ-MDP $M_\mu = (\mathcal{S}, \mathcal{A}, p^\mu, r^\mu, \gamma)$ whose dynamics p^μ and rewards r^μ are task dependent, while state and action spaces are shared across tasks and γ is the discount factor. The descriptor μ can be a simple id ($\mu \in \mathbb{N}$) or a set of parameters ($\mu \in \mathbb{R}^d$).

When the reward function and the transition probabilities are unknown, RL agents need to devise a strategy that balances exploration to gather information about the system and exploitation to maximize the cumulative reward. Such a strategy can be defined as the solution of a partially observable MDP (POMDP), where the hidden variable is the descriptor μ of the MDP. Given a trajectory $\tau_t = (s_1, a_1, r_1, \ldots, s_{t-1}, a_{t-1}, r_{t-1}, s_t)$, a POMDP policy $\pi(a_t|\tau_t)$ maps the trajectory to actions. In particular, the optimal policy in a POMDP is a history-dependent policy that uses τ_t to construct a belief state b_t, which describes the uncertainty about the task at hand, and then maps it to the action that maximizes the expected sum of rewards [KLC98]. In this case, maximizing the rewards may require taking explorative

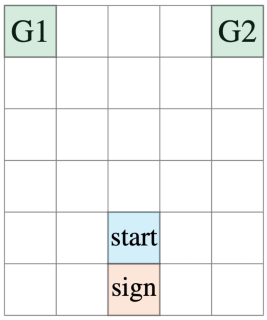


Figure 3.1 – An environment with two tasks: the goal location ($G1$ or $G2$) changes at each episode. The sign reveals the location of the goal. Optimal informed policies are shortest paths from *start* to either $G1$ or $G2$, which never visit the sign. Thompson sampling cannot represent the optimal exploration/exploitation policy (go to the sign first) since going to the sign is not feasible by any informed policy.

actions that improve the belief state enough so that future actions are more effective in collecting reward.

The task is sampled at the beginning of an episode from a distribution $q(\mu)$. After training, the agent returns a policy $\pi(a_t|\tau_t)$ that aims at maximizing the average performance across tasks generated from q, i.e.,

$$\mathbb{E}_{\mu\sim q(\mu)}\left[\sum_{t=1}^{|\tau|}\gamma^{t-1}r_t^{\mu}\,\middle|\,\pi\right] \tag{3.1}$$

where the expectation is taken over a full-episode trajectory τ and task distribution q, and $|\tau|$ is the length of the trajectory.

The objective is then to find an architecture for π that is able to express strategies that perform the best according to Eq. 3.1 and, at the same time, can be efficiently learned even for moderately short training phases.

At training time, we assume the agent has unrestricted access to the task descriptor μ. Access to such a task descriptor during training is a common assumption in the multi-task literature and captures a large variety of concrete problems. It can be of two types: i) a vector of features corresponding to (physical) parameters of the environment/agent (for instance, such features maybe available in robotics, or when learning on a simulator) [YLT18; Meh+19; Tob+17]. ii) It can be a single task identifier (i.e an integer) which is a less restrictive assumption [CYZ01; Hum+19] and corresponds to different concrete problems: learning in a set of M training levels

in a video game, learning to control M different robots or learning to interact with M different users.

3.3 Related Work and Contributions

In this section, we review how the online adaptation setting has been tackled in the literature. The main approaches are depicted in Fig. 3.2. We first compare the different methods in terms of expressiveness, and whether they leverage the efficient learning of informed policies. We then discuss learning task embeddings and how the various methods deal with unknown or irrelevant task descriptors. The last subsection summarizes our contributions.

Evaluation of RL agent in Meta-Reinforcement Learning. The *online adaptation* evaluation setting is standard in the Meta-RL literature [Yu+17; Hum+19] but is not the only way to evaluate agents on unseen tasks in the meta-RL literature. Indeed, several works have considered that given a new task, an agent is given an amount of "free" interactions episodes or steps to perform system identification, then is evaluated on the cumulative reward on one [BYM19; Rak+19] or several execution episodes [Liu+20]. This is different to what we study here where the agent has to identify the task to solve and solved it within one episode, the reward of the agent being considered during all these steps.

Online Adaptation with Deep RL. In the previous section we mentioned that the optimal strategy corresponds to the solution of the associated POMDP. Given a belief state b_t as a sufficient statistic of the history τ_t, POMDP policies takes the form $\pi(a_t|\tau_t) = \pi(a_t|s_t, b_t)$. While it is impractical to compute the exact belief state even for toy discrete problems, approximations can be learnt using Recurrent Neural Networks (RNNs) [Bak01; Hee+15]. RNN-based policies are trained to maximize the cumulative reward and do not leverage task descriptors at train time. While this class of policies can represent rich exploratory strategies, their large training complexity makes them impractical.

In order to reduce the training complexity of RNN policies, existing strategies have constrained the set of possible exploratory behaviors by leveraging privileged information about the task. Probe-Then-Exploit (PTE) [ZPG19] works in two phases. First, it executes a pure exploratory policy with the objective of identifying the underlying task

μ, i.e maximizing the likelihood of the task, then runs the optimal policy associated to the estimated task. Both the probing and the informed policies are learned using task descriptors, leading to a much more efficient training process. PTE has two main limitations. First, similarly to explore-then-commit approaches in bandits [GLK16], the exploration can be suboptimal because it is not reward-driven: valuable time is wasted to estimate unnecessary information. Second, the switch between probing and exploiting is hard to tune and problem-dependent.

Thompson Sampling (TS) [Tho33] leverages randomization to mix exploration and exploitation. Similarly to the belief state of an RNN policy, TS maintains a distribution over task descriptors that represents the uncertainty on the current task given τ_t. The policy samples a task from the posterior and executes the corresponding informed policy for several steps. Training is limited to learning informed policies together with a maximum likelihood estimator to map trajectories to distributions over tasks. This strategy proved successful in a variety of problems [CL11; OR17]. However, TS cannot represent certain probing policies such as the one of Figure 3.1, because it is constrained to executing informed policies; indeed informed policies behave optimally with respect to the assumed current, which might be too different from the exploration strategy needed to understand the task. Another drawback of TS approaches is that the re-sampling frequency needs to be carefully tuned.

The Task Inference (TaskInference) approach [Hum+19] is a RNN trained to simultaneously learn a good policy and predict the task descriptor μ. Denoting by $m : H \to Z$ the mapping from histories to a latent representation of the belief state ($Z \subseteq \mathbb{R}^d$), the policy $\pi(a_t|z_t)$ selects the action based on the representation $z_t = m(\tau_t)$ constructed by the RNN. During training, z_t is also used to predict the task descriptor μ, using the *task-identification* module $g : Z \to \mathcal{M}$. The overall objective is:

$$\mathbb{E}\Big[\sum_{t=1}^{|\tau|} \gamma^{t-1} r_t^{\mu} \Big| \pi\Big] + \beta \mathbb{E}\Big[\sum_{t=1}^{|\tau|} \ell(\mu, g(z_t)) \Big| \pi\Big] \tag{3.2}$$

where $\ell(\mu, g(z_t))$ is the log-likelihood of μ under distribution $g(z_t)$. The auxiliary loss is meant to structure the memory of the RNN m rather than be an additional reward for the policy, so training is done by ignoring the effect of m on π when computing the gradient of the auxiliary loss with respect to m. [Hum+19] proposed two variants, AuxTask and `TI`, described in Fig. 3.2 (b) and (c). In `TI` (contrary to AuxTask), the gradient of the policy sub-network is not backpropagated through the RNN (the dashed green arrow in Fig. 3.2 (c), and the policy subnetwork receives the original state features as additional input; the main objective of stopping gradients is to make

training more stable with TI. For both AuxTask and TI, the training of π is purely reward-driven, so they do not suffer from the suboptimality of PTE/TS. However, in contrast to PTE/TS, they do not leverage the smaller sample complexity of training informed policies, and the auxiliary loss is defined over the whole value of μ while only some dimensions may be relevant to solve the task.

In the non-stationary setting, only a few models have been proposed, mainly based on the MAML algorithm. For instance, [Nag+18] combines MAML with model-based RL by meta-learning a transition model that helps an MPC controller predicting the action sequence to take. The method does not make use of the value of μ at train time and is specific to MPC controllers.

Learning Task Embeddings While in principle the minimal requirement for the approaches above is access to *task identifiers*, i.e. one-hot encodings of the task, these approaches are sensitive to the encoding on task descriptions, and prior knowledge on them. In particular, irrelevant variables have a significant impact on PTE approaches since the probing policy aims at identifying the task. For instance, an agent might waste time reconstructing the full μ when only part of μ is needed to act optimally w.r.t the reward. Moreover, TS, TI and AuxTask are guided by a prior distribution over μ that has to be chosen by hand to fit the ground-truth distribution of tasks. [Rak+19] proposed to use a factored Gaussian distribution over transitions as a task embedding architecture rather than a RNN.

Several approaches have been proposed to learn task embeddings [Gup+18; Rak+19; Zin+19; Hau+18]. The usual approach is to train embeddings of task identifiers jointly with the policies. [Hum+19] mentions using TI with task embeddings, but the embeddings are pre-trained separately, which requires either additional interactions with the environment or expert traces. Nonetheless, we show in our experiments that TI can be used with task descriptors, considering task prediction as a multiclass classification problem.

Summary of the contributions As for RNN/TI, IMPORT learns an RNN policy to maximize cumulative reward, with no decoupling between probing and exploitation. As such, our approach does not suffer from scheduling difficulties instrinsic to PTE/TS approaches. On the other hand, similarly to PTE/TS and contrarily to RNN/TI, IMPORT leverages the fast training of informed policies through a joint training of an RNN and an informed policy. In addition, IMPORT does not rely on probabilistic

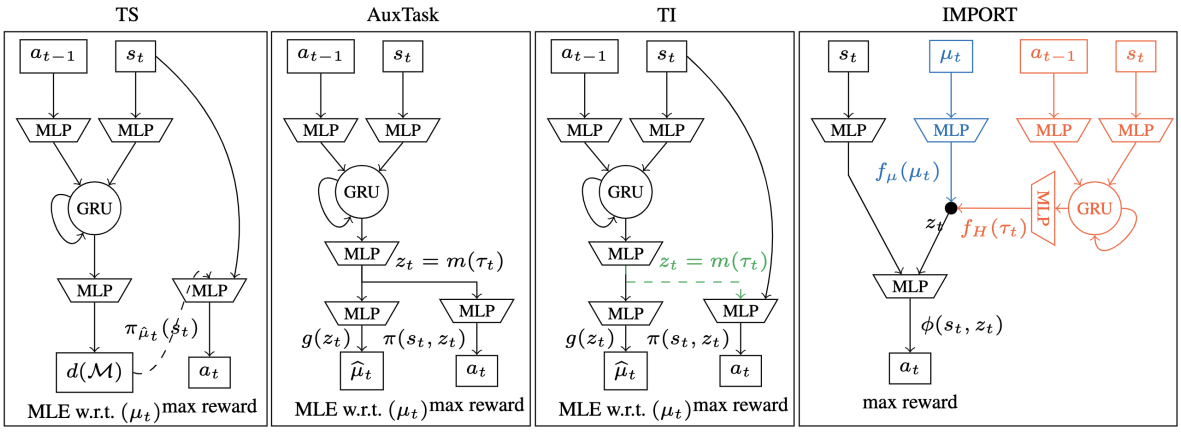


Figure 3.2 – Representation of the different architectures. IMPORT is composed of two models sharing parameters: The (black+blue) architecture is the informed policy π_μ optimized through **(B)** while the (black+red) architecture is the history-based policy π_H (used at test time) trained through **(A)**+**(C)**.

models of task descriptors. Learning task embeddings makes the approach robust to irrelevant task descriptors contrary to TaskInference, makes IMPORT applicable when only task identifiers are available and able to better generalize when few training tasks are available.

3.4 Method

In this section, we describe the main components of the IMPORT model (described in Fig. 3.2), as well as the online optimization procedure and an additional auxiliary loss to further speed-up learning.

Our approach leverages the knowledge of the task descriptor μ and informed policies to construct a latent representation of the task that is *purely reward driven*. Since μ is unknown at testing time, we use this informed representation to train a predictor based on a recurrent neural network. To leverage the efficiency of informed policies even in this phase, we propose an architecture *sharing parameters* between the informed policy and the final policy such that the final policy will benefit from parameters learned with privileged information. The idea is to constrain the final policy to stay close to the informed policy while allowing it to perform probing actions when needed to effectively reduce the uncertainty about the task. We call this approach InforMed POlicy RegularizaTion (IMPORT).

Algorithm 3.1: IMPORT Training

```
Initialize σ, ω, θ randomly
for k = 1, ..., K do
    if k is odd then
        Collect M transitions following π_H
        Update σ, ω and the parameters of the value function of (A) based on
         objective (A) + (C)
    else
        Collect M transitions following π_μ
        Update σ, θ, ω and the parameters of the value function of (B) based on
         objective (B) + (C)
```

Formally, we denote by $\pi_\mu(a_t|s_t,\mu)$ and $\pi_H(a_t|\tau_t)$ the informed policy and the history-dependent (RNN) policy that is used at test time. The informed policy $\pi_\mu = \phi \circ f_\mu$ is the functional composition of f_μ and ϕ, where $f_\mu : \mathcal{M} \to Z$ projects μ in a latent space $Z \subseteq \mathbb{R}^k$ and $\phi : \mathcal{S} \times Z \to \mathcal{A}$ selects the action based on the latent representation. The idea is that $f_\mu(\mu)$ captures the relevant information contained in μ while ignoring dimensions that are not relevant for learning the optimal policy. This behavior is obtained by training π_μ directly to maximize the task reward r^μ.

While π_μ leverages the knowledge of μ at training time, π_H acts based on the sole history. To encourage π_H to behave like the informed policy while preserving the ability to probe, π_H and π_μ share ϕ, the mapping from latent representations to actions. We thus define as $\pi_H = \phi \circ f_H$ where $f_H : \mathcal{H} \to Z$ encodes the history into the latent space. By sharing the policy head ϕ, the approximate belief state constructed by the RNN is mapped to the same latent space as μ. When the uncertainty about the task is small, π_H then benefits from the joint training with π_μ.

More precisely, let θ, ω, σ the parameters of ϕ, f_H and f_μ respectively, so that $\pi_\mu^{\sigma\theta}(a_t|s_t,\mu) = \phi^\theta \circ f_\mu^\sigma = \phi^\theta(a_t|s_t, f_\mu^\sigma(\mu))$ and $\pi_H^{\omega\theta}(a_t|\tau_t) = \phi^\theta \circ f_H^\omega = \phi^\theta(a_t|s_t, f_H^\omega(\tau_t))$. The goal of IMPORT is to maximize over θ, ω, σ the objective function defined in Eq. 3.3.

$$\underbrace{\mathbb{E}\left[\sum_{t=1}^{|\tau|}\gamma^{t-1}r_t^\mu \,\middle|\, \pi_H^{\omega,\theta}\right]}_{\textbf{(A)}} + \underbrace{\mathbb{E}\left[\sum_{t=1}^{|\tau|}\gamma^{t-1}r_t^\mu \,\middle|\, \pi_\mu^{\sigma,\theta}\right]}_{\textbf{(B)}} \quad \underbrace{-\beta\mathbb{E}\left[\sum_{t=1}^{|\tau|} D\Big(f_\mu(\mu), f_H(\tau_t)\Big)\right]}_{\textbf{(C)}} \tag{3.3}$$

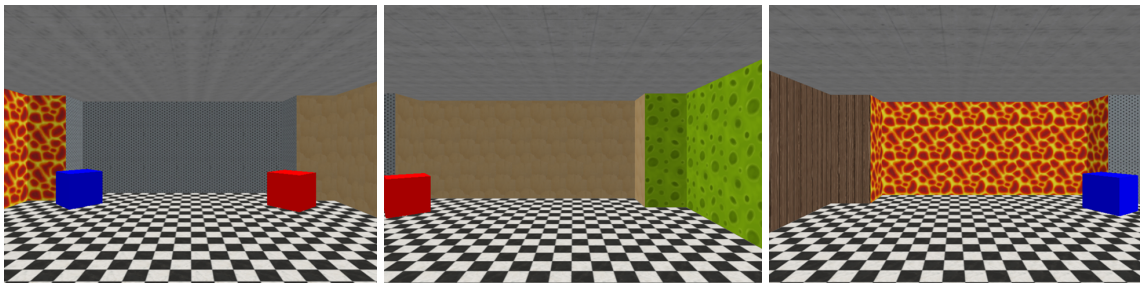


Figure 3.3 – Maze 3D. The goal is either located at the blue or the red box. When the back wall (i.e. not observed in the leftmost image) has a wooden texture, the correct goal is the blue box, whereas if the texture is green, the red box is the goal.

Speeding Up the Learning. The optimization of (**B**) in Eq. 3.3 produces a reward-driven latent representation of the task through f_μ. In order to encourage the history-based policy to predict a task embedding close to the one predicted by the informed policy, we augment the objective with an auxiliary loss (**C**) weighted by $\beta > 0$. D is the squared 2-norm in our experiments. Note that because we treat the objective (**C**) as an auxiliary loss, only the average gradient of D with respect to f_H is backpropagated, ignoring the effect of f_H on π_μ, in order to avoid f_H to influence the informed policy behavior. The expectation of (**C**) is optimized over trajectories generated using $\pi_H^{\omega,\theta}$ and $\pi_\mu^{\sigma,\theta}$, respectively used to compute (**A**) and (**B**).

Optimization. IMPORT is trained using Advantage Actor Critic (A2C) [Mni+16] with generalized advantage estimation (GAE) [Sch+15]. There are two value functions[1], one for each objective (**A**) and (**B**). The algorithm is summarized in Alg. 3.1. Each iteration collects a batch of M transitions using either π_H or π_μ.[2] If the batch is sampled according to π_H, we update with A2C-GAE the parameters of the policy ω and θ according to both objectives (**A**) and (**C**), as well as the parameters of the value function associated to objective (**A**). If the batch is sampled according to π_μ, we update with A2C-GAE the parameters of the policy σ and θ according to both objectives (**B**) and (**C**), as well as the parameters of the value function associated to objective (**B**).

	$N = 10$	$N = 20$	$N = 100$	$N = 10$	$N = 20$	$N = 100$
RNN	73.4(4.8)	92.9(1.3)	87.5(0.2)			
	Using μ at train time			Using task identifier at train time		
IMPORT	**94.4(0.7)**	**94.8(0.8)**	**95.3(0.4)**	**92.8(0.8)**	**95.5(0.4)**	95.1(1.0)
AuxTask	91.0(0.7)	92.0(1.9)	92.6(0.7)	90.5(1.8)	91.2(1.7)	94.3(0.7)
TI	91.5(0.6)	94.4(0.4)	94.6(0.3)	90.8(0.5)	90.7(1.2)	**97.0(0.2)**
TS	88.7(4.2)	87.3(2.5)	91.3(1.7)	85.9(2.4)	90.1(1.3)	91.0(0.5)

Table 3.1 – CartPole with different number N of training tasks. Note that RNN does not μ at train time.

	$S = 1$	$S = 3$	$S = 5$	$S = 1$	$S = 3$	$S = 5$
Size of μ:	5	60	150	5	60	150
RNN	74.0(5.8)	77.0(1.1)	66.9(0.7)			
	Using μ			Using task identifier		
IMPORT	**89.2(0.1)**	**79.4(0.6)**	**72.7(0.1)**	**89.1(0.1)**	**79.7(0.1)**	**74.0(0.3)**
AuxTask	84.7(1.1)	76.2(2.0)	66.9(1.2)	87.7(0.8)	78.0(0.4)	72.4(0.5)
TI	78.7(0.9)	76.5(0.3)	68.3(0.8)	85.3(0.8)	77.5(1.1)	70.9(0.2)
TS	85.9(0.5)	64.1(0.6)	60.7(0.1)	84.1(2.5)	65.7(0.3)	60.2(1.0)

Table 3.2 – Result over Tabular-MDP with S states and $A = 5$ actions, trained over $N = 100$ tasks.

3.5 Experiments

We first present the environments.

CartPole & Acrobot [Bro+16b] The task descriptor μ represents parameters of the physical system, e.g., the weight of the cart, the size of the pole, etc. The dimension of μ is 5 for Cartpole and 7 for Acrobot. The entries of μ are normalized in $[-1, 1]$ and sampled uniformly.

The environments that follow provide basic comparison points where the optimal exploration-exploitation trade-off is relatively straightforward, since the dynamics can be inferred from a few actions.

[1]In our implementation, the value network is shared and takes as an input either $f\mu(\mu)$ or $f_H(\tau_t)$.

[2]In practice, data collection is multithreaded. We collect 20 transitions per thread with 24 to 64 threads depending on the environment, based on available GPU memory

Bandit. We consider a standard Bernoulli multi-armed bandit problem with K arms. The vector $\mu \in \mathbb{R}^K$ denotes the probability of success of the independent Bernoulli distributions. Each dimension of μ is sampled uniformly between 0 and 0.5, the best arm is randomly selected and associated to a probability of 0.9. An episode is 100 arm pulls. At every timestep the agent is allowed to pull an arm in $[\![1, K]\!]$ and observes the resulting reward. Although relatively simple, this environment assesses the ability of algorithms to learn nontrivial probing/exploitation strategies.

Tabular MDP. We consider a finite MDP with $|\mathcal{S}|$ states and $\mathcal{A}|$ actions such that the transition matrix is sampled from a flat Dirichlet distribution, and the reward function is sampled from a uniform distribution in $[0, 1]$ as in [Dua+16]. In that case, μ is the concatenation of the transition and the reward functions, resulting in a vector of size $S^2A + SA$. This environment is much more challenging as μ is high-dimensional, there is nearly complete uncertainty on the task at hand and each task is a reinforcement learning problem. .

Maze 3D. We consider a 3D version of the toy problem depicted in Fig. 3.1, implemented using gym-miniworld [Che18]. It has three discrete actions (*forward*, *left*, *right*) and the objective is to reach one of the two possible goals (see Figure 3.3 in appendix), resulting in a reward of 1 (resp. -1) when the correct (resp. wrong) goal is reached. The episode terminates when the agent touches a box or after 100 steps. The agent always starts at a random position, with a random orientation. The information about which goal to reach at each episode is encoded by the use of two different textures on the wall located at the opposite side of the maze w.r.t. the goals. This domain allows to evaluate the models when observations are high dimensional ($3 \times 60 \times 60$ RGB images). The maximum episode length is 100 on CartPole, Bandit, Tabular-MDP and Maze3D, and 500 on Acrobot. To evaluate the ability of IMPORT and the baselines to deal with different types of task descriptors μ, we also perform experiments on CartPole and Tabular-MDP in the setting where μ is only a task identifier (i.e., a one-hot vector representing the index of the training task) which is a very weak supervision available at train time.

We compare to previously discussed baselines. First, a vanilla RNN policy [Hee+15] using GRUs that never uses μ. Second, we compare to TS, `TI` and AuxTask, with μ only observed at train time, similarly to IMPORT. For TS, at train time, the policy conditions on the true μ, whereas at test time, the policy conditions on an estimated $\widehat{\mu}$

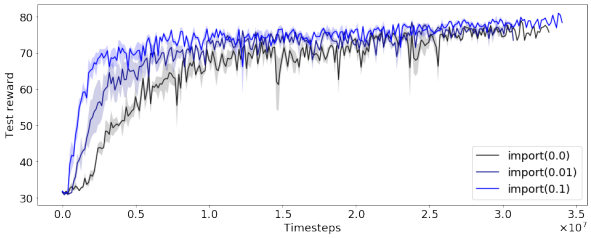
resampled from the posterior every k steps where $k \in \{1, 5, 10, 20\}$. On bandits, UCB [Aue02] with tuned exploration parameters is our topline.

Implementation details Contrarily to IMPORT, TS, `TI` and AuxTask are based on maximizing the log-likelihood of μ. When using informative task descriptors (i.e. a vector of real values), the log-likelihood uses a Gaussian distribution with learnt mean and diagonal covariance matrix. For the bandit setting, we have also performed experiments using a beta distribution which may be more relevant for this type of problem. When using task identifiers, a multinomial distribution is used. All approaches are trained using A2C with Generalized Advantage Estimation [Mni+16; Sch+15]. The precise values of the hyper-parameters and architectures are given in Appendix A.2.2.

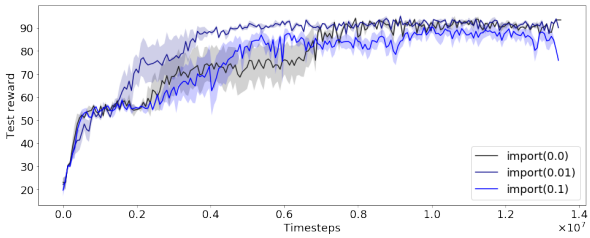
All approaches use similar network architectures with the same number of hidden layers and units.

Evaluation The meta-learning scenario is implemented by sampling N training tasks, N validation tasks and $10,000$ test tasks with no overlap between task sets (except in Maze3D where there is only two possible tasks). Each sampled training task is given a unique identifier. Each model is trained on the training tasks, and the best model is selected on the validation tasks. We report the performance on the test tasks, averaged over three trials with different random seeds, corresponding to different sets of train/validation/test tasks. Training uses a discount factor, but for validation and test, we compute the undiscounted cumulative reward on the validation/test tasks. The learning curves show test reward as a function of the environment steps. They are the average of the three curves associated to the best validation model of each of the three seeds used to generate different tasks sets.

Overall performances. IMPORT performs better than its competitors in almost all the settings. For instance, on CartPole with 10 tasks (see Table 3.1), our model reaches 94.4 reward while `TI` reaches only 91.5. Qualitatively similar results are found on Acrobot (Table A.2 in Chapter A), as well as on Bandit with 20 arms (Table 3.3), even though AuxTask performs best with only 10 arms. IMPORT particularly shines when μ encodes complex information, as on Tabular-MDP (see Table 3.2) where it outperforms all baselines in all settings. By varying the number of training tasks on CartPole and Acrobot, we also show that IMPORT's advantage over the baselines is larger with fewer training tasks. In all our experiments, as expected, the vanilla RNN performs worse than the other algorithms.

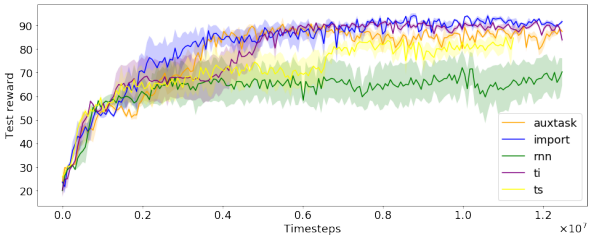


(**a**) Bandits with $K = 10, N = 100$

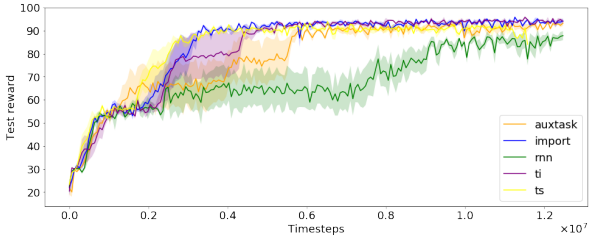


(**b**) CartPole with $N = 10$

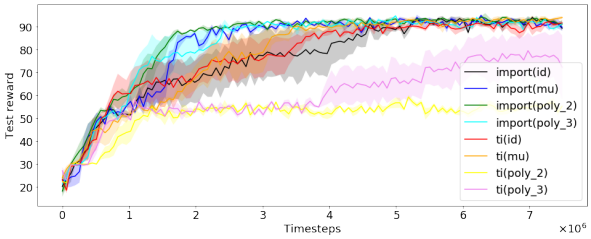
Figure 3.4 – Test performance of IMPORT for different values of β from Eq. 3.3



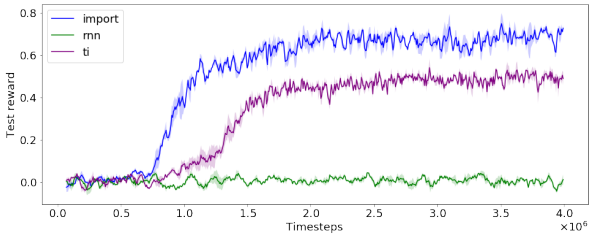
(**a**) CartPole $N = 10$



(**b**) CartPole $N = 100$



(**c**) Effect of transforming μ (CartPole, $N = 20$).



(**d**) Maze 3D.

Figure 3.5 – Learning curves on CartPole (a and b) and Maze3D (d) test tasks. Figure (c) studies the impact of the structure of the task descriptor on the performances of `TI` and IMPORT in CartPole.

Sample Efficiency. Figure 3.5 shows the convergence curves on CartPole with 10 and 100 training tasks and are representative of what we obtain on other environments (see Appendix). IMPORT tends to converge faster than the baselines. We also observe a positive effect of using the auxiliary loss ($\beta > 0$) on sample efficiency, in particular with few training tasks. Note that using the auxiliary loss is particularly efficient in environments where the final policy tends to behave like the informed on.

Influence of μ. The experiments with uninformative μ (i.e., task identifiers) reported in Table 3.1 and 3.2 for CartPole and Tabular-MDP respectively show that the methods are effective even when the task descriptors do not include any prior knowledge. In the two cases, IMPORT can use these tasks descriptors to generalize well. Moreover, experimental results on CartPole (Fig. 3.5) and Tabular MDP (Fig. A.10) suggest that when μ is a vector of features (and not a task identifier only) ,

	$K = 10$	$K = 20$
IMPORT	77.5(0.2)	**56.6(0.1)**
AuxTask (Gaussian)	78.7(0.4)	50.5(1.6)
AuxTask (Beta)	78.2(0.7)	37.1(0.6)
RNN	73.6(0.7)	32.1(1.2)
TI (Gaussian)	73.7(1.6)	41.4(2.4)
TI (Beta)	**79.5(0.1)**	53.3(2.4)
TS (Gaussian)	50.4(0.4)	38.8(2.0)
TS (Beta)	41.3(1.5)	36.3(1.1)
UCB	78.5(0.3)	68.2(0.4)

Table 3.3 – Bandits performance for $K = 10$ and $K = 20$ arms, with $N = 100$ training tasks.

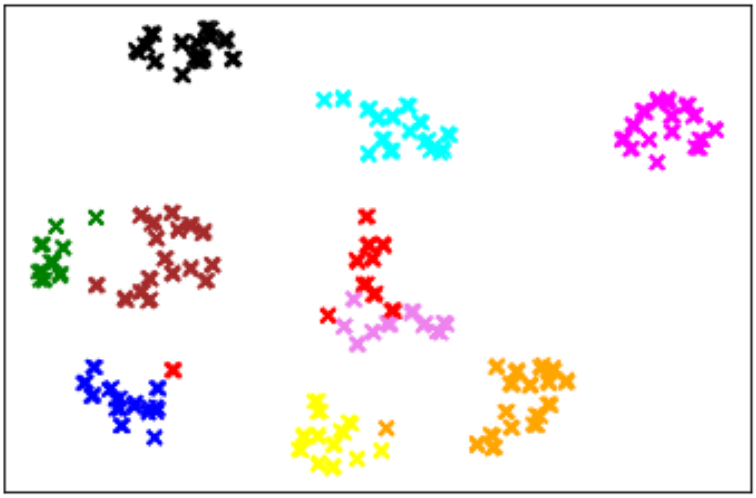


Figure 3.6 – Task embeddings learnt on Bandit (10 arms). Colors indicate the best arm.

it improves sample efficiency but does not change the final performance. This can be explained by the fact that informed policies are faster to learn with features in μ since, in that case, μ is capturing similarities between tasks. Equivalent performance of IMPORT on both types of task descriptors is observed and shows that our method can deal with different (rich and weak) task descriptors.

We further analyze the impact of the encoding of μ on the models, by using non-linear projections of the informative μ to change the shape of the prior knowledge. Figure 3.5c shows the learning curves of TI and IMPORT on CartPole with task identifiers, the original μ and polynomial expansions of μ of order 2 and 3, resulting in 21 and 56 features. IMPORT's task embedding approach is robust to the encoding of μ, while TI's log-likelihood approach underperforms with the polynomial transformation.

Task embeddings. To have a qualitative assessment of the task embedding learnt by IMPORT, we consider a bandit problem with 10 arms and embedding dimension 16. Figure 3.6 shows the clusters of task embeddings obtained with t-SNE [MH08b]. Each cluster maps to an optimal arm, showing that IMPORT structures the embedding space based on the relevant information. In addition, we have studied the influence of the β hyperparameter from Eq. 3.3 (in Fig. 3.4 and Section A.4). It shows that the auxiliary loss helps to speed-up the learning process, but is not necessary to achieve great performance.

High dimensional input space. We show the learning curves on the Maze3D environment in Figure 3.5d. IMPORT is succeeding in 90% of cases (reward ≈ 0.8), while `TI` succeeds only in 70% of cases. This shows that IMPORT is even more effective with high-dimensional observations (here, pixels). IMPORT and `TI` benefit from knowing μ at train time, which allows them to rapidly identify that the wall texture behind the agent is informative, while the vanilla RNN struggles and reaches random goals. TS is not reported since this environment is a typical failure case as discussed in Fig.3.1.

Additional results. In Appendix A.3.1, we show that IMPORT outperforms `TI` by a larger margin when the task embedding dimension is small. We also show that IMPORT outperforms its competitors in dynamic environments, i.e., when the task changes during the episode.

3.6 Conclusion

We proposed a new policy architecture for meta reinforcement learrning. The IMPORT model is trained only on the reward objective, and leverages the informed policy to discover effective trade-offs between exploration and exploitation. It is thus able to learn better strategies than Thompson Sampling approaches, and faster than recurrent neural network policies and Task Inference approaches.

This work assumes access to a set of training tasks (including extrinsic rewards) as well as the ability to clearly identify which task the agent is facing at every episode. It assumes that the training tasks come from the same distribution $q(M)$ than the test tasks. In the next chapter, we will remove this assumption; the agent does not observe extrinsic rewards during training and will be evaluated on a set of test tasks that share the same dynamics. We use this unsupervised training phase to learn exploration strategies that cover efficiently the state-space.

Chapter 4

Learning exploration strategies without rewards

4.1 Introduction

In Unsupervised RL (URL), the agent first interacts with the environment without any extrinsic reward signal. Afterward, the agent leverages the experience accumulated during the unsupervised learning phase to efficiently solve a variety of downstream tasks defined on the same environment. This approach is particularly effective in problems such as navigation [see e.g., BSK21] and robotics [see e.g., Pon+20] where the agent is required to readily solve a wide range of tasks (determined by new test reward functions) while the dynamics of environment remains fixed.

In this chapter, we focus on the unsupervised objective of discovering a set of skills that can be used to efficiently solve sparse-reward downstream tasks. In particular, we build on the insight that *mutual information* (MI) between the skills' latent variables and the states reached by them can formalize the dual objective of learning policies that both cover and navigate the environment efficiently. Indeed, maximizing MI has been shown to be a powerful approach for encouraging exploration in RL [Hou+16; MR15] and for unsupervised skill discovery [e.g., GRW16; Eys+19; Ach+18; Sha+20; Cam+20]. Nonetheless, learning policies that maximize MI is a challenging optimization problem. Several approximations have been proposed to simplify it at the cost of possibly deviating from the original objective of coverage and directedness (see Sect. 4.3 for a review of related work).

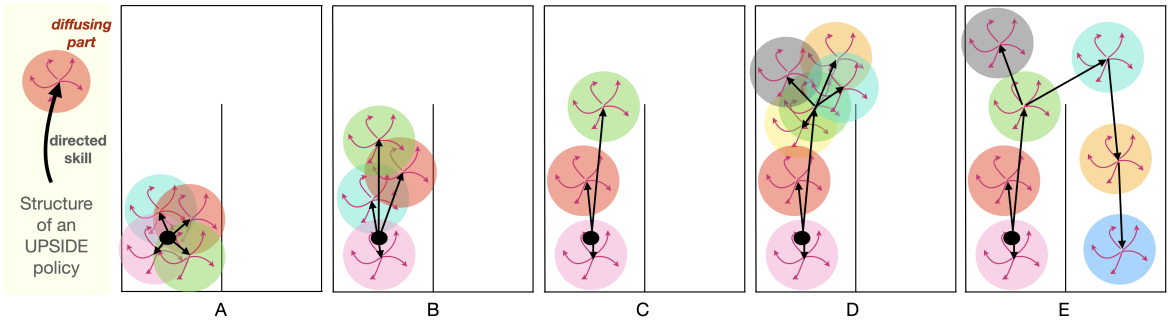


Figure 4.1 – Overview of UPSIDE. The black dot corresponds to the initial state. (A) A set of random policies is initialized, each policy being composed of a *directed* part called *skill* (illustrated as a black arrow) and a *diffusing* part (red arrows) which induces a local coverage (colored circles). (B) The skills are then updated to maximize the discriminability of the states reached by their corresponding diffusing part (Sect. 4.4.1). (C) The least discriminable policies are iteratively removed while the remaining policies are re-optimized. This is executed until the discriminability of each policy satisfies a given constraint (Sect. 4.4.2). In this example two policies are consolidated. (D) One of these policies is used as basis to add new policies, which are then optimized following the same procedure. For the "red" and "purple" policy, UPSIDE is not able to find sub-policies of sufficient quality and thus they are not expanded any further. (E) At the end of the process, UPSIDE has created a tree of policies covering the state space, with skills as edges and diffusing parts as nodes (Sect. 4.4.3).

We propose UPSIDE (*UnsuPervised Skills that dIrect then DiffusE*) to learn a set of policies that can be effectively used to cover the environment and solve goal-reaching downstream tasks. Our solution builds on the following components (Fig. 4.1):

- *Policy structure* (*Sect. 4.4.1, see Fig. 4.1* (A)). We consider policies composed of two parts: **1)** a *directed* part, referred to as the *skill*, that is trained to reach a specific region of the environment, and **2)** a *diffusing* part that induces a local coverage around the region attained by the first part. This structure favors coverage and directedness at the level of a single policy.
- *New constrained objective* (*Sect. 4.4.2, see Fig. 4.1* (B) *&* (C)). We then introduce a constrained optimization problem designed to maximize the number of policies under the constraint that the states reached by *each* of the diffusing parts are distinct enough (i.e., they satisfy a minimum level of discriminability). We prove that this problem can be cast as a lower bound to the original MI objective, thus preserving its coverage-directedness trade-off. UPSIDE solves it by *adaptively* adding or removing policies to a given initial set, without requiring any prior knowledge on a sensible number of policies.
- *Tree structure* (*Sect. 4.4.3, see Fig. 4.1* (D) *&* (E)). Leveraging the directed nature of the skills, UPSIDE effectively composes them to build longer and longer policies

organized in a tree structure. This overcomes the need of defining a suitable policy length in advance. Thus in UPSIDE we can consider short policies to make the optimization easier, while composing their skills along a growing tree structure to ensure an adaptive and thorough coverage of the environment.

The combination of these components allows UPSIDE to effectively adapt the number and the length of policies to the specific structure of the environment, while learning policies that ensure coverage and directedness. We study the effectiveness of UPSIDE and the impact of its components in hard-to-explore continuous navigation and control environments, where UPSIDE improves over existing baselines both in terms of exploration and learning performance.

4.2 Setting

We consider the URL setting where the agent interacts with a Markov decision process (MDP) M with state space $\mathcal{S}$, action space $\mathcal{A}$, dynamics $p(s'|s,a)$, and **no reward**. The agent starts each episode from a designated initial state $s_0 \in \mathcal{S}$.[1] Upon termination of the chosen policy, the agent is then reset to s_0. This setting is particularly challenging from an exploration point of view since the agent cannot rely on the initial distribution to cover the state space.

We recall the MI-based unsupervised skill discovery approach [see e.g., GRW16]. Denote by Z some (latent) variables on which the policies of length T are conditioned (we assume that Z is categorical for simplicity and because it is the most common case in practice). There are three optimization variables: **(i)** the cardinality of Z denoted by N_Z, i.e., the number of policies (we write $Z = \{1, \dots, N_Z\} = [N_Z]$), **(ii)** the parameters $\pi(z)$ of the policy indexed by $z \in Z$, and **(iii)** the policy sampling distribution ρ (i.e., $\rho(z)$ is the probability of sampling policy z at the beginning of the episode). Denote policy z's action distribution in state s by $\pi(\cdot|z,s)$ and the entropy function by $\mathcal{H}$. Let the variable S_T be the random (final) state induced by sampling a policy z from ρ and executing $\pi(z)$ from s_0 for an episode. Denote by $p_{\pi(z)}(s_T)$ the distribution over (final) states induced by executing policy z, by $p(z|s_T)$ the probability of z being the policy to induce (final) state s_T, and let $\overline{p}(s_T) = \sum_{z \in Z} \rho(z) p_{\pi(z)}(s_T)$. Maximizing the

[1] More generally, s_0 could be drawn from any distribution supported over a compact region.

MI between Z and S_T can be written as $\max_{N_Z,\rho,\pi} \mathcal{I}(S_T; Z)$, where

$$\begin{aligned}\mathcal{I}(S_T; Z) &= \mathcal{H}(S_T) - \mathcal{H}(S_T|Z) = -\sum_{s_T} \overline{p}(s_T) \log \overline{p}(s_T) + \sum_{z \in Z} \rho(z) \mathbb{E}_{s_T|z} \left[\log p_{\pi(z)}(s_T)\right] \\ &= \mathcal{H}(Z) - \mathcal{H}(Z|S_T) = -\sum_{z \in Z} \rho(z) \log \rho(z) + \sum_{z \in Z} \rho(z) \mathbb{E}_{s_T|z} \left[\log p(z|s_T)\right]\end{aligned} \tag{4.1}$$

where in the expectations $s_T|z \sim p_{\pi(z)}(s_T)$. In the first formulation, the entropy term over states captures the requirement that policies thoroughly cover the state space, while the second term measures the entropy over the states reached by each policy and thus promotes policies that have a directed behavior. Learning the optimal N_Z, ρ, and π to maximize eq. (4.1) is a challenging problem and several approximations have been proposed, e.g. [GRW16; Eys+19; Ach+18; Cam+20], based on the optimization of the following lower bound:

$$\mathcal{I}(S_T; Z) \geq \mathbb{E}_{z \sim \rho(z), \tau \sim \pi(z)} \left[\log q_\phi(z|s_T) - \log \rho(z)\right] \tag{4.2}$$

where we denote by $\tau \sim \pi(z)$ trajectories sampled from the policy indexed by z. As a result, each policy $\pi(z)$ can be trained with RL to maximize the intrinsic reward $r_z(s_T) := \log q_\phi(z|s_T) - \log \rho(z)$.

4.3 Related work

URL methods can be broadly categorized depending on how the experience of the unsupervised phase is summarized to solve downstream tasks in a zero- or few-shot manner. This includes model-free [Pon+20], model-based [Sek+20] and representation learning [e.g., Yar+21] methods that build a representative replay buffer to learn accurate estimates or low-dimensional representations. An alternative line of work focuses on discovering a set of skills in an unsupervised way. Our approach falls in this category, on which we now focus this section.

Skill discovery based on MI maximization was first proposed in `VIC` [GRW16], where the trajectories' final states are considered in the reverse form of eq. (4.1) and the policy parameters $\pi(z)$ and sampling distribution ρ are simultaneously learned (with a fixed number of skills N_Z). `DIAYN` [Eys+19] fixes a uniform ρ and weights skills with an action-entropy coefficient (i.e., it additionally minimizes the MI between actions and skills given the state) to push the skills away from each other. `DADS`

[Sha+20] learns skills that are not only diverse but also predictable by learned dynamics models, using a generative model over observations and optimizing a forward form of MI $\mathcal{I}(s'; z|s)$ between the next state s' and current skill z (with continuous latent) conditioned on the current state s. `EDL` [Cam+20] shows that existing skill discovery approaches can provide insufficient coverage and relies on a fixed distribution over states that is either provided by an oracle or learned. `SMM` [Lee+19b] uses the MI formalism to learn a policy whose state marginal distribution matches a target state distribution (e.g., uniform). Other MI-based skill discovery methods include Florensa et al. [FDA17], Hansen et al. [Han+19], Modhe et al. [Mod+20], and Xie et al. [Xie+21], and extensions in non-episodic settings [Xu+20; Lu+20].

While most skill discovery approaches consider a fixed number of policies, a curriculum with increasing N_Z is studied in Achiam et al. [Ach+18] and Aubret et al. [AMH20]. We consider a similar discriminability criterion in the constraint in (4.4) and show that it enables to maintain skills that can be readily composed along a tree structure, which can either increase or decrease the support of available skills depending on the region of the state space. Recently, Zhang et al. [ZYX21] propose a hierarchical RL method that discovers abstract skills while jointly learning a higher-level policy to maximize extrinsic reward. Our approach builds on a similar promise of composing skills instead of resetting to s_0 after each execution, yet we articulate the composition differently, by exploiting the direct-then-diffuse structure to ground skills to the state space instead of being abstract. Our connection of the latent Z to the state space S enables us to compose our skills so as to cover and navigate the state space in the absence of extrinsic reward. Hartikainen et al. [Har+20] perform unsupervised skill discovery by fitting a distance function; while their approach also includes a directed part and a diffusive part for exploration, it learns only a single directed policy and does not aim to cover the entire state space. Approaches such as `DISCERN` [War+19] and `Skew-Fit` [Pon+20] learn a goal-conditioned policy in an unsupervised way with an MI objective. As explained by Campos et al. [Cam+20], this can be interpreted as a skill discovery approach with latent $Z = S$, i.e., where each goal state can define a different skill. Conditioning on either goal states or abstract latent skills forms two extremes of the spectrum of unsupervised RL. As argued in Sect. 4.4.1, we target an intermediate approach of learning "cluster-conditioned" policies. Finally, an alternative approach to skill discovery builds on "spectral" properties of the dynamics of the MDP. This includes eigenoptions [MBB17; Mac+18] and covering options [Jin+19; Jin+20b], and the algorithm of Bagaria et al. [BSK21] that builds a discrete graph representation which learns and composes spectral skills.

	UPSIDE directed skill	UPSIDE diffusing part	VIC policy	DIAYN policy
state variable	S_{diff}	S_{diff}	S_T	S
$\mathcal{J}$	$\{T, \dots, T+H\}$	$\{T, \dots, T+H\}$	$\{T\}$	$\{1, ..., T\}$
(α, β)	$(1, 0)$	$(0, 1)$	$(1, 0)$	$(1, 1)$

Table 4.1 – Instantiation of eq. (4.3) for each part of an UPSIDE policy, and for VIC [GRW16] and DIAYN [Eys+19] policies.

4.4 Method

In this section we detail the three main components of UPSIDE, which is summarized in Sect. 4.4.4.

4.4.1 Decoupled Policy Structure of Direct-then-Diffuse

While the trade-off between coverage and directedness is determined by the MI objective, the amount of stochasticity of each policy (e.g., injected via a regularization on the entropy over the actions) has also a major impact on the effectiveness of the overall algorithm [Eys+19]. In fact, while randomness can promote broader coverage, a highly stochastic policy tends to induce a distribution $p_{\pi(z)}(s_T)$ over final states with high entropy, thus increasing $\mathcal{H}(S_T|Z)$ and losing in directedness. In UPSIDE, we define policies with a *decoupled structure* (see Fig. 4.1 (A)) composed of **a)** a *directed* part (of length T) that we refer to as *skill*, with low stochasticity and trained to reach a specific region of the environment and **b)** a *diffusing* part (of length H) with high stochasticity to promote local coverage of the states around the region reached by the skill.

Coherently with this structure, the state variable in the conditional entropy in Equation (4.1) becomes any state reached during the diffusing part (denote by S_{diff} the random variable) and not just the skill's terminal state. Following Section 4.2 we define an intrinsic reward $r_z(s) = \log q_\phi(z|s) - \log \rho(z)$ and the skill of policy z maximizes the cumulative reward over the states traversed by the diffusing part. Formally, we can conveniently define the objective function:

$$\max_{\pi(z)} \mathbb{E}_{\tau \sim \pi(z)} \Big[\sum_{t \in \mathcal{J}} \alpha \cdot r_z(s_t) + \beta \cdot \mathcal{H}(\pi(\cdot|z, s_t)) \Big], \tag{4.3}$$

where $\mathcal{J} = \{T, \dots, T+H\}$ and $\alpha = 1, \beta = 0$ (resp. $\alpha = 0, \beta = 1$) when optimizing for the skill (resp. diffusing part). In words, the skill is incentivized to bring the diffusing part to a discriminable region of the state space, while the diffusing part is optimized by a simple random walk policy (i.e., a stochastic policy with uniform distribution over actions) to promote local coverage around s_T.

Table 4.1 illustrates how UPSIDE's policies compare to other methods. Unlike VIC and similar to DIAYN, the diffusing parts of the policies tend to "push" the skills away so as to reach diverse regions of the environment. The combination of the directedness of the skills and local coverage of the diffusing parts thus ensures that the whole environment can be properly visited with $N_Z \ll |\mathcal{S}|$ policies.[2] Furthermore, the diffusing part can be seen as defining a *cluster of states* that represents the goal region of the directed skill. This is in contrast with DIAYN policies whose stochasticity may be spread over the whole trajectory. This allows us to "ground" the latent variable representations of the policies Z to specific regions of the environment (i.e., the clusters). As a result, maximizing the MI $\mathcal{I}(S_{\text{diff}}; Z)$ can be seen as learning a set of "cluster-conditioned" policies.

4.4.2 A Constrained Optimization Problem

In this section, we focus on how to optimize the number of policies N_Z and the policy sampling distribution $\rho(z)$. The standard practice for eq. (4.1) is to preset a fixed number of policies N_Z and to fix the distribution ρ to be uniform (see e.g., [Eys+19; Bau+21; Str+21]). However, using a uniform ρ over a fixed number of policies may be highly suboptimal, in particular when N_Z is not carefully tuned. In App. B.1.2 we give a simple example and a theoretical argument on how the MI can be ensured to increase by removing skills with low discriminability when ρ is uniform. Motivated by this observation, in UPSIDE we focus on *maximizing the number of policies that are sufficiently discriminable*. We fix the sampling distribution ρ to be uniform over N policies and define the following constrained optimization problem

$$\max_{N \geq 1} N \quad \text{s.t.} \quad g(N) \geq \log \eta, \quad \text{where} \quad g(N) := \max_{\pi, \phi} \min_{z \in [N]} \mathbb{E}_{s_{\text{diff}}} [\log q_\phi(z | s_{\text{diff}})] \quad (\mathcal{P}_\eta) \tag{4.4}$$

[2] eq. (4.1) is maximized by setting $N_Z = |\mathcal{S}|$ (i.e., $\max_Y \mathcal{I}(X, Y) = \mathcal{I}(X, X) = \mathcal{H}(X)$), where each z represents a goal-conditioned policy reaching a different state, which implies having as many policies as states, thus making the learning particularly challenging.

where $q_\phi(z|s_{\text{diff}})$ denotes the probability of z being the policy traversing s_{diff} during its diffusing part according to the discriminator and $\eta \in (0, 1)$ defines a *minimum discriminability threshold*. By optimizing for (Equation (4.4)), UPSIDE *automatically adapts* the number of policies to promote coverage, while still guaranteeing that each policy reaches a distinct region of the environment. Alternatively, we can interpret Equation (4.4) under the lens of *clustering*: the aim is to find the largest number of clusters (i.e., the region reached by the directed skill and covered by its associated diffusing part) with a sufficient level of inter-cluster distance (i.e., discriminability) (see Fig. 4.1). The following lemma (proof in App. B.1.1) formally links the constrained problem $(\mathcal{P}_\eta)$ back to the original MI objective.

Lemma 1. *There exists a value $\eta^\dagger \in (0, 1)$ such that solving $(\mathcal{P}_{\eta^\dagger})$ is equivalent to maximizing a lower bound on the mutual information objective $\max_{N_Z,\rho,\pi,\phi} \mathcal{I}(S_{\text{diff}}; Z)$.*

Since $(\mathcal{P}_{\eta^\dagger})$ is a lower bound to the MI, optimizing it ensures that the algorithm does not deviate too much from the dual covering and directed behavior targeted by MI maximization. Interestingly, Lem. 1 provides a rigorous justification for using a uniform sampling distribution *restricted to the η-discriminable policies*, which is in striking contrast with most of MI-based literature, where a uniform sampling distribution ρ is defined over the predefined number of policies.

In addition, our alternative objective $(\mathcal{P}_\eta)$ has the benefit of providing a simple *greedy* strategy to optimize the number of policies N. Indeed, the following lemma (proof in App. B.1.1) ensures that starting with $N = 1$ (where $g(1) = 0$) and increasing it until the constraint $g(N) \geq \log \eta$ is violated is guaranteed to terminate with the optimal number of policies.

Lemma 2. *The function g is non-increasing in N.*

4.4.3 Composing Skills in a Growing Tree Structure

Both the original MI objective and our constrained formulation (4.4) depend on the initial state s_0 and on the length of each policy. Although these quantities are usually predefined and only appear implicitly in the equations, they have a crucial impact on the obtained behavior. In fact, resetting after each policy execution unavoidably restricts the coverage to a radius of at most $T + H$ steps around s_0. This may suggest to set T and H to large values. However, increasing T makes training the skills more challenging, while increasing H may not be sufficient to improve coverage.

Instead, we propose to "extend" the length of the policies through composition. We rely on the key insight that *the constraint in* (4.4) *guarantees that the directed skill of each η-discriminable policy reliably reaches a specific* (*and distinct*) *region of the environment and it is thus re-usable and amenable to composition*. We thus propose to chain the skills so as to reach further and further parts of the state space. Specifically, we build a growing tree, where the root node is a diffusing part around s_0, *the edges represent the skills, and the nodes represent the diffusing parts*. When a policy z is selected, the directed skills of its predecessor policies in the tree are executed first (see Fig. B.3 in App. B.2 for an illustration). Interestingly, this growing tree structure builds a curriculum on the episode lengths which grow as the sequence $(iT + H)_{i\geq 1}$, thus avoiding the need of prior knowledge on an adequate horizon of the downstream tasks.[3] Here this knowledge is replaced by T and H which are more environment-agnostic and task-agnostic choices as they rather have an impact on the size and shape of the learned tree (e.g., the smaller T and H the bigger the tree).

4.4.4 Implementation

We are now ready to introduce the `UPSIDE` algorithm, which provides a specific implementation of the components described before (see Fig. 4.1 for an illustration, Alg. algorithm 4.1 for a short pseudo-code and Alg. B.1 in App. B.2 for the detailed version). We first make approximations so that the constraint in eq. (4.4) is easier to estimate. We remove the logarithm from the constraint to have an estimation range of $[0, 1]$ and thus lower variance. [4] We also replace the expectation over s_{diff} with an empirical estimate $\widehat{q}^{\mathcal{B}}_{\phi}(z) = \frac{1}{|\mathcal{B}_z|}\sum_{s\in\mathcal{B}_z} q_\phi(z|s)$, where $\mathcal{B}_z$ denotes a small replay buffer, which we call *state buffer*, that contains states collected during a few rollouts by the diffusing part of π_z. In our experiments, we take $B = |\mathcal{B}_z| = 10H$. Integrating this in eq. (4.4) leads to

$$\max_{N\geq 1} N \quad \text{s.t.} \quad \max_{\pi,\phi} \min_{z\in[N]} \widehat{q}^{\mathcal{B}}_{\phi}(z) \geq \eta \tag{4.5}$$

[3] See e.g., the discussion in [MPR21] on the importance of properly choosing the training horizon in accordance with the downstream-task horizon the policy will eventually face.

[4] While Gregor et al. [GRW16] and Eysenbach et al. [Eys+19] employ rewards in the log domain, we find that mapping rewards into $[0, 1]$ works better in practice, as observed in [War+19; Bau+21].

where η is an hyper-parameter.[5] From Lem. 2, this optimization problem in N can be solved using the incremental policy addition or removal in Alg. 4.1 (lines 8 & 14), independently from the number of initial policies N.

Algorithm 4.1: `UPSIDE`

1 **Parameters**: Discriminability threshold $\eta \in (0, 1)$, branching factor N^{start}, $N^{\max}$.
2 **Initialize**: Tree $\mathcal{T}$ initialized as a root node 0, policies candidates $\mathcal{Q} = \{0\}$.
3 **while** $\mathcal{Q} \neq \emptyset$ **do** `// tree expansion`
4 Dequeue a policy $z \in \mathcal{Q}$ and create $N = N^{\text{start}}$ policies $\mathcal{C}(z)$.
5 PolicyLearning$(\mathcal{T}, \mathcal{C}(z))$.
6 **if** $\min_{z' \in \mathcal{C}(z)} \widehat{q}^{\mathcal{B}}_{\phi}(z') > \eta$ **then** `// Node addition`
7 **while** $\min_{z' \in \mathcal{C}(z)} \widehat{q}^{\mathcal{B}}_{\phi}(z') > \eta$ **and** $N < N^{\max}$ **do**
8 Increment $N = N + 1$ and add one policy to $\mathcal{C}(z)$.
9 PolicyLearning$(\mathcal{T}, \mathcal{C}(z))$.
10 **end**
11 **end**
12 **else** `// Node removal`
13 **while** $\min_{z' \in \mathcal{C}(z)} \widehat{q}^{\mathcal{B}}_{\phi}(z') < \eta$ **and** $N > 1$ **do**
14 Reduce $N = N - 1$ and remove least discriminable policy from $\mathcal{C}(z)$.
15 PolicyLearning$(\mathcal{T}, \mathcal{C}(z))$.
16 **end**
17 **end**
18 Add η-discriminable policies $\mathcal{C}(z)$ to $\mathcal{Q}$, and to $\mathcal{T}$ as nodes rooted at z.
19 **end**

We then integrate the optimization of Equation (4.5) into an adaptive tree expansion strategy that incrementally composes skills (Sect. 4.4.3). The tree is initialized with a root node corresponding to a policy only composed of the diffusing part around s_0. Then `UPSIDE` iteratively proceeds through the following phases: **(Expansion)** While policies/nodes can be expanded according to different ordering rules (e.g., a FIFO strategy), we rank them in descending order by their discriminability (i.e., $\widehat{q}^{\mathcal{B}}_{\phi}(z)$), thus favoring the expansion of policies that reach regions of the state space that are not too saturated. Given a candidate leaf z to expand from the tree, we introduce new policies by adding a set $\mathcal{C}(z)$ of $N = N^{\text{start}}$ nodes rooted at node z (line 5, see also steps (A) and (D) in Fig. 4.1). **(Policy learning)** The new policies are

[5] Ideally, we would set $\eta = \eta^{\dagger}$ from Lem. 1, however $\eta^{\dagger}$ is non-trivial to compute. A strategy may be to solve $(\mathcal{P}_{\eta'})$ for different values of η' and select the one that maximizes the MI lower bound $\mathbb{E}\left[\log q_{\phi}(z|s_{\text{diff}}) - \log \rho(z)\right]$. In our experiments we rather use the same predefined parameter of $\eta = 0.8$ which avoids computational overhead and performs well across all environments.

optimized in three steps (see App. B.2 for details on the PolicyLearning subroutine): **i)** sample states from the diffusing parts of the new policies sampled uniformly from $\mathcal{C}(z)$ (state buffers of consolidated policies in $\mathcal{T}$ are kept in memory), **ii)** update the discriminator and compute the discriminability $\widehat{q}^{\mathcal{B}}_{\phi}(z')$ of new policies $z' \in \mathcal{C}(z)$, **iii)** update the skills to optimize the reward (Sect. 4.4.1) computed using the discriminator (see step (B) in Fig. 4.1). (**Node adaptation**) Once the policies are trained, `UPSIDE` proceeds with optimizing N in a greedy fashion. If all the policies in $\mathcal{C}(z)$ have an (estimated) discriminability larger than η (lines 6-8), a new policy is tentatively added to $\mathcal{C}(z)$, the policy counter N is incremented, the *policy learning* step is restarted, and the algorithm keeps adding policies until the constraint is not met anymore or a maximum number of policies is attained. Conversely, if every policy in $\mathcal{C}(z)$ does not meet the discriminability constraint (lines 12-14), the one with lowest discriminability is removed from $\mathcal{C}(z)$, the *policy learning* step is restarted, and the algorithm keeps removing policies until all policies satisfy the constraint or no policy is left (see step (C) in Fig. 4.1). The resulting $\mathcal{C}(z)$ is added to the set of *consolidated* policies (line 18) and `UPSIDE` iteratively proceeds by selecting another node to expand until no node can be expanded (i.e., the *node adaptation* step terminates with $N = 0$ for all nodes) or a maximum number of environment iterations is met.

4.5 Experiments

Our experiments investigate the following questions: **i)** Can `UPSIDE` incrementally cover an unknown environment while preserving the directedness of its skills? **ii)** Following the unsupervised phase, how can `UPSIDE` be leveraged to solve sparse-reward goal-reaching downstream tasks? **iii)** What is the impact of the different components of `UPSIDE` on its performance?

We report results on navigation problems in continuous 2D mazes[6] and on continuous control problems [Bro+16a; TET12]: Ant, Half-Cheetah and Walker2d. We evaluate performance with the following tasks: **1)** "coverage" which evaluates the extent to which the state space has been covered during the unsupervised phase, and **2)** "unknown goal-reaching" whose objective is to find and reliably reach an unknown

[6]The agent observes its current position and its actions (in $[-1, +1]$) control its shift in x and y coordinates. We consider two topologies of mazes illustrated in Fig. 4.2 with size 50×50 such that exploration is non-trivial. The Bottleneck maze is a harder version of the one in Campos et al. [Cam+20, Fig. 1] whose size is only 10×10.

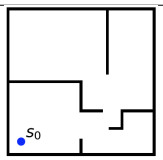
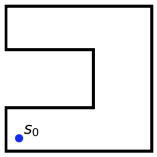
	 Bottleneck Maze		 U-Maze	
RANDOM	29.17	(±0.57)	23.33	(±0.57)
DIAYN-10	17.67	(±0.57)	14.67	(±0.42)
DIAYN-20	23.00	(±1.09)	16.67	(±1.10)
DIAYN-50	30.00	(±0.72)	25.33	(±1.03)
DIAYN-curr	18.00	(±0.82)	15.67	(±0.87)
DIAYN-hier	38.33	(±0.68)	49.67	(±0.57)
EDL-10	27.00	(±1.41)	32.00	(±1.19)
EDL-20	31.00	(±0.47)	46.00	(±0.82)
EDL-50	33.33	(±0.42)	52.33	(±1.23)
SMM-10	19.00	(±0.47)	14.00	(±0.54)
SMM-20	23.67	(±1.29)	14.00	(±0.27)
SMM-50	28.00	(±0.82)	25.00	(±1.52)
Flat UPSIDE-10	40.67	(±1.50)	43.33	(±2.57)
Flat UPSIDE-20	47.67	(±0.31)	55.67	(±1.03)
Flat UPSIDE-50	51.33	(±1.64)	57.33	(±0.31)
UPSIDE	**85.67**	(±1.93)	**71.33**	(±0.42)

Table 4.2 – Coverage on Bottleneck Maze and U-Maze: UPSIDE covers significantly more regions of the discretized state space than the other methods. The values represent the number of buckets that are reached, where the 50×50 space is discretized into 10 buckets per axis. To compare the global coverage of methods (and to be fair w.r.t. the amount of injected noise that may vary across methods), we roll-out for each model its associated deterministic policies.

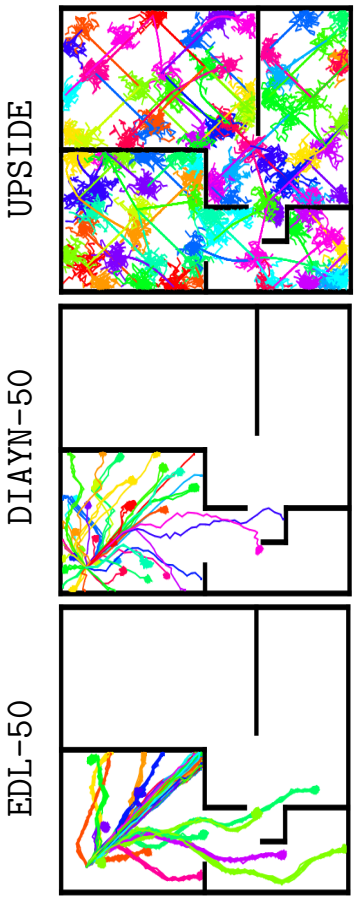
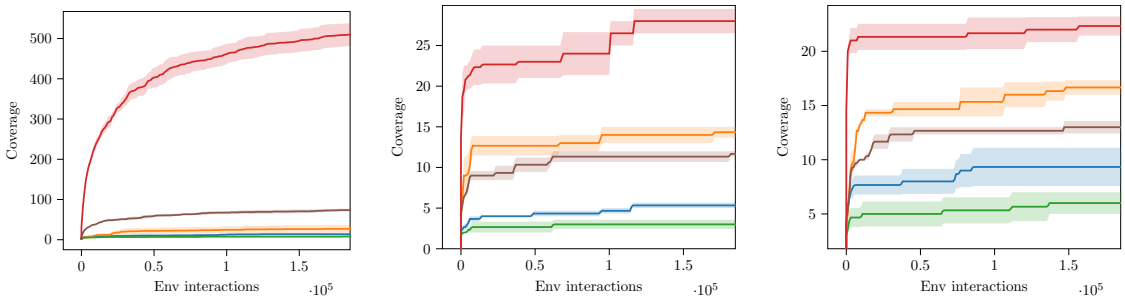


Figure 4.2 – Policies learned on the Bottleneck Maze (see Fig. B.8 in App. B.3 for the other methods): contrary to the baselines, UPSIDE successfully escapes the bottleneck region.

goal location through fine-tuning of the policy. We perform our experiments based on the SaLinA framework [Den+21].

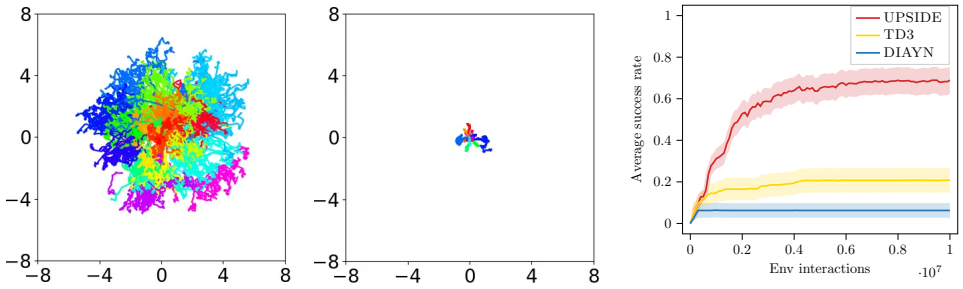
We compare UPSIDE to different baselines. First we consider DIAYN-N_Z [Eys+19], where N_Z denotes a fixed number of skills. We introduce two new baselines derived from DIAYN: a) DIAYN-curr is a curriculum variant where the number of skills is automatically tuned following the same procedure as in UPSIDE, similar to [Ach+18], to ensure sufficient discriminability, and b) DIAYN-hier is a hierarchical extension of DIAYN where the skills are composed in a tree as in UPSIDE but without the diffusing part. We also compare to SMM [Lee+19b], which is similar to DIAYN but includes an exploration bonus encouraging the policies to visit rarely encountered states. In addition, we consider EDL [Cam+20] with the assumption of the available state distribution oracle (since replacing it by SMM does not lead to satisfying results in presence of bottleneck states as shown in [Cam+20]). Finally, we consider the RANDOM policy, which samples actions uniformly in the action space. We use TD3 as the policy optimizer [FHM18] though we also tried SAC [Haa+18] which showed equivalent results than TD3 with harder tuning. Similar to e.g., Eysenbach et al. [Eys+19] and Bagaria and Konidaris [BK20], we restrict the observation space of the discriminator to the cartesian coordinates (x, y) for Ant and x for Half-Cheetah and Walker2d. All algorithms were ran on $T_{\text{max}} = 1e7$ unsupervised environment interactions in episodes of size $H_{\text{max}} = 200$ (resp. 250) for mazes (resp. for control). For baselines, models are selected according to the cumulated intrinsic reward (as done in e.g., [Str+21]), while UPSIDE, DIAYN-hier and DIAYN-curr are selected according to the highest number of η-discriminable policies. On the downstream tasks, we consider ICM [Pat+17] as an additional baseline. See Section B.3 for the full experimental details.

Coverage. We analyze the coverage achieved by the various methods following an unsupervised phase of at most $T_{\text{max}} = 1e7$ environment interactions. For UPSIDE, we report coverage for the skill and diffusing part lengths $T = H = 10$ in the continuous mazes (see App. B.4.4 for an ablation on the values of T, H) and $T = H = 50$ in control environments. Fig. 4.2 shows that UPSIDE manages to cover the near-entirety of the state space of the bottleneck maze (including the top-left room) by creating a tree of directed skills, while the other methods struggle to escape from the bottleneck region. This translates quantitatively in the coverage measure of Table 4.2 where UPSIDE achieves the best results. As shown in Fig. 4.3 and 4.4, UPSIDE clearly outperforms DIAYN and RANDOM in state-coverage of control environments, for the same number of environment interactions. In the Ant domain, traces from DIAYN (Fig. 4.4b) and discriminator curves in App. B.4.3 demonstrate that even though DIAYN successfully



(**a**) Ant (**b**) Half-Cheetah (**c**) Walker2d

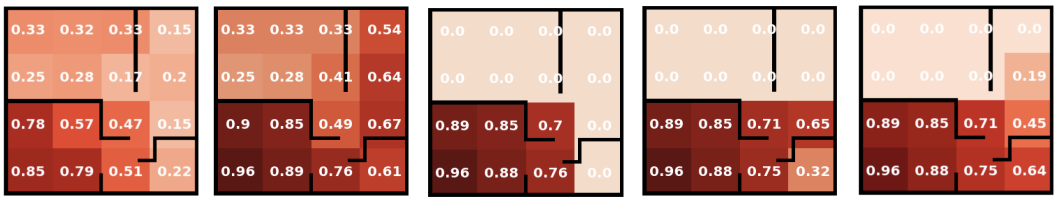
Figure 4.3 – Coverage on control environments: `UPSIDE` covers the state space significantly more than `DIAYN` and `RANDOM`. The curve represents the number of buckets reached by the policies extracted from the unsupervised phase of `UPSIDE` and `DIAYN` as a function of the number of environment interactions. `DIAYN` and `UPSIDE` have the same amount of injected noise. Each axis is discretized into 50 buckets.



(**a**) `UPSIDE` policies on Ant (**b**) `DIAYN` policies on Ant (**c**) Fine-tuning on Ant

Figure 4.4 – (a) & (b) Unsupervised phase on Ant: visualization of the policies learned by `UPSIDE` and `DIAYN-20`. We display only the final skill and the diffusing part of the `UPSIDE` policies. (c) Downstream tasks on Ant: we plot the average success rate over 48 unknown goals (with sparse reward) that are sampled uniformly in the $[-8, 8]^2$ square (using stochastic roll-outs) during the fine-tuning phase. `UPSIDE` achieves higher success rate than `DIAYN-20` and `TD3`.

fits 20 policies by learning to take a few steps then hover, it fails to explore the environment. In Half-Cheetah and Walker2d, while DIAYN policies learn to fall on the agent's back, UPSIDE learns to move forward/backward on its back through skill composition.



(a) UPSIDE: before and after fine-tuning (b) DIAYN (c) EDL (d) ICM

Figure 4.5 – Downstream task performance on Bottleneck Maze: UPSIDE achieves higher discounted cumulative reward on various unknown goals (See Fig. B.9 in App. B.3 for SMM and TD3 performance). From each of the 16 discretized regions, we randomly sample 3 *unknown goals*. For every method and goal seed, we roll-out each policy (learned in the unsupervised phase) during 10 episodes and select the one with largest cumulative reward to fine-tune (with sparse reward $r(s) = \mathbb{I}[\|s - g\|_2 \leq 1]$). Formally, for a given goal g the reported value is $\gamma^\tau \mathbb{I}[\tau \leq H_{\text{max}}]$ with $\tau := \inf\{t \geq 1 : \|s_t - g\|_2 \leq 1\}$, $\gamma = 0.99$ and horizon $H_{\text{max}} = 200$.

Unknown goal-reaching tasks. We investigate how the tree of policies learned by UPSIDE in the unsupervised phase can be used to tackle goal-reaching downstream tasks. All unsupervised methods follow the same protocol: given an unknown[7] goal g, i) we sample rollouts over the different learned policies, ii) then we select the best policy based on the maximum discounted cumulative reward collected, and iii) we fine-tune this policy (i.e., its sequence of directed skills and its final diffusing part) to maximize the sparse reward $r(s) = \mathbb{I}[\|s - g\|_2 \leq 1]$. Fig. 4.5 reports the discounted cumulative reward on various goals after the fine-tuning phase. We see that UPSIDE accumulates more reward than the other methods, in particular in regions far from s_0, where performing fine-tuning over the entire skill path is especially challenging. In Fig. 4.6 we see that UPSIDE's fine-tuning can slightly deviate from the original tree structure to improve the goal-reaching behavior of its candidate policy. We also perform fine-tuning on the Ant domain under the same setting. In Fig. 4.4c, we show that UPSIDE clearly outperforms DIAYN-20 and TD3 when we evaluate the average success

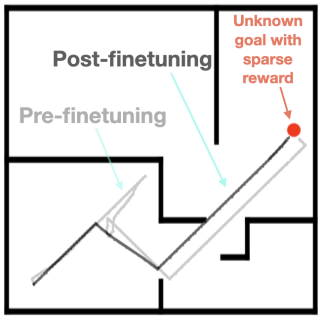


Figure 4.6 – For an unknown goal location, UPSIDE identifies a promising policy in its tree and fine-tunes it.

[7]Notice that if the goal was known, the learned discriminator could be directly used to identify the most promising skill to fine-tune.

rate of reaching 48 goals sampled uniformly in $[-8, 8]^2$. Note that DIAYN particularly fails as its policies learned during the unsupervised phase all stay close to the origin s_0.

Ablative study of the UPSIDE components. The main components of UPSIDE that differ from existing skill discovery approaches such as DIAYN are: the decoupled policy structure, the constrained optimization problem and the skill chaining via the growing tree. We perform ablations to show that all components are simultaneously required for good performance. First, we compare UPSIDE to flat UPSIDE, i.e., UPSIDE with the tree depth of 1 ($T = 150, H = 50$). Table 4.2 reveals that the tree structuring is key to improve exploration and escape bottlenecks; it makes the agent learn on smaller and easier problems (i.e., short-horizon MDPs) and mitigates the optimization issues (e.g., non-stationary rewards). However, the diffusing part of flat UPSIDE largely improves the coverage performance over the DIAYN baseline, suggesting that the diffusing part is an interesting structural bias on the entropy regularization that pushes the policies away from each other. This is particularly useful on the Ant environment as shown in Fig. 4.4. A challenging aspect is to make the skill composition work. As shown in Table 4.1, DIAYN-hier (a hierarchical version of DIAYN) does not perform as well as UPSIDE by a clear margin. In fact, UPSIDE's direct-then-diffuse decoupling enables both policy re-usability for the chaining (via the directed skills) and local coverage (via the diffusing part). Moreover, as shown by the results of DIAYN-hier on the bottleneck maze, the constrained optimization problem (Equation (4.4)) combined with the diffusing part is crucial to prevent consolidating too many policies, thus allowing a sample efficient growth of the tree structure.

4.6 Conclusion and Limitations

We introduced UPSIDE, a novel algorithm for unsupervised skill discovery designed to trade off between coverage and directedness and develop a tree of skills that can be used to perform efficient exploration and solve sparse-reward goal-reaching downstream tasks. Limitations of our approach that constitute natural venues for future investigation are: **1**) The diffusing part of each policy could be explicitly trained to maximize local coverage around the skill's terminal state; **2**) UPSIDE assumes a good state representation is provided as input to the discriminator, it would be interesting to pair UPSIDE with effective representation learning techniques to tackle problems with high-dimensional input; **3**) As UPSIDE relies on the ability to reset to establish a root

node for its growing tree, it could be relevant to extend the approach in non-episodic environments.

Part II

Transformer-based Symbolic World Models

Regression plays a crucial role in the modeling of systems. Model-based RL (MBRL) is an agent learning approach, known for its superior sample efficiency, that relies on learned predictive world models. However, recent studies predominantly employ neural networks for this purpose, which often suffer from limited generalization performance in scenarios with limited data availability. Consequently, model-based algorithms generally underperform compared to their model-free counterparts due to inaccurate world modeling. In this study, we investigate the application of *symbolic regression* as an alternative method for acquiring predictive equation-like models that offer interpretability and demonstrate superior generalization capabilities. We hypothesize and show that the latter helps mitigating the need to augment agents with adaptations modules such as the ones presented in Part I.

We propose and demonstrate that these models help alleviate the necessity of augmenting agents with adaptation modules, as presented in Part I. Moreover, we formally introduce the problem of symbolic regression, which involves discovering analytical expressions from data, and explore a collection of techniques collectively referred to as Deep Generative Symbolic Regression. These techniques approach the task as a natural language processing problem utilizing generative models. Finally, we conduct experiments to evaluate the effectiveness of symbolic regression within the context of MBRL.

Chapter 5

Symbolic Regression

5.1 Motivation

Regression is a central problem in machine learning that involves predicting continuous outcomes based on input variables; given a set of n observations, i.e. features $\{\mathbf{x}_i\}_{i\leq N}$ and target variable $\{y_i\}_{i\leq N}$, it implies finding a function f such that $y_i \approx f(x_i), \forall i \leq N$. As mentioned in the introductory paragraph of the present part, it is a core component of MBRL for learning predictive dynamics models. The ultimate goal of achieving generalization as explained in Section 2.3.1; to achieve this, careful attention must be paid to the selection of the hypothesis space $\mathcal{F}$ from which f belongs to, as it directly affects the model's ability to generalize to unseen data. Some methods, e.g. linear or polynomial regression, heavily restrict the class of functions allowed; they are easy to fit, provide interpretable models but may underfit the data by oversimplifying relationships and limiting flexibility. In contrast, approaches with high capacity, irrespective of their parametric and non-parametric properties, offer greater flexibility and can capture complex relationships and patterns. However, they run the risk of overfitting, where the model becomes too sensitive to noise and struggles to generalize to new, unseen data. Regularization techniques have been developed to address this issue by constraining the model's complexity, reducing the risk of overfitting to the training data. As suggested by the No Free Lunch theorem [WM97], the optimal choice of algorithm depends on the specific problem, available data, and the desired trade-off between model complexity, interpretability, and generalization.

Symbolic regression (SR) attempts at discovering f from the class $\mathcal{F}$ of interpretable analytical expressions with low-complexity. Human-readable models have

shown to be particularly useful to gain a deeper understanding of the underlying mechanisms of physical systems, e.g., materials sciences [WWR19; Kab+21; Ma+22] or physics [SL09; Vad+20; Sun+22; Cra+20; Her+19; UT20]. SR is a challenging task, which implies composing inherently discrete objects (operators and variables) to make the resulting expression fit well the given data. It involves simultaneously finding the structure, i.e. the "shape" of the expression, its operators (from a set of allowed operators e.g. **??** in chapter 6), variables, and the numerical parameters – *constants* – of an expression. [VP22] showed that SR is NP-Hard, therefore typical approaches are based on heuristics. The recent benchmarking effort `SRBench` [La +21a] has shown that SR algorithms have superior generalization on a set of real-world and synthetic datasets than classical regression algorithms, e.g. decision-tree ensembles or neural networks.

5.2 Problem formalization

Let us denote by $\mathcal{F}$ the family of symbolic expressions[1] that form the search space of the SR problem. Generally speaking, $\mathcal{F}$ is defined by the set of building blocks from which expressions can be composed [VP22]. These usually include constants (possibly sampled from a distribution, e.g., $\mathcal{N}(0, 1)$), variables, and basic operators as well as trigonometric or transcendental ones (e.g., $+, -, \times, \div, \sin, \cos, \exp, \log$).

These expressions can be encoded as computation trees as shown in fig. 5.1. In an analytical expression, the tree structure, associated operators and variables are usually grouped under the name of expression *form*, *skeleton* or *structure*, and numerical constants are degrees of freedom – for a fixed structure – on which one can apply continuous optimization techniques.

A common formulation of SR poses to seek a well-fitting expression for the dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i \leq N}$ where $(\forall i \leq N \mathbf{x_i}, y_i) \in \mathbb{R}^d \times \mathbb{R}$ by finding $f \in \mathcal{F}$ that minimizes prediction errors:

$$f^* = \arg\min_{f \in \mathcal{F}} \mathcal{L}(f, \mathcal{D}) \tag{5.1}$$

where commonly used risk functions can be the expected absolute error, expected squared error or the coefficient of determination R^2. In this work, we will consider

[1] Note that even though different expressions may be functionally equivalent, this is normally not taken into account in existing approaches, as determining functional equivalence can be undecidable [BL82].

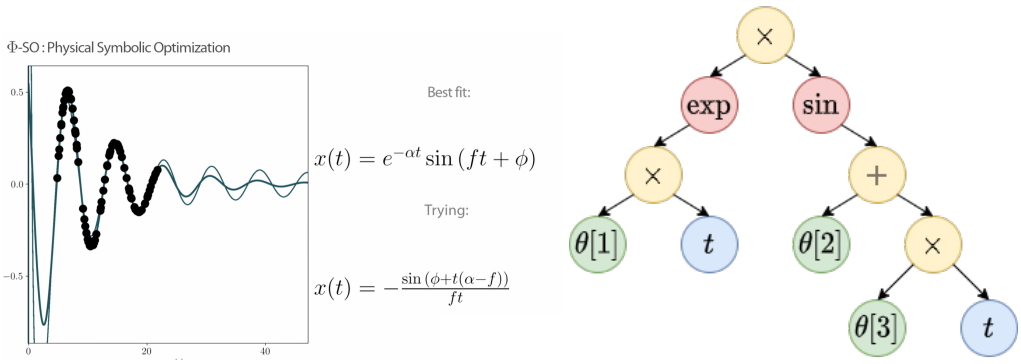


Figure 5.1 – (Left) An illustration of the SR problem (from [TID23]): the algorithm observes the black dots and tries to discover the $x(t) = \exp(-\alpha t)\sin(ft + \phi)$. (Right) Description of $x(t)$ as a tree where constants are represented by constant placeholders θ.

$\mathcal{L} = -R^2$ where:

$$R^2(f, \mathcal{D}) = 1 - \frac{MSE(\mathbf{y}, f(\mathbf{x}))}{VAR(\mathbf{y})} = 1 - \frac{\sum_i (y_i - f(\mathbf{x}_i))^2}{\sum_i (y_i - \bar{y}_i)^2}. \tag{5.2}$$

R^2 is classically used in statistics, but it is unbounded, hence a single bad prediction can cause the average R^2 over a set of examples to be extremely bad. To circumvent this, we set $R^2 = 0$ upon pathological examples as in [La +21a], which can be obtained by finding the expression $f = \bar{y}$.

Certain works include regularization terms in the loss function, i.e. $\lambda C(f)$, where $\lambda \in \mathbb{R}$ controls the regularization strength and $C : \mathcal{F} \mapsto \mathbb{R}$ is a function of the *complexity* of f. Its purpose is to reduce overfitting (equivalently improve generalization), as well as improving the interpretability of f. In SRBench, complexity is defined as the *expression size* (i.e., the number of operators, variables and constants in the expression, each counted with the same weight). There exists different definition of complexity, e.g. considering the structure of the expression tree and using different weights for operators, constants and variables [Kom+15b].

Evaluation setting. Recently, [La +21a] proposed SRBench, a benchmark to evaluate SR methods in a rigorous ways, with a set of design choices. Its repository contains a set of 252 regression datasets from the Penn Machine Learning Benchmark (PMLB)[Fri01] in addition to 14 open-source SR and 7 ML baselines. The datasets consist in "ground-truth" problems where the true underlying function is known, as well as "black-box" problems which are more general regression datasets without an underlying ground truth. Each dataset is split into 75% training data and 25% test data, on which

performance is evaluated. The performance of algorithms is measured in terms of trade-off between accuracy represented by the R^2 on the test set and expression size.

In the following sections, we first review main families of SR approaches then propose a general formulation of the SR problem, which allows to include every approach into a unifying framework.

5.3 Algorithms

Two main families to SR are Genetic Programming (GP) and neural approaches. GP approaches have been dominanting the SR literature and the latter have only emerged a few years ago thanks to the increasing attention to deep learning. By no means exhaustive, the following review aims at showcasing the main ideas behind those for the reader to grasp the main differences between GP and neural approaches.

5.3.1 Genetic Programming

Genetic Programming (GP) is a meta-heuristic inspired by natural evolution and is a popular approach for many combinatorial optimization problems, including discovering computer programs [Koz94; PLM08]. GP lends particularly well to SR as expressions are tree structures. As illustrated in fig. 5.2, they operate by initially sampling a population of random trees then modifying the candidate population using the following following steps in an iterative fashion.

1. *mutation* and *cross-overs* (illustrated respectively in Figure 5.3a and fig. 5.3b)
2. evaluation of $\mathcal{L}$ by executing the trees (potentially constant optimization)
3. *selection*, i.e. stochastic survival of the fittest.

GP-based SR algorithms are a form of hill-climbing where search is poorly guided, i.e. actions are generally randomly picked irrespective of the the dataset and current expression, thus leading to dramatically changing functions (in terms of outputs), invalid or bad expressions; finding an accurate expression can sometimes appear to be explained by sheer-luck. The GP community has developed over the years a set of heuristics to improve search as described non-exhaustively below for the interested reader (not required to understand the manuscript).

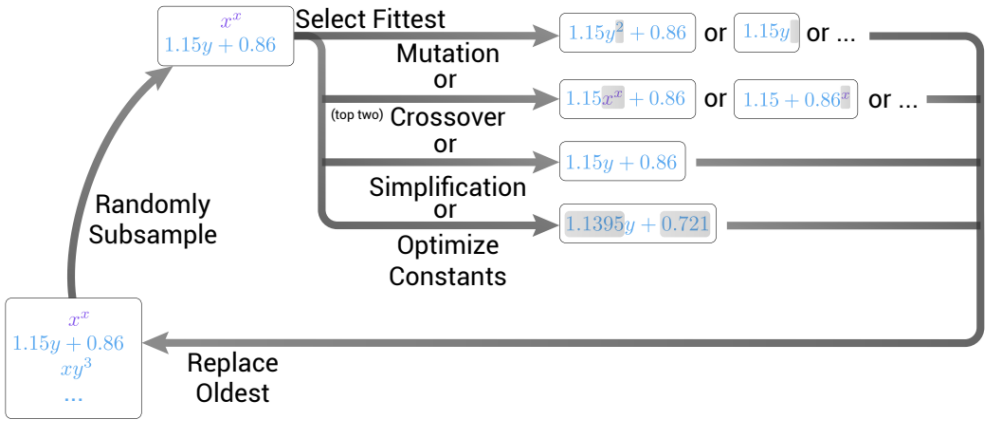


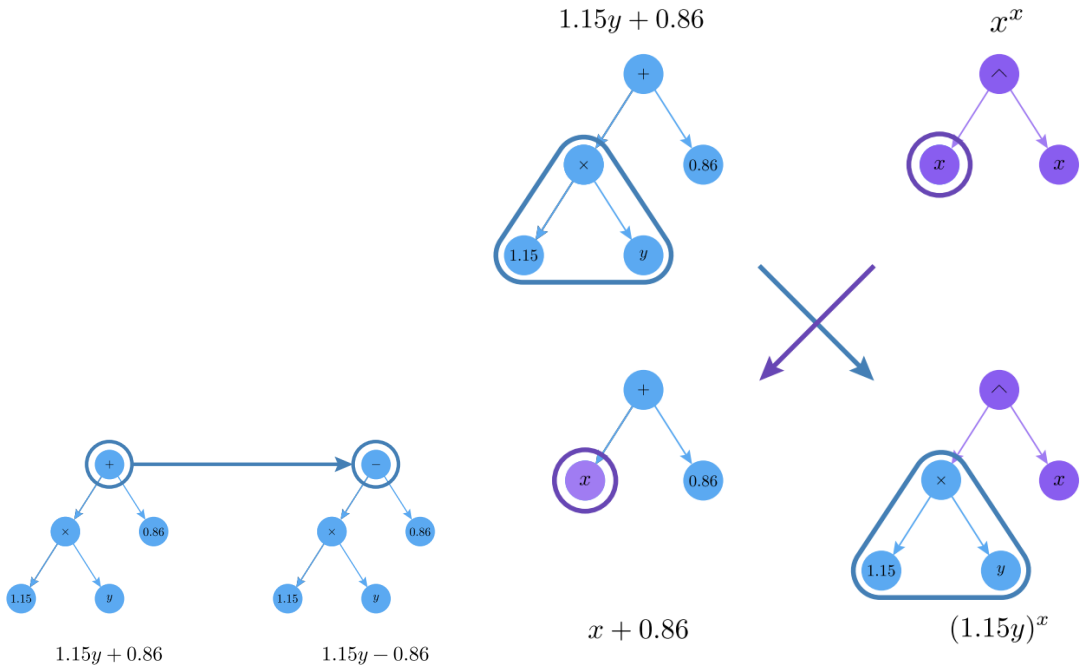
Figure 5.2 – Illustration borrowed from [Cra23] of the high-level population evolution in GP algorithms, notably the PySR [Cra20] algorithm.

Special *mutations* that leverage program semantics by using the candidate's functions intermediate outputs over training samples. For instance, SBP-GP [VAB19] considers sub-problems by computing the intermediate objective at a given node. Special treatment is granted to *constants optimization* as they are responsible for large variations in the outputs. [Kom+20; Cra20] incorporates non-linear squares constants optimization, respectively with Levenberg-Marquadt and BFGS.

Another line of work has reduced the size of search space at the cost of expressiveness by using structural constraints on the *program representation*. For instance, [McC11; La +18; AKO14] uses linear combinations of evolved expressions, the latter optimizing constants with Lasso regularization for sparse models. [FA21] (resp. [Fra22]) manually defines expressions to be composition of a unary function and a polynomial function (resp. invertible unary function and rationale of polynomial expressions).

During *selection*, the sub-population that survives trades off between multiple competing objectives, e.g. accuracy, complexity, model age [SL09; SL11]. Objectives specific to SR have also been introduced, e.g. [La +19] keeps models that perform well in specific regions of the training data, or shape-constrained SR [Kro+22; Hai+22] keeps models that satisfy a set of hypotheses, e.g. monotonically increasing functions. The trade-off between the multiple objectives is usually dealt with Pareto optimization methods [SK05].

Being a greedy search approach, GP algorithms are prone to falling into local minima, and extensive exploration leads to relatively large run times. In practice



(**a**) Mutation (**b**) Cross-over

Figure 5.3 – Illustrations of evolution operators borrowed from [Cra23]; a mutation (a) and a cross-over (b) is a component recombination between parent expressions.

with time constraints, such as the 24 hours-limit in [La +21a], the most accurate GP methods provide expressions with overly large complexity thus preventing the derivation of meaningful physical insights; on the Feynman datasets [UT20], whose ground-truth expressions have averaged complexity 20 as defined in [La +21b], the current state-of-the-art [BKK20b] predicts expressions with averaged size ≈ 100.

5.3.2 Neural approaches

Lately, there has been a growing interest in the SR community for neural network-based approaches. Firstly, with approaches including neural predictors as a subroutine within a GP algorithm. For instance, [UT20] use NNs to explicitly detect data properties (e.g., symmetries in the data) which are then used to prune the search space of SR.

More recent works, we will group under the name *Deep Generative Symbolic Regression* (`DGSR`), directly act upon generating expression candidates. They tackle the SR problem as a natural language problem by representing expressions (equivalently trees) as a sentence, i.e. f is represented by a sequence of tokens $[e_0, \ldots, e_T]$. For

instance, $x(t)$ from fig. 5.1 can be represented in a prefixed form as

$$[\times, \exp, \times, \theta_1, t, \sin, +, \theta_2, \times, \theta_3, t]$$

Expression candidates are generated using auto-regressive sampling following the distribution g. The probability distribution of sample f, or equivalently $[e_0, \ldots, e_T]$ is:

$$P_f(f) = g(e_0)\Pi_{1 \leq t \leq T} g(e_t|e_{<t}) \tag{5.3}$$

As the first work in generative models for SR, DSR [Pet+19] proposed an approach where g is a recurrent NN, whose parameters are updated using policy gradients with the accuracy of sampled expressions as rewards. Subsequent work have built on top of DSR, e.g. by adding GP sampling [Mun+21] or variable units considerations to restrict generations to be homogeneous [TID23].

Another line of work learn conditional generative models that generate expressions by "looking" at the data at hand, which means that Equation (5.3) becomes:

$$P_f(f|\mathcal{D}) = g(e_0|\mathcal{D})\Pi_{1 \leq t \leq T} g(e_t|e_{<t}, \mathcal{D}) \tag{5.4}$$

Several works [Big+20; Big+21; Val+21; Kam+22] have considered a fixed g distribution but trained on synthetic examples and augmented by the considered dataset $\mathcal{D}$ as an input. During training, the learning problem becomes a supervised learning problem as the ground-truth expression is known and used as a target in a multi-step prediction problem. Rather than focusing on the accuracy of the sampled expressions regarding some criteria such as Equation (5.2) as in DSR, they train on the next-token prediction objective. We will give further details on extensions in Section 5.4 and on synthetic training Chapter 6.

5.4 Meta-learning view of symbolic regression

We now generalize the SR problem described in section 5.2 to the case where good performance is sought across multiple datasets (as one usually seeks a generally-competent search algorithm rather than a dataset-specific one). In SRBench, this is evaluated by considering the ranking of methods by their average or median performance over the different datasets that compose the benchmark.

Algorithm	Pre-train of P_f^0	P_f^t conditioned by	Update of P_f^t
GP [PLM08]	No	Population & Genetic operators	Selection operators
EDA [Kim+14]	No	Explicit Factorization	Selection operators
E2E [Kam+22]	SSL on synthetic data	$\mathcal{D}$ & θ^0	No
DSR [Pet+19]	No	θ^t	Update θ^t with policy gradients
uDSR [Lan+22]	SSL on synthetic data	$\mathcal{D}$ & θ^t	Update θ^t with policy gradients
DGSR+MCTS [Kam+23]	SSL and MCTS on synthetic data	MCTS using $\mathcal{D}$ & θ^t & ψ^t	Update θ^t & ψ^t via Selection & Imitation

Table 5.1 – Unifying view of SR. θ represents weights of a probabilistic neural network that embodies P_f. ψ is parameters of a critic network (as explained in chapter 7). We dive into the details of [Kam+22] in chapter 6 and [Kam+23] in chapter 7.

Given Ω a distribution over datasets $\mathcal{D}$, and a limited budget for the exploration process T, the general objective of SR is to define an algorithm that produce distributions of expressions $f \in \mathcal{F}$, that minimize the following theoretical risk at step T:

$$\mathcal{R}_{\Omega,\mathcal{F}} = \mathbb{E}_{\mathcal{D}\sim\Omega(\mathcal{D})}\mathbb{E}_{f\sim P_f^T(\mathcal{D})}[\mathcal{L}(f,\mathcal{D})] \tag{5.5}$$

with $P_f^T(\mathcal{D})$ the final distribution over expressions provided by the considered algorithm. The typical loss function $\mathcal{L}$ considered in the SR literature is the negative R^2 as explained in section 5.2.

Considering algorithms that start from $P_f^0(\mathcal{D})$ to incrementally build $P_f^T(\mathcal{D})$, the problem can be decomposed into two main steps: 1) define an accurate starting point $P_f^0(\mathcal{D})$ for the search process; 2) specify the exploration process that allows to update $P_f^0(\mathcal{D})$ to form $P_f^T(\mathcal{D})$ in a maximum of T iterations (search trials). Some approaches in the literature only consider the first step (i.e., $P_f^T(\mathcal{D}) = P_f^0(\mathcal{D})$ —no search process). Others only investigate step 2 (i.e., $P_f^0(\mathcal{D}) = P_f^0(\emptyset)$ —no inductive context), as described in the following. Note the algorithm only observe datasets from Ω at search time.

5.4.1 Pre-training: How to define P_f^0

The formulation of Equation (5.5) is reminiscent of meta-learning or few-shot learning where the goal is to train a model on a set of training tasks such that it can solve new ones more efficiently [Sch87; TP12; FAL17]. In that vein, many approaches focus on learning a generative model to induce an implicit form of P_f^0. However, since Ω is unknown before search/test time, P_f^0 is obtained in different ways. For example, traditionally in GP, the dataset does not play a role (i.e., it is $P_f^0(\emptyset)$), but there exist different strategies to randomly initialize the population [Loo07; PK16; AH18].

As explained in Section 5.3.2, generative models [Big+21; Val+21; Kam+22] are trained on large synthetic datasets built from randomly generated expressions [LC19] built without knowing expressions underlying datasets in Ω; we denote this process *pre-training* in what follows Although these methods tend to produce expressions similar to the synthetic ones seen during pre-training, they have the advantage of explicitly considering the dataset as an input: P_f is conditioned on $\mathcal{D}$. Consequently, pre-training approaches to `DGSR` can produce expressions by the simple action of sampling. Different decoding strategies can be considered, for instance Beam Search [WR16]. However, sampling from P_f^0 is limited as the accuracy improvement stops after a small number of samples (50 in [Kam+22]), and can perform badly on out-of-domain datasets given at test-time.

Unlike pre-trained approaches, GP and DSR [Pet+19] do not leverage past experience: every new problem is learned from scratch.

5.4.2 Search Process: How to build P_f^T

Given a dataset $\mathcal{D}$, the aim of any process at search time is to define a procedure to build an accurate distribution $P_f^T(\mathcal{D})$ from the starting one $P_f^0(\mathcal{D})$, via a mix of exploration and exploitation (maximization of Equation (5.5)).

In that aim, the learning process of different SR algorithms can be unified into a general, iterative framework: until the termination condition is not met (e.g., runtime budged exceeded or satisfactory accuracy reached),

(i) sample one or more symbolic expressions $f \in \mathcal{F}$ using the current probability distribution P_f^t at step t;

(ii) update P_f^t using statistics of the sample, such as the accuracy of the expression.

Steps (i) and (ii) constitute an iteration of the framework.

Different SR algorithms have considered various definitions of P_f^t and strategies to update it. Its definition can be implicit, explicit, or learned, and be biased towards certain expression structures. Table 5.1 provides a non-exhaustive summary of popular algorithms, which we elaborate upon in the following paragraphs.

GP implements step (i) by maintaining a population of expressions which undergoes stochastic modifications (via crossover and mutation), and step (ii) by selection, i.e., stochastic survival of the fittest, where better expressions are duplicated and

worse ones are removed. In other words, sampling and updating P_f^t is implicitly defined by the heuristics that are chosen to realize the evolution.

Estimation of Distribution Algorithms (EDAs) attempt to be more principled than GP by explicitly defining the form of P_f^t [SS97; Hem+12; Kim+14]. Normally, P_f^t is factorized according to the preference of the practitioner, e.g., expression terms can be modelled and sampled independently or jointly in tuples, or chosen among multiple options as the search progresses, using, e.g., the minimum description length principle [Har+99]. EDAs use methods similar to those of GP to realize step (ii).

In DSR [Pet+19], P_f^t is modelled by the action of auto-regressive sampling from g_ϕ and improvement of P_f^t is realized by policy gradients using the accuracy of sampled expressions as rewards. Though being very general and only biased by the model parametrization θ, this approach was found to generate very short expressions with low accuracy on `SRBench`. This is likely due to sparse reward and credit assignment issues typical of RL [Sut84]. [Mun+21] adds GP search samples on top of DSR. Very recently, the work by [Pet+19] was integrated with a pre-training component [Lan+22], albeit in a large pipeline that also includes GP, NNs for search space reduction [UT20], and linear regression. In [Kam+23], present in chapter 7, pre-training and search are also combined using Monte-Carlo Tree Search.

In the recent benchmarking effort `SRBench` by [La +21a], modern GP algorithms have shown to be the most successful. An interesting property that distinguishes GP from other `DGSR` approaches is that its crossover and mutation operators tend to edit existing expressions (thus preserving substantial parts of them), rather than sampling them from scratch [Koz94; PLM08]. The algorithm we develop in Chapter 7 expands expressions over time, similarly to GP.

5.5 Outline of the part

The following chapters will be dedicated to our contributions to the field of SR with methods that were developed under this unifying view (section 5.4) of the learning process of SR algorithms. Chapter 6 will describe our pre-training strategy (Section 5.4.1) to define $P_f^0(\mathcal{D})$ in [Kam+22] as well as place this work within its specific related work. In Chapter F, we present [dAs+22] where we address the particular problem of recurrence prediction, i.e. inferring the recurrence expression that generated a sequence of integer or float numbers. In Chapter G, we present [BBK23] where we augment the model with additional context of prior knowledge on the problem.

Finally, Chapter 7 will focus on [Kam+23] where we enable pre-training approaches to be amenable to search (Section 5.4.2).

Lastly, Chapter 8 will explore the application of SR to model-based reinforcement learning.

Chapter 6

Pre-training deep generative symbolic regression models

6.1 Introduction

Supervised training neural networks built for language modelling on large datasets of synthetic examples has recently been proposed for SR [Val+21; Big+21]. They train Transformer models [Vas+17] on expert examples, i.e. tuples $(\mathcal{D}, f)$ where the model takes as input $\mathcal{D}$ and decodes auto-regressively the tokenized f.

As explained in section 5.2, analytical expressions are composed of both:

- *skeleton*, a parametric expression using variables and a pre-defined list of operators, e.g. $+, \times, \div, \mathrm{sqrt}, \exp, \sin$, which determines the general shape of the law up to a choice of constants, e.g. $f(x) = \cos(ax + b)$.
- numerical constants; they are estimated using optimization techniques, typically the Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS).

Given a trained model, the inference of [Val+21; Big+21] follow a two-step procedure inspired from GP-based algorithms: first predict a candidate skeleton then fit its constants. The skeleton is predicted via a simple forward pass, and a single call to BFGS is needed, thus resulting in a significant speed-up compared to GP. In these works, the language model is thus trained on the task of skeleton prediction. These work are considerably less accurate as state-of-the-art GP, and have so far been limited to low-dimensional functions ($d_{\text{in}} \leq 3$). We argue that two reasons underlie their shortcomings.

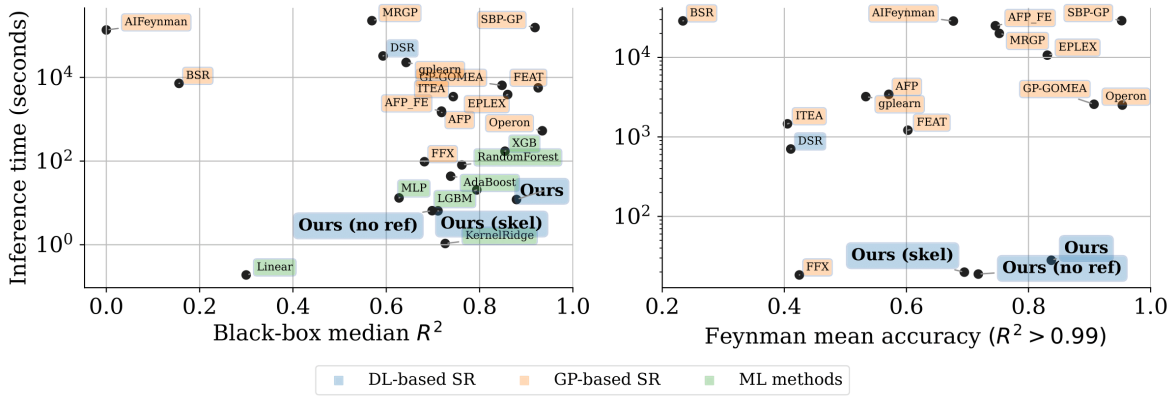


Figure 6.1 – E2E outperforms previous DL-based methods and offers at least an order of magnitude inference speedup compared to SOTA GP-based methods. Pareto plot comparing the average test performance and inference time of our models with baselines provided by the SRbench benchmark [La +21a], both on Feynman SR problems [UT20] and black-box regression problems. We use colors to distinguish three families of models: **deep-learning based SR**, **genetic programming-based SR** and **classic machine learning methods** (which do not provide symbolic solutions). A similar Pareto plot against formula complexity is provided in Fig. C.6.

First, skeleton prediction is an ill-posed problem that does not provide sufficient supervision: different instances of the same skeleton can have very different shapes, and instances of very different skeletons can be very close. Second, the loss function minimized by BFGS can be highly non-convex: even when the skeleton is perfectly predicted, the correct constants are not guaranteed to be found. For these reasons, we believe, and will show, that removing the skeleton estimation as an intermediary step, can greatly improve the SR capacities of language models.

Contributions In this chapter, we introduce E2E, a Transformer model trained on synthetic datasets to perform **end-to-end (E2E)** symbolic regression: solutions are predicted directly, without resorting to skeletons. To this effect, we leverage a hybrid **symbolic-numeric vocabulary**, that uses both symbolic tokens for the operators and variables and numeric tokens for the constants. One can then perform a **refinement** of the predicted constants by feeding them as informed guess to BFGS, mitigating non-linear optimization issues. Finally, we introduce **generation and inference techniques** that allow our models to scale to larger problems: up to 10 input features against 3 in concurrent works.

Evaluated over the SRBench benchmark [La +21a], our model significantly narrows the accuracy gap with state-of-the-art GP techniques, while providing several

orders of magnitude of inference time speedup (see Fig. 6.1). We are the first work that considers inference time, i.e. time required to output a satisfactory expression, to be an important property to tackle practical applications with time constraints, e.g. control or reinforcement learning [Kub+21; Der+19]. We also demonstrate strong robustness to noise and extrapolation capabilities.

6.2 Data generation

Our approach consists in training language models on vast synthetic datasets. Each synthetic training example is a pair: a set of N points $(x, y) \in \mathbb{R}^D \times \mathbb{R}$ as the input, and an expression f such that $y = f(x)$ as the target[1] In what follows, we will explain the process of constructure synthetic samples. We first show how we sample random expressions f, then a set of N features $(x_i)_{i \in \mathbb{N}_N}$ in $\mathbb{R}^D$. Finally, we compute $y_i = f(x_i)$.

6.2.1 Generating expressions

To sample functions f, we follow the seminal approach of Lample and Charton [LC19], and generate random trees with mathematical operators as internal nodes and variables or constants as leaves. The procedure is detailed below (see Table C.1 in the Appendix for the values of parameters):

1. Sample the desired **input dimension** d_{in} of the function f from $\mathcal{U}\{1, D_{\text{max}}\}$.
2. Sample the number of **binary operators** b from $\mathcal{U}\{D-1, D+b_{\text{max}}\}$ then sample b operators from $\mathcal{U}\{+, -, \times\}$[2].
3. Build a **binary tree** with those b nodes, using the sampling procedure of [LC19].
4. For each **leaf** in the tree, sample one of the variables x_d, $d \in [\![1, d_{\text{in}}]\!]$.
5. Sample the number of **unary operators** u from $\mathcal{U}\{0, u_{\text{max}}\}$ then sample u operators from the list O_u in Table C.1, and insert them at random positions in the tree.
6. For each variable x_d and unary operator u, apply a random **affine transformation**, i.e. replace x_d by $ax_d + b$, and u by $au + b$, with (a, b) sampled from $\mathcal{D}_{\text{aff}}$.

[1] We only consider functions from $\mathbb{R}^D$ into $\mathbb{R}$; the general case $f : \mathbb{R}^D \to \mathbb{R}^P$ can be handled as P independent subproblems.

[2] Note that although the division operation is technically a binary operator, it appears much less frequently than additions and multiplications in typical expressions [Gui+20], hence we replace it by the unary operator $\text{inv}: x \to 1/x$.

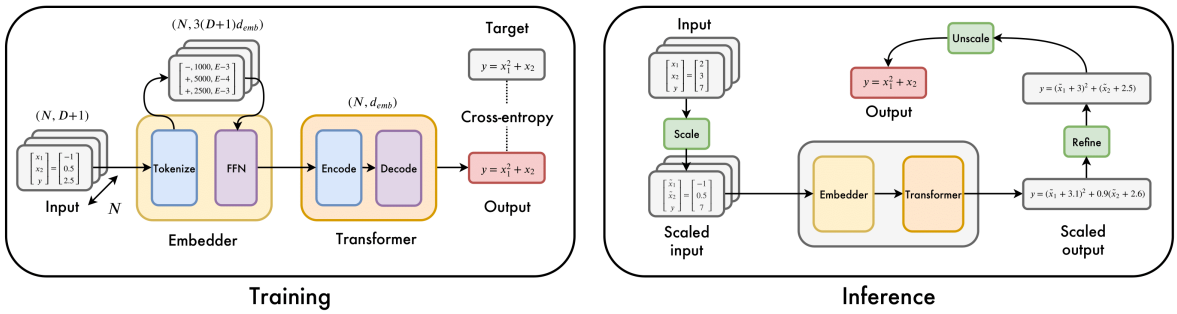


Figure 6.2 – Sketch of our model. During training, the inputs are all whitened. At inference, we whiten them as a pre-processing step; the predicted function must then be unscaled to account for the whitening.

As discussed quantitatively in App. C.3, the number of possible skeletons as well as the random sampling of numerical constants guarantees that our model almost never sees the same function twice, and cannot simply perform memorization. See App. C.2 for examples of the skeleton of generated expressions.

6.2.2 Generating $\mathcal{D}$

For each expression $f : \mathbb{R}^d_{\text{in}} \to \mathbb{R}$, we sample $N \in \mathcal{U}\{10d_{\text{in}}, N_{\text{max}}\}$ **input values** $x_i \in \mathbb{R}^d_{\text{in}}$ from the distribution $\mathcal{D}_x$ described below, and compute the corresponding **output values** $y_i = f(x_i)$. If any x_i is outside the domain of definition of f or if any y_i is larger 10^{100}, the process is aborted, and we start again by generating a new expression. Note when rejecting and resampling out-of-domain values of x_i, the obvious and cheaper alternative, we observed the model used this additional information to predict expressions with a restricted domain of definition.

To maximize the diversity of input distributions seen during training, we sample our inputs from a mixture of distributions (uniform or gaussian), centered around k random centroids[3], see App. C.1 for some illustrations at $d_{\text{in}} = 2$. Input samples are generated as follows:

1. Sample a **number of clusters** $k \sim \mathcal{U}\{1, k_{max}\}$ and k **weights** $w_i \sim \mathcal{U}(0, 1)$, which are then normalized so that $\sum_i w_i = 1$.
2. For each cluster $i \in \mathbb{N}_k$, sample a **centroid** $\mu_i \sim \mathcal{N}(0, 1)^d_{\text{in}}$, a vector of **variances** $\sigma_i \sim \mathcal{U}(0, 1)^D$ and a **distribution shape** (gaussian or uniform) $\mathcal{D}_i \in \{\mathcal{N}, \mathcal{U}\}$.

[3]For $k \to \infty$, such a mixture could in principe approximate any input distribution.

3. For each cluster $i \in \mathbb{N}_k$, sample $\lfloor w_i N \rfloor$ **input points** from $\mathcal{D}_i(\mu_i, \sigma_i)$ then apply a **random rotation** sampled from the Haar distribution, thus allowing to obtain correlated features.
4. Finally, **concatenate** all the points obtained and **whiten** them by substracting the mean and dividing by the standard deviation along each dimension. We observed experimentally that having "standardized" features helped the model to get better performance.

6.3 Method

Below we describe our approach for end-to-end symbolic regression; please refer to Fig. 6.2 for an illustration.

6.3.1 Model

Transformer We use a sequence to sequence Transformer architecture [Vas+17] with 16 attention heads and an embedding dimension of 512, containing a total of 86M parameters. Like [Cha21], we observe that the best architecture for this problem is asymmetric, with a deeper decoder: we use 4 layers in the encoder and 16 in the decoder. A notable property of this task is the permutation invariance of the N input points. To account for this invariance, we remove positional embeddings from the encoder.

As shown in Fig. 6.3 and detailed in App. C.4, the encoder captures the most distinctive features of the functions considered, such as critical points and periodicity, and blends a mix of short-ranged heads focusing on local details with long-ranged heads which capture the global shape of the function.

Tokenization We discretize the input and output of the model as follows. To represent mathematical functions as sequences, we enumerate the trees in prefix order, i.e. direct Polish notation, as in [LC19]. Following [Cha21], we represent numbers in base 10 floating-point notation, round them to four significant digits, and encode them as sequences of 3 tokens: their sign, mantissa (between `0` and `9999`), and exponent (from `E-100` to `E100`). The precision of the mantissa can be either increased by either representing as several tokens (which increases the sequence length) or using a larger float vocabulary. This configuration yielded the best performance according to pre-

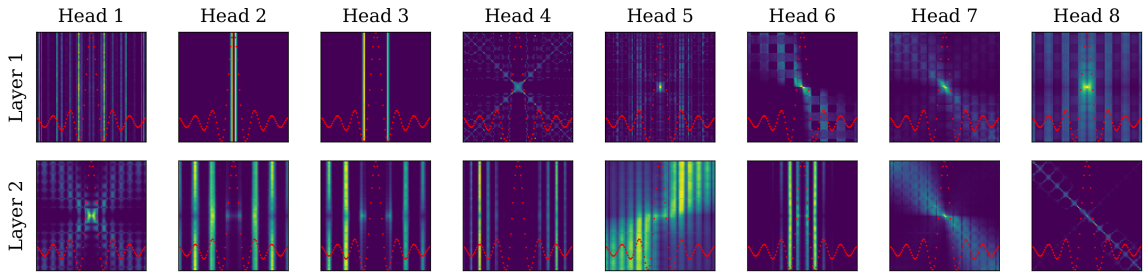


Figure 6.3 – Attention heads reveal intricate mathematical analysis. We considered the expression $f(x) = \sin(x)/x$, with $N = 100$ input points sampled between -20 and 20 (red dots; the y-axis is arbitrary). We plotted the attention maps of a few heads of the encoder, which are $N \times N$ matrices where the element (i, j) represents the attention between point i and point j. Notice that heads 2, 3 and 4 of the second layer analyze the periodicity of the function in a Fourier-like manner.

liminary experiments. For example, the expression $f(x) = \cos(2.4242x)$ is tokenized as `[cos,mul,+,2424,E-3,x]`. Note that the vocabulary of the decoder contains a mix of symbolic tokens (operators and variables) and numeric tokens, whereas that of the encoder contains only numeric tokens[4].

Embedder Our model is provided N input points $(x, y) \in \mathbb{R}^{d_{\text{in}}+1}$, each of which is represented as $3(d_{\text{in}} + 1)$ tokens of dimension d_{emb}. As D and N become large, this results in long input sequences (e.g. 6600 tokens for $d_{\text{in}} = 10$ and $N = 200$), which challenge the quadratic complexity of Transformers. To mitigate this, we introduce an embedder to map each input point to a single embedding. The embedder pads the empty input dimensions to D_{max}, then feeds the $3(D_{\text{max}} + 1)d_{\text{emb}}$-dimensional vector into a 2-layer fully-connected feedforward network (FFN) with ReLU activations, which projects down to dimension d_{emb}[5] The resulting N embeddings of dimension d_{emb} are then fed to the Transformer.

Training We train the model to maximize the log-likelihood of the next token when it conditions on previous tokens. We use the cross-entropy loss with the Adam optimizer, warming up the learning rate from 10^{-7} to 2.10^{-4} over the first 10,000 steps, then decaying it as the inverse square root of the number of steps, following [Vas+17]. We hold out a validation set of 10^4 examples from the same generator, and train our models until the accuracy on the validation set saturates (around 50 epochs of 3M

[4]The embeddings of numeric tokens are *not* shared between the encoder and decoder.

[5]We explored various architectures for the embedder, but did not obtain any improvement; this does not appear to be a critical part of the model.

examples). Input sequence lengths vary significantly with the number of points N; to avoid wasteful padding, we batch together examples of similar lengths, ensuring that a full batch contains a minimum of 10,000 tokens. On 32 GPU with 32GB memory each, one epoch is processed in about half an hour.

6.3.2 Inference tricks

In this section, we describe three tricks to improve the performance of our model at inference.

Table 6.1 – The importance of an end-to-end model with refinement.

Model	Function $f(x, y)$
Target	$\sin(10x)\exp(0.1y)$
Skeleton + BFGS	$-\sin(1.7x)(0.059y + 0.19)$
E2E no BFGS	$\sin(9.9x)\exp(0.1y)$
E2E + BFGS random init	$-\sin(0.095x)\exp(0.27y)$
E2E + BFGS model init	$\sin(10x)\exp(0.1y)$

The skeleton approach recovers an incorrect skeleton. The E2E approach predicts the right skeleton. Refinement worsens original prediction when randomly initialized, and yields the correct result when initialized with predicted constants.

Refinement Previous language models for SR, such as [Big+21], follow a *skeleton* approach: they first predict equation skeletons, then fit the constants with a non-linear optimisation solver such as BFGS. In this work, we follow an *mixed vocabulary* (E2E) approach: predicting simultaneously the function and the values of the constants. However, we improve our results by adding a *refinement* step: fine-tuning the constants a posteriori with BFGS, initialized with our model predictions[6].

This results in a large improvement over the skeleton approach, as we show by training a Transformer to predict skeletons in the same experimental setting. The improvement comes from two reasons: first, prediction of the full formula provides better supervision, and helps the model predict the skeleton; second, the BFGS routine strongly benefits from the informed initial guess, which helps the model predict the constants. This is illustrated qualitatively in Table 6.1, and quantitatively in Table 6.2.

[6]To avoid BFGS having to approximate gradients via finite differences, we provide the analytical expression of the gradient using *sympytorch* [Kid21] and *functorch* [Hor21].

Scaling As described in Section 6.2.2, all input points presented to the model during training are whitened: their distribution is centered around the origin and has unit variance. To allow accurate prediction for input points with a different mean and variance, we introduce a scaling procedure during inference. Let f the function to be inferred, x be the input points, and $\mu = \text{mean}(x), \sigma = \text{std}(x)$. As illustrated in Fig. 6.2 we pre-process the input data by replacing x by $\tilde{x} = \frac{x-\mu}{\sigma}$. The model then predicts $\hat{f}(\tilde{x})$, and we can recover an approximation of f by unscaling the variables using $\tilde{x} = \frac{x-\mu}{\sigma}$ in $\hat{f}$.

This gives our model the desirable property to be insensitive to the scale of the input points: DL-based approaches to SR are known to fail when the inputs are outside the range of values seen during training [dAs+22; Cha21]. Note that here, the scale of the inputs applies to the scale of the constants in the function f; although these coefficients are sampled in $\mathcal{D}_{\text{aff}}$ during training, coefficients outside $\mathcal{D}_{\text{aff}}$ can be expressed by multiplication of constants in $\mathcal{D}_{\text{aff}}$.

Feature selection. Our method still remains improvable in scaling to datasets with $d_{\text{in}} > 10$. The reason we restricted our model to dimension ≤ 10 is that the input sequence length becomes prohibitively long beyond, and that generating high-dimensional functions in an unbiased way becomes increasingly tricky. Nonetheless, since the objective of SR is to output interpretable formulas, we argue that SR is most useful for moderately low dimensional problems. For example, $1-10$ dimensional problems already cover a large class of physical systems : for instance, point objects can be represented by their position, speed and mass, 7 parameters. Additionally, in many real world problems where more than 10 features are available, some of the features are often irrelevant or heavily correlated. To mitigate this, one typically carries out feature selection before modeling the data. We only keep the 10 features most linearly correlated with the output.

Bagging and decoding Since our model was trained on $N \leq 200$ input points, it does not perform satisfactorily at inference when presented with more than 200 input points. To take advantage of large datasets while accommodating memory constraints, we perform *bagging*: whenever N is larger than 200 at inference, we randomly split the dataset into B bags of 200 input points[7].

[7]Smarter splits, e.g. diversity-preserving, could be envisioned, but were not considered here.

For each bag, we apply a forward pass and generate C function candidates via random sampling or beam search using the next token distribution. As shown in App. C.6 (Fig. C.11), the more commonly used beam search [WR16] strategy leads to much less good results than sampling due to the lack of diversity induced by constant prediction (typical beams will look like $\sin(x), \sin(1.1x), \sin(0.9x), \ldots$). This provides us with a set of BC candidate solutions. Since BC can become large, we rank candidate functions (according to their error on *all* input points), get rid of redundant skeleton functions and keep the best K candidates for the refinement step[8]. To speed up the refinement, we use a subset of at most 1024 input points for the optimization. The parameters B, C and K can be used as cursors in the speed-accuracy tradeoff: in the experiments presented in Fig. 6.1, we selected $B = 100$, $C = 10$, $K = 10$.

Our model inference speed has two sources: the forward passes described above on one hand (which can be parallelized up to memory limits of the GPU), and the refinements of candidate expressions on the other (which are CPU-based and could also be parallelized, although we did not consider this option here). Please note that both can be parallelized.

6.4 Experiments

In this section, we present the results of our model. We begin by studying in-domain accuracy, then present results on out-of-domain datasets.

6.4.1 In-domain performance

We report the in-domain performance of our models by evaluating them on a fixed validation set of 100,000 examples, generated as per Section 6.2. Validation functions are uniformly spread out over three difficulty factors: number of unary operators, binary operators, and input dimension. For each function, we evaluate the performance of the model when presented $N = [50, 100, 150, 200]$ input points (x, y), and prediction accuracy is evaluated on $N_{\text{test}} = 200$ points sampled from a fresh instance of the multimodal distribution described in Section 6.2.2.

[8]Though these candidates are the best functions without refinement, there are no guarantees that these would be the best after refinement, especially as optimization is particularly prone to spurious local optimas.

Table 6.2 – Our approach outperforms the skeleton approach.

Model	R^2	$\text{Acc}_{0.1}$	$\text{Acc}_{0.01}$	$\text{Acc}_{0.001}$
Skeleton + BFGS	0.43	0.40	0.27	0.17
E2E no BFGS	0.62	0.51	0.27	0.09
E2E + BFGS random init	0.44	0.44	0.30	0.19
E2E + BFGS model init	**0.68**	**0.61**	**0.44**	**0.29**

Metrics are computed over the $10,000$ examples of the evaluation set.

We assess the performance of our model using two popular metrics: R^2-score [La+21a] already presented in Chapter 5 and accuracy to tolerance τ [Big+21; dAs+22]:

$$R^2 = 1 - \frac{\sum_i^{N_{\text{test}}} (y_i - \hat{y}_i)^2}{\sum_i^{N_{\text{test}}} (y_i - \bar{y})^2}, \qquad \text{Acc}_\tau = \mathbb{1}\left(\max_{1 \leq i \leq N_{\text{test}}} \left|\frac{\hat{y}_i - y_i}{y_i}\right| \leq \tau\right), \tag{6.1}$$

where $\mathbb{1}$ is the indicator function.

The accuracy metric provides an idea of the precision of the predicted expression as it depends on a desired tolerance threshold. However, due to the presence of the $\max$ operator, it is sensitive to outliers, and hence to the number of points considered at test time (more points entails a higher risk of outlier). To circumvent this, we discard the 5% worst predictions, following [Big+21].

End-to-end outperforms skeleton In Table 6.2, we report the average in-domain results of our models. Without refinement, our E2E model outperforms the skeleton model trained under the same protocol in terms of low precision prediction (R^2 and $\text{Acc}_{0.1}$ metrics), but small errors in the prediction of the constants lead to lower performance at high precision ($\text{Acc}_{0.001}$ metric). The refinement procedure alleviates this issue significantly, inducing a three-fold increase in $\text{Acc}_{0.001}$ while also boosting other metrics. Initializing BFGS with the constants estimated in the E2E phase plays a crucial role: with random initialization, the BFGS step actually *degrades* E2E performance. However, refinement with random initialization still achieves better results than the skeleton model: this suggests that the E2E model predicts skeletons better that the skeleton model.

Ablation Fig. 6.4 (A,B,C) presents an ablation over three indicators of formula difficulty (from left to right): number of unary operators, number of binary operators and input dimension. In all cases, increasing the factor of difficulty degrades performance,

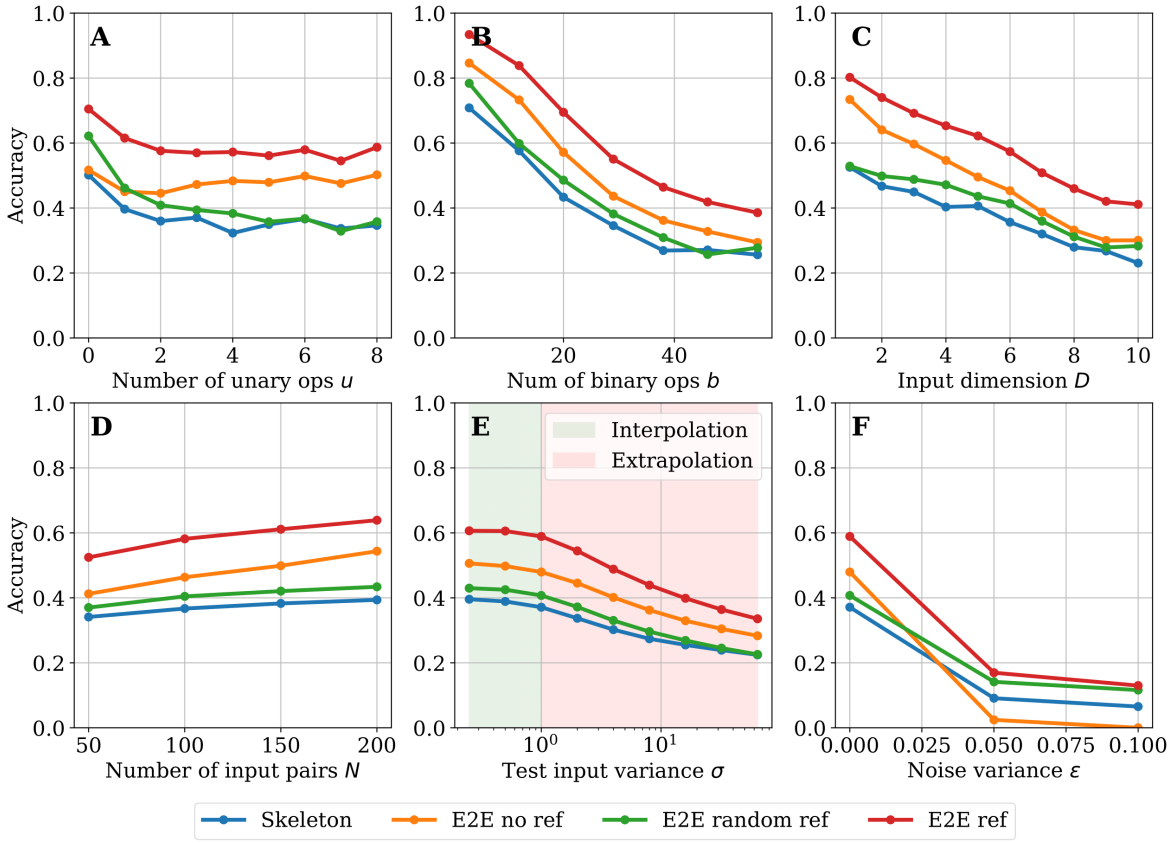


Figure 6.4 – Ablation over the function difficulty (top row) and input difficulty (bottom row). We plot the accuracy at $\tau = 0.1$ (Eq. 6.1), see App. C.5 for the R^2 score. We distinguish four models: **skeleton**, **E2E without refinement**, **E2E with refinement from random guess** and **E2E with refinement**. **A:** number of unary operators. **B:** number of binary operators. **C:** input dimension. **D:** Low-resource performance, evaluated by varying the number of input points. **E:** Extrapolation performance, evaluated by varying the variance of the inputs. **F:** Robustness to noise, evaluated by varying the multiplicative noise added to the labels.

as one could expect. This may give the impression that our model does not scale well with the input dimension, but we show that our model scales in fact very well on out-of-domain datasets compared to concurrent methods (see Fig. C.10 of the Appendix). We include a qualitative analysis on the improvement induced by the use of mixture of distributions in App. C.5.

Fig. 6.4 (D) shows how performance depends on the number of input points fed to the model, N. In all cases, performance increases, but much more signicantly for the E2E models than for the skeleton model, demonstrating the importance of having a lot of data to accurately predict the constants in the expression.

Extrapolation and robustness In fig. 6.4 (E), we examine the ability of our models to interpolate/extrapolate by varying the scale of the test points: instead of normalizing the test points to unit variance, we normalize them to a scale σ. As expected, performance degrades as we increase σ, however the extrapolation performance remains decent even very far away from the inputs ($\sigma = 32$).

Finally, in fig. 6.4 (F), we examine the effect of corrupting the targets y with a multiplicative noise of variance σ: $y \to y(1+\xi), \xi \sim \mathcal{N}(0, \varepsilon)$. The results reveal something interesting: without refinement, the E2E model is not robust to noise, and actually performs worse than the skeleton model at high noise. This shows how sensitive the Transformer is to the inputs when predicting constants. Refinement improves robustness significantly, but the initialization of constants to estimated values has less impact, since the prediction of constants is corrupted by the noise.

6.4.2 Out-of-domain generalization

We evaluate our method on the recently released benchmark SRBench [La +21a].

The overall performance of our models is illustrated in the Pareto plot of Fig. 6.1, where we see that on both types of problems, our model achieves performance close to state-of-the-art GP models such as Operon with a fraction of the inference time[9]. Impressively, our model outperforms all classic ML methods (e.g. XGBoost and Random Forests) on real-world problems with a lower inference time, and while outputting an interpretable formula.

We provide more detailed results on Feynman problems in Fig. 6.5, where we additionally plot the formula complexity, i.e. the number of nodes in the mathematical tree (see App. C.6 for similar results on black-box and Strogatz problems). Varying the noise applied to the targets noise, we see that our model displays similar robustness to state-of-the-art GP models. We additionally include ablation on the use of scaling during inference in App. C.5.

While the average accuracy or our model is only ranked fourth, it outputs formulas with lower complexity than the top 2 models (Operon and SBP-GP), which is an important criteria for SR problems: see App. C.6 for complexity-accuracy Pareto plots. To the best of our knowledge, our model is the first non-GP approach to achieve such competitive results for SR.

[9] Inference uses a single GPU for the forward pass of the Transformer.

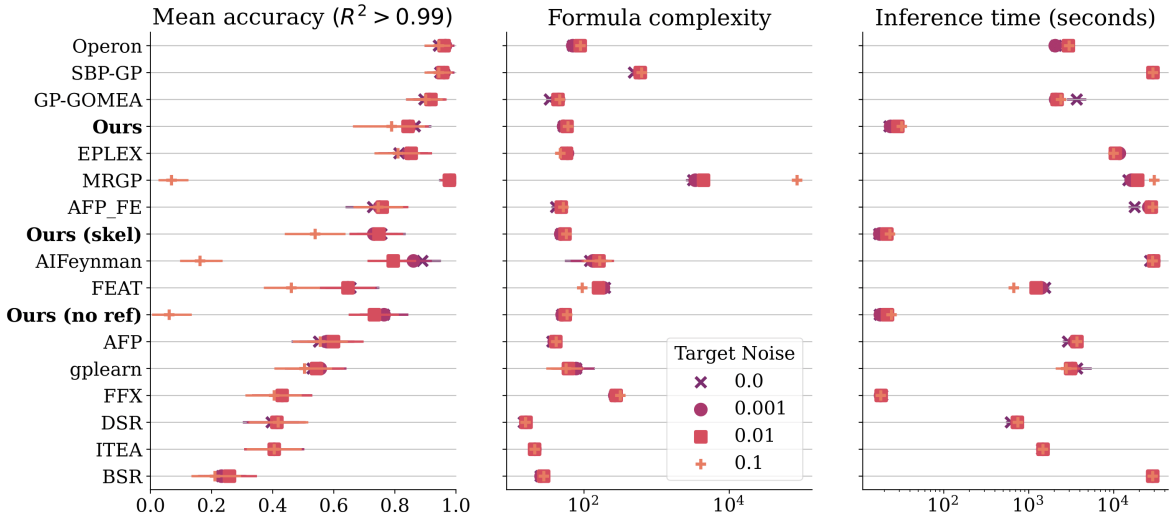


Figure 6.5 – Our model presents strong accuracy-speed-complexity tradeoffs, even in presence of noise. Results are averaged over all 119 Feynman problems, for 10 random seeds and three target noises each as shown in the legend. The accuracy is computed as the fraction of problems for which the R^2 score on test examples is above 0.99. Models are ranked according to the accuracy averaged over all target noise.

6.5 Conclusion

In this work, we introduced a competitive deep learning model for SR by using a novel numeric-symbolic approach. Through rigorous ablations, we showed that predicting the constants in an expression not only improves performance compared to predicting a skeleton, but can also serve as an informed initial condition for a solver to refine the value of the constants.

Our model outperforms previous deep learning approaches by a margin on SR benchmarks, and scales to larger dimensions. Yet, the dimensions considered here remain moderate ($d_{\text{in}} < 10$): adapting to the truly high-dimensional setup is an interesting future direction, and will likely require qualitative changes in the data generation protocol. While our model narrows the gap between GP and DL based SR, closing the gap also remains a challenge for future work. We propose new directions for filling this gap in the next chapter by including a search component and enabling the method to improve at test time.

We were the first to apply SR to the task of recurrence prediction, i.e. predicting a recurrence $u_{n+1} = f(u_n, \cdots, u_{n-k}, n)$ when observing $u_0, \ldots, u_{k-1}$. We build on a similar model than E2E. The interested reader can find details in Chapter F.

Chapter 7

Augmenting deep generative symbolic regression methods with search

7.1 Introduction

Even though they all have in common their ability to search, GP-based methods as well as DSR [Pet+19], ϕ-SO [TID23] and DSR+GP [Mun+21] face a *tabula rasa* setup of the problem for each new dataset. On the other hand, `DGSR` approaches [Big+21; Val+21] or `E2E` [Kam+22], presented in the chapter 6, are purely inductive: they are pre-trained to predict an expression in a single forward pass for any new dataset, by feeding the dataset as input tokens [Big+21; Val+21; Kam+22]. As such, these approaches have the appeal of generating expressions extremely quickly. However, their lack of a search component (i.e. $P_f^t = P_f^0$ under the unifying view of Section 5.4) makes them unable to *improve* for the specific dataset at hand. This aspect can be particularly problematic when the given data is out-of-distribution compared to the synthetic data the NN was pre-trained upon.

A promising direction to cope with the limitations of inductive `DGSR` is therefore to include a suitable search strategy. The use of neural policies in Monte-Carlo Tree Search (MCTS) has led to improved exploration via long-term planning in the context of automated theorem proving with formal systems [PS20; Lam+22], algorithm discovery [Faw+22], and also games [Sil+16; Sil+18]. In the context of SR, search tree nodes are expressions (e.g., $x_1 + x_2$), and edges between them represent mutations (e.g., $x_2 \to (7 \times x_3)$, leading to the expression $x_1 + 7 \times x_3$). [WYS15; Sun+22] proposed a classic formulation of MCTS for SR, where possible mutations are pre-defined along

with prioritization rules to decide which nodes to mutate in priority. [Li+19] use a recurrent neural network to produce possible mutations, which is conditioned on shape constraints for the expressions, but not on the dataset. Most similar to our approach is the study of [Lu+21], that uses a pre-trained model to sample promising but rather simple mutations (up to one expression term). However, the model is not fine-tuned on the specific dataset at hand as the search progresses. In our proposal, MCTS is augmented with neural policies that are both pre-trained and fine-tuned. Existing works have only showed good performance on simple benchmark problems (e.g., no observed competitive performance on `SRBench` real-world problems). Furthermore, we found in preliminary experiments that the combination of NN within MCTS with pre-training is key to achieve good performance.

Contributions In this chapter, we seek to overcome the limitations of `DGSR`, by proposing a synergistic combination, which we call `DGSR+MCTS` [Kam+23], where MCTS is seeded with pre-trained `DGSR`, and `DGSR` is fine-tuned over time on multiple datasets simultanously as the search progresses. A *mutation* policy is responsible for producing mutations that expand expressions from the tree search. A *selection* policy prioritizes which expression to mutate next, by trading-off between exploration (low number of visits) and exploitation (most promising expressions) using a critic network. The mutation policy first undergoes a pre-training phase. Then, both the mutation policy and the critic network are updated in online fashion, by leveraging results from search trials on new provided datasets.

7.2 Method

Following the unified framework of SR detailed in section 5.4, we derive `DGSR+MCTS`, as an expert-iteration algorithm [ATB17], which iteratively 1) samples expressions from P_f^t and 2) updates P_f^t by improving the distribution via imitation learning (i.e., log-likelihood maximization of solution expressions).

Recall that in `DGSR` methods, sampling expressions from P_f^t involve producing tokens step-by-step by sampling from a next-token distribution with techniques such as Monte-Carlo sampling or Beam-search. Turning pre-trained `DGSR` methods into ones that can update P_f^t with expert-iteration is a challenging task as 1) reward signal can be very sparse (i.e., very few sequences may correspond to accurate expressions for the given dataset); 2) such left-to-right blind way of decoding does not allow

for accurate planning; 3) using accuracy objectives to guide the decoding would be difficult with such auto-regressive approaches, since intermediate sequences of tokens are not valid expressions in that setting.

Rather, we propose to derive P_f^t as the distribution induced by a Monte-Carlo Tree Search (MTCS) process [Bro+12], which we iteratively refine via imitation learning on samples it produces that solve the problem. Our MCTS process also considers expression mutations, following a mutation policy M_θ^t which deals with transformations of existing expressions, rather than a greedy concatenation of tokens from a given vocabulary, allowing a more systematic evaluation of intermediate solutions for a more guided exploration. This section first explains our MCTS search process then the way we pre-train the mutation policy from synthetic data.

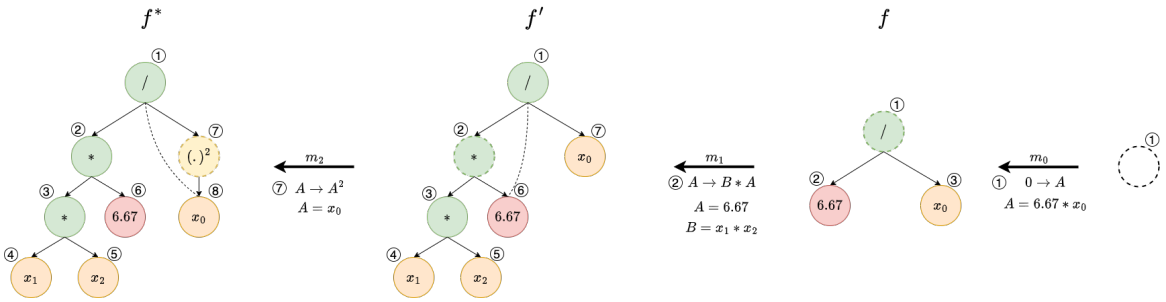


Figure 7.1 – Example of data generation to train the mutation model. Given a starting ground-truth expression (e.g., $f^*(x_0, x_1, x_2) = 6.67x_1x_2/x_0^2$ as a tree, we procedurally dismantle the tree until no node is left. This is done by, at each step (red arrows), a) picking a node (dashed contour), b) removing the picked node and, if the operator is binary, additionally remove the subtree rooted in one of the two child nodes B, c) adding an edge (black dotted line) between the parent node and the remaining child node A to obtain a well-formed expression. When the picked node is the root node, the entire tree is removed, and the dismantling stops. Then, we train the mutation model to assemble the tree back via subsequent mutations (green arrows), which revert the dismantling process. The mutation model is conditioned on the current tree (initially empty) as well as the dataset $\mathcal{D}$.

7.2.1 MCTS Sampling

Our sampling process of new expressions is derived from an MCTS process, where each node of the search tree[1] corresponds to an expression $f \in \mathcal{F}$. Following a similar process as in [Lam+22], our MCTS process is made of three steps: 1) Selection, 2) Expansion, 3) Back-propagation, that we detail below. In our MCTS, M_θ is used

[1]Not to be confused with tree-based representations of expressions, which we discussed in the previous section.

to sample promising mutations that populate the MCTS search tree via expansions (described below). Besides M_θ, we additionally leverage a critic network C_ψ to assess how likely an expression can lead to a solution. We use the same NN (weights are shared) to realize M_θ and C_ψ, with the exception that C_ψ includes an additional head that is trained during the process to output a value in $\mathbb{R}$. These three steps are iteratively repeated 1000 times, a period[2] that we call *search trial*, before M_θ and C_ψ is updated.

Selection The selection step of MCTS aims at following a path in the search tree, from its root, i.e. the empty expression, to a leaf in the search tree, that trades off between exploration and exploitation. Following PUCT [Sil+16], at each new node f, we select its child f' that maximizes:

$$V(f') + p_{uct} E(f') M_\theta(f'|f, \mathcal{D}, \theta) \tag{7.1}$$

with $E(f') = \frac{\sqrt{\sum_{f' \in \text{child}(f)} N(f')}}{1+N(f')}$. $p_{uct} \in \mathrm{R}^+$ is the exploration coefficient, $N(f)$ is the number of times f was selected so far during the search, and $V(f)$ is an estimate of the expected value of the expression, expressed as $v(f)/N(f)$, with $v(f)$ accumulating values of all the nodes depending on f in the tree search. This selection criteria balances exploration and exploitation, controlled by hyper-parameter p_{uct}.

Expansion Once the selection step reaches a leaf f of the tree, this leaf is expanded to produce new child nodes by applying K mutations to f, sampled from $M_\theta(\mathcal{D}, f, \theta)$. This leads to modified expressions $\{f'_k\}_{k \leq K}$. In our case, the distribution $P_f(f'|f, \mathcal{D}, \theta)$ is induced by the application of $m \sim M_\theta$ on f, resulting in $f' = m(f)$. For each $k \leq K$, we add a node f'_k to the search tree, as well as an edge between f'_k and f, labeled m_k.

Each expression resulting from an expansion is evaluated (with or without constant optimization as discussed in Section 7.3.1) to check whether it *solves* the SR problem. Here, we assess whether a relatively high accuracy is reached to determine whether the expression is a solution for the given dataset (we use $R^2 \geq 0.99$ in our experiments). Nodes that solve the problem obtain a value $v(f) = 1$. Others obtain an estimate from the critic network: $v(f) = C_\psi(\mathcal{D}, f)$. We remark that a simpler strategy is to define $v(f)$ as the accuracy of f, i.e. $v(f) = R^2(f, \mathcal{D})$. However, this strategy usually induces deceptive rewards, because a few mutations that lead to less accurate expressions (e.g.,

[2] This corresponds to a single step i) in the unifying framework of section 5.4.

akin to sacrificing pieces in chess) may be needed before a very accurate expression is found (resp., we obtain a winning position). We confirmed that our strategy outperforms the naive use of accuracy in preliminary experiments

Back-Propagation After each expansion, the value of every ancestor f' of any new node f is updated as $V(f') \leftarrow V(f') + v(f)$. Note that this form of back-propagation regards weighing the nodes of the MCTS search tree, and should not be confused with the algorithm used to train NNs.

As mentioned before, selection, expansion, and back-propagation are repeated 1000 times, after which the search trial is completed. At completion, the parameters of M_θ and C_ψ are updated as described in section 7.2.2. Finally, the MCTS search tree is reset, and built anew using updated M_θ and C_ψ during the next trial.

7.2.2 Learning critic C_ψ and M_θ

After each search trial, we update the two parametrized components C_ψ and M_θ. To that end, training samples from the previous search trials are stored in two separate first-in-first-out queues: a buffer stores mutation sequences $(f^{(\tau)}, m_t)$ that produced a solution expression f^* to update M_θ[3], the other contains V values of nodes. For the latter, nodes that lead to a solution expression f^* are assigned a score of 1.0. Others are considered for training only if their visit count is higher than a given threshold N_{visits}^{min}, in order to focus training on sufficiently reliable estimates. At each training step, batches containing an equal proportion of mutation and critic data points are sampled from the queues to train M_θ and C_ψ respectively. Both training objectives are weighted equally. To prevent any mode collapse, we continue sampling training examples from the supervised data used to pre-train the mutation model M_θ, as described in section 7.2.3.

Note that even though the pre-training data generation is biased in different ways, stochasticity of M_θ enables its adaptation over search trials thanks to its updates; for instance, even if M_θ was trained to output mutations with arguments of size $B \approx 10$ can learn mutations of size 1, and vice-versa. As a result, M_θ can automatically learn the appropriate (and dataset-dependent) size of mutations through MCTS search.

We set up our MCTS search to simultaneously operate on multiple datasets at the same time, so that M_θ and C_ψ can leverage potential information that is shared across

[3] If multiple such sequences exist, we select the one with the smallest number of mutations.

different SR instances (transfer learning). Given a set of datasets, a *controller* schedules a queue of datasets that includes a proportion of unsupervised (i.e. the ground-truth expression is not known) datasets to send to *workers* in charge of handling search trials. To avoid catastrophic forgetting, we also continue training on pre-training examples from synthetic datasets i.e. with example mutations as described in section 7.2.3, in the spirit of AlphaStar [Vin+19] or HTPS [Lam+22] with human annotated data. M_θ and C_ψ updates are tackled by *trainers*.

7.2.3 Mutation Policy

Our search process relies on a mutation policy M_θ^t, a distribution over promising mutations of a given expression $f \in \mathcal{F}$. In what follows, we drop the t index from M_θ for clarity. Expressions are represented as a tree structure and mutations act by modifying it.

Definition of M_θ. We define mutations as transformations that can be applied on any node of a given expression tree. Table 7.1 contains the list of considered transformations, where A stands for an existing node of the expression tree and B stands for a new (sub-)expression to be included in the tree. The $\rightarrow$ symbol represents the replacement operation, e.g., $A \rightarrow A + B$ means that node A from f is replaced by a new addition node with A as its left child and B its the right one. Thus, a valid mutation from M_θ is a triplet $< A, op, B >$, where A is a node from the expression tree, op is an operation from table 7.1 and B is a new expression whose generation is described below. A constant optimization step, detailed in Section D.5.2, can be performed after the mutation to better fit the data; we explore in Section 7.3 whether including constant optimization improves the performance. We call mutation *size* the size of the expression B.

The mutation policy M_θ provides a distribution over possible mutations. Rather than having B be generated completely at random, we parameterize M_θ so that it is $M_\theta : \Omega \times \mathcal{F}$, i.e. the mutations conditions on the dataset $\mathcal{D}$ and on the current expression f. The dependance from $\mathcal{D}$ is akin to the approach in inductive DGSR works, while the dependence on f is novel. Both are passed as inputs to M_θ as a sequence of tokens, f by its prefix notation and $\mathcal{D}$ as in [Kam+22]. We use the transformer-based NN architecture from [Kam+22] but task the model to decode token-by-token a sequence ω (flattened version of $< A, op, B >$) until a EOS token is reached. A is represented in ω as the index of that node (i.e., $\in [\![1, n]\!]$ for an expression that contains

Table 7.1 – Set of operators that can be applied on a sub-expression A of a function, with optional argument B as a new sub-expression to include in the tree structure. 0 refers to the root node of the function.

Unary	$A \to \cos(A), A \to \sin(A), A \to \tan(A)$
	$A \to \exp(A), A \to \log(A)$
	$A \to A^{0.5}, A \to A^{-1}, A \to A^2, 0 \to B$
Binary	$A \to A + B, A \to A - B$
	$A \to A * B, A \to A/B$
	$A \to B + A, A \to B - A$
	$A \to B * A, A \to B/A$

n nodes). While this may allow to output an invalid mutation expression, this happens very rarely in practice as shown in Section D.4, thanks to an efficient pre-training of the policy (described below).

We remark that our mutation distribution is different from those that are commonly used in GP, in that the latter are not conditioned on $\mathcal{D}$ nor parameters (i.e., they are not updated via learning), and they can also shrink the size of an expression or keep it as is, whereas our mutations strictly increase the expression. Note that it is possible to consider mutations that remove and/or replace parts of the expression, but we left exploring this to future work. We also restrict our mutation process to only generate expressions with less than 60 operators and without nesting operations other than the basic arithmetic ones $(+, -, \times, \div)$.

Pre-training of M_θ. Since our mutation policy M_θ is expected to produce mutations for a given expression, and not the final expression directly (as it is the case in the majority of `DGSR` approaches), it requires a specifically designed pre-training process. To that end, pre-training labeled examples (dataset & ground-truth expression with up to 10 features) are first sampled from a hand-crafted generator [LC19] as done in most pre-training NSR approaches (c.f. section D.1). Next, given a ground-truth expression f^*, we extract a sequence of mutations $[m_l]_{\leq L}$ that iteratively map the empty expression $f^{(0)}$ to the final expression f^*. As illustrated in Figure 7.1, starting from the ground-truth expressions f^*, we deconstruct f^* by procedurally removing a node (and if the node is binary also one of its child subtree B) from the current f until we get to the empty expression $f^{(0)}$. After reversing this sequence, we obtain a training sequence of expressions and mutations $f^{(0)} \xrightarrow{m_1} f^{(1)} \xrightarrow{m_2} f^{(2)} \xrightarrow{m_3} \dots \xrightarrow{m_L} f^{(L)} = f^*$ (more details in section D.1). After tokenization, every mutation m_l serves as target for the pre-training process: M_θ is classically trained as a sequence-to-sequence encoder-

decoder, using a cross-entropy loss, to sequentially output each token w_l^e from m_l, given the considered dataset $\mathcal{D}$, the previous expression $f^{(l-1)}$, and the sequence of previous tokens $w_l^{(<e)}$ from the target operator m_l.

7.3 Experiments

In this section, we present the results of `DGSR+MCTS`. We begin by studying the performance on test synthetic datasets. Then, we present results on the `SRBench` datasets.

7.3.1 Analysis on synthetic datasets

In this sub-section, we consider a set of 1000 unseen synthetic expressions of which half are *in-domain* (exactly same generator described in section 7.2 and section D.1) and half are *out-of-domain* (bigger expressions with up-to 40 operators instead of 25). We provide a set of explorative experiments to bring insights on how different hyper-parameters contribute to the performance, as well as to select a good configuration of hyper-parameters for evaluation on `SRBench`. In what follows, we always select the best expression on a given dataset by evaluating the accuracy of each expression on the training set; as mentioned before, we consider a dataset to be *solved* if the R^2 achieved by the best expression is greater than 0.99. Pre-training was performed on 8 GPUs for a total time of 12 days. We controlled overfitting on the training set of expressions by i) using a sufficiently large training dataset, ii) controlling the cross-entropy loss and prediction accuracy on a held-out validation set of expressions. Each run in this subsection was obtained by MTCS search trials (which fine-tune M_θ and C_ψ) with a time limit of 24 hours, using 4 trainers (1 GPU/CPU each), 4 MCTS workers (1 GPU/CPU each).

Breadth or depth? First, we analyze whether, given a pre-trained M_θ, it is more desirable to explore in breadth or in depth the search tree. The number of samples implicitly influences the breadth/depth trade-off; the larger the number of samples, the more it will be encouraged to explore, whereas when there is little number of samples K, it is forced to go deep. We run a single search trial of 2000 iterations for different values of $K \in \{1, 2, 8, 16, 32\}$.

To compare these different configurations in a fair manner, we make a few design choices that we justify here. First, as shown in earlier work [DOW20; Kom+20;

Kam+22] and later in this section, optimization of constants contained in the expression can be important to reach high accuracy levels in SR. The deeper in the search tree an expression is, the bigger the expression is, as well as the larger the number of constants it includes, which can add degrees of freedom to the optimization. Because of this, depth can be expected to outperform breadth. For this reason and also because we consider constant optimization in a following ablation, we choose not to optimize constants in this experiment. Secondly, we impose that each configuration considers the same number of expressions. We realize this by allowing only a subset of K expressions to be visited out of the 32 that are sampled during expansion. As shown in table 7.2, a choice of K that is between 8 and 16 seems to be the best compromise in both in-domain and out-of-domain datasets, therefore we will sample $K \in [\![8, 16]\!]$ before each expansion in what follows.

Table 7.2 – Percentage of solved datasets for different K

K	In-Domain	Out-of-domain
1 (greedy)	9.6	0.8
2	10.8	2.4
8	44.6	**19.6**
16	**54.0**	18.4
32	42.8	10.2

Big or small mutation sizes? Secondly, we do ablations on three M_θ models pre-trained on mutation examples generated via different strategies with varying mutation sizes (as defined in section 7.2.3); the higher in the expression tree a node is picked (as described in step a) of the caption of Figure 7.1), the bigger the mutation tends to be. We consider `DGSR+MCTS` @1 (respectively `DGSR+MCTS` @10), a model pre-trained on mutation sizes 1 (respectively approximately[4] 10), and finally `DGSR+MCTS` @∞, a model trained to output the target expression in a single iteration. Note that `DGSR+MCTS` @∞ is essentially reduces to approaches like those in [Big+21; Kam+22], as M_θ is tasked to predict the entire expression f^* from scratch $f^{(0)}$, while updating M_θ with expert-iteration as described in section 7.2.

Interestingly, table 7.3 shows mutation size of 10 performs better than size 1 both in-domain and out-of-domain and that the `DGSR+MCTS` @∞ does not generalize to ouf-of-domain datasets, confirming the importance of search.

[4]Exact mutation size cannot be guaranteed without special expression tree structures.

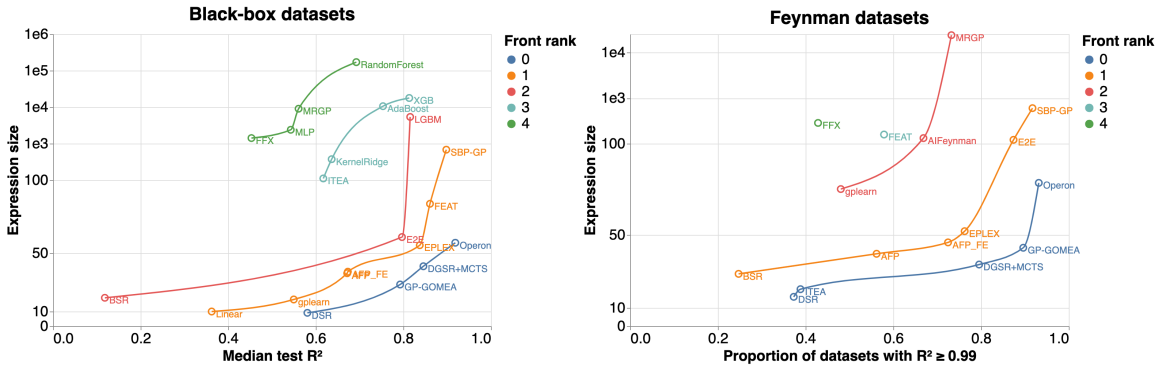


Figure 7.2 – Performance on test splits of `SRBench`, respectively the median R^2 over black box datasets and the proportion of Feynman datasets where the R^2 is larger than 0.99. To nicely visualize the trade-off between accuracy and expression size, we use a linear scale for expression size values up to 100 then a logarithm scale. Note that AI-Feynman [UT20] was removed from the black-box plot for readability (scores $R^2 = -0.6$ and expression size 744).

Table 7.3 – $\%$ **of solved datasets for different mutation sizes.**

Model	In-Domain	Out-of-domain
`DGSR+MCTS` @1	52.2	26.8
`DGSR+MCTS` @10	**74.8**	**44.0**
`DGSR+MCTS` @∞	72.4	16.8

How important is constant optimization? As shown in [Kam+22], optimizing constants predicted by a NN model with an optimizer like BFGS greatly improves performance on `SRBench`. Similarly, we study whether including constants optimization is important for `DGSR+MCTS` in the context of search. We remark that while we use constant optimization to compute accuracy, we store the non-optimized expression in the MCTS search tree. We make this choice because optimized constants may be out-of-distribution w.r.t. the pre-trained M_θ, which can lower its performance. As shown in Table 7.4, optimizing expression constants improves performance substantially. However, constant optimization comes at the price of speed, especially if done after every mutation.

7.3.2 `SRBench` results

We evaluate `DGSR+MCTS` on the regression datasets of the `SRBench` benchmark [La+21a], in particular the *"black-box"* datasets (no ground-truth expression is given) and the *"Feynman"* datasets (conversely, the underlying physics' equation is given).

Table 7.4 – % of solved datasets for different constant optimization strategies. We compare constant optimization done: never, only on the best expression of each trial and after each mutation.

Constant optimization	In-Domain	Out-of-domain
Never	74.8	44.0
Only best expression	77.2	59.4
All expressions	**79.6**	**66.2**

As our approach is trained on datasets with up to 10 features, its use on higher-dimensional datasets requires feature selection. Following [Kam+22], we consider only datasets with at most 10 features so that our results are independent of the quality of a feature selection algorithm. This leads to 57 black-box datasets and 119 Feynman datasets. Each dataset is split into 75% training data and 25% test data using sampling with a random seed (we use 3 seeds per dataset, giving a total of 528 datasets). We consider all baselines provided as part of `SRBench`, which includes GP algorithms, e.g GP-GOMEA [Vir+21], Operon [BKK20b], ITEA [FA20], `DGSR` algorithms, DSR [Pet+19] and `E2E` [Kam+22] as well as classic machine learning regression models, e.g., multi-layer perceptron [Hay94] and XGBoost [CG16].

For each dataset from SRBench, algorithms are allowed a maximum number of expression evaluations (as well as a running time limit). Fixing the number of evaluations allow to avoid algorithms to be compared by their implementation efficiency, e.g. C++ vs Python. [5] We run `DGSR+MCTS` with a budget of $500,000$ evaluations (equivalently mutations) and a maximum time limit of 24 hours. `SRBench` imposes to use at most 500000 evaluations per hyper-parameter configuration, and allows for six configurations, yet we provide a single configuration (resulting from Section 7.3.1); we use the pre-trained M_θ with mutation size 10, with $K \in [\![8, 16]\!]$ and alternate search trials with and without constants optimization.

The performance of all SR algorithms is illustrated in fig. 7.2 along two metrics, accuracy on the test data (as measured by R^2) and expression size computed by counting all operators, variables and constants in an expression, after simplification by SymPy [Meu+17a]. Results are aggregated by taking the average over seeds for each dataset, then the median for black-box datasets and mean for Feynman as done in [La +21a]. We visualize the trade-off between accuracy and simplicity of expressions

[5] This is particularly motivated in cases where expression evaluations are the time bottleneck, as it is the case in GP algorithms. In `DGSR` approaches, the main bottleneck is candidates generation by the action of auto-regressive sampling, therefore the field could benefit from comparison rules updates.

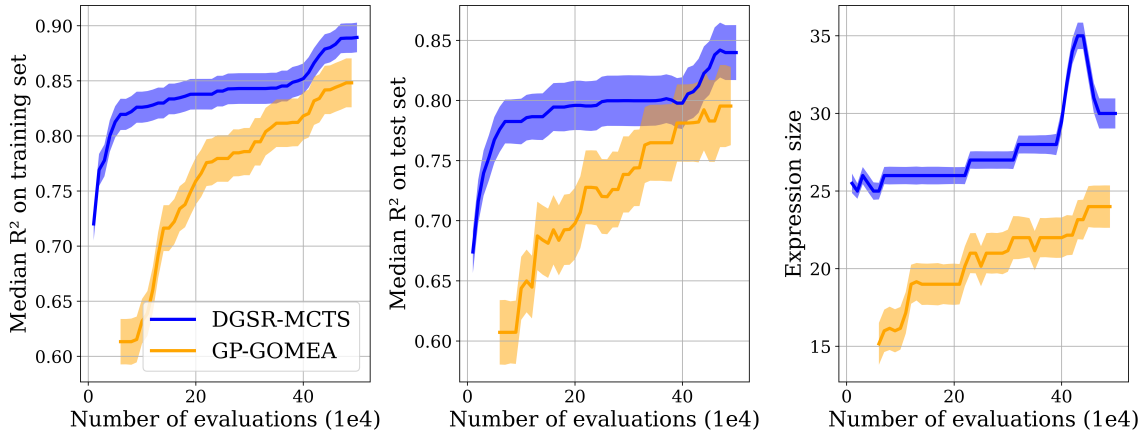


Figure 7.3 – Mean ± confidence interval performance on the black-box datasets over the number of evaluated expressions for `DGSR+MCTS` and its closest competitor, GP-GOMEA, on the black-box datasets. Thanks to pre-training, `DGSR+MCTS` achieves high-levels of R^2 (and larger expressions) much more quickly than GP-GOMEA. On the training set, `DGSR+MCTS` is consistently superior across the entire search process. On the test set and towards the end of the search, `DGSR+MCTS` and GP-GOMEA achieve similar results, due to a larger generalization gap for larger expressions.

obtained by the different algorithms; the lower and righter the better. We compute front ranks by Pareto-dominance. An algorithm Pareto-dominates another if it is not worse in all metrics and it is strictly better in at least one of them. The definition of ranks is then recursive: an algorithm is at rank 0 if there exists no algorithm that Pareto-dominates it; for successive ranks, an algorithm is at rank i if, when excluding the algorithms up to rank $i-1$, there exist no algorithm that Pareto-dominates it. `DGSR+MCTS` is placed on the rank 0-front on both black-box and Feynman datasets. GP-GOMEA and `DGSR+MCTS` seem to be the best approaches for achieving simple-yet-accurate models, and interestingly switch place in their trade-off between the two metrics on the black-box and Feynman datasets. We additionally plot the performance over time on the black-box datasets against this baseline in Figure 7.3. Another interesting point is the difference between `DGSR+MCTS` and `E2E`; `DGSR+MCTS` achieve better test accuracy (0.846 and 0.797 respectively) with less complex expressions (41 and 61 respectively) on the black-box datasets. On 80% (resp. 87% for `E2E`) of Feynman datasets, we achieve $R^2 \geq 0.99$ with expression sizes of 33 (resp. 121).

Ablations. Finally, we present several ablations in Table 7.5. Namely, we observe whether using synthetic datasets for training C_ψ and fine-tuning M_θ is better than not doing so, and whether our strategy of training `DGSR+MCTS` simultaneously on multiple

datasets is better than training iteratively, i.e., one dataset at the time. Our findings suggest that utilizing synthetic datasets has a positive effect on the performance of our model, particularly on the Feynman datasets, which may be attributed to the similarities between the synthetic and Feynman datasets (similar expression sizes, non-noisy observations...) On the black-box dataset (real-world scenarios), training on all datasets simultaneously appears to result in better performance than using synthetic datasets alone, likely due to the sharing of gradients across multiple datasets. Overall, our results indicate that both components play a significant role in the strong performance of `DGSR+MCTS`.

Table 7.5 – Ablations for different training configurations. We report the same performance metrics used in fig. 7.2. Respective expression sizes (not shown here) remain similar.

Use synthetic datasets	Simultaneous training	Black-box	Feynman
no	no	0.801	0.655
yes	no	0.812	0.748
no	yes	0.823	0.689
yes	yes	**0.846**	**0.796**

7.4 Limitations

Our approach is subject to the known limitations of the Transformer models as in [Kam+22]. For instance, learning on large context lengths is challenging and necessitates significant GPU memory resources. This limitation could be circumvented by the use of Transformers specifically designed for large inputs, such as LongFormer [BPC20]. The main source of latency in our algorithm comes from sampling mutations (i.e. forwarding the Transformer model), an aspect shared by other DGSR methods that use large pre-trained transformers. In our case, better sample efficiency is at the cost of latency. While GP approaches can run on CPU only, GPUs can be useful to batch computations (to generate mutations in our case) for DGSR methods.

7.5 Conclusion

In this work, we introduced a competitive SR algorithm that address the limits of `DGSR` by incorporating search as a tool to further improve performance of inductive

DGSR approaches. We showed DGSR+MCTS performs on par with state-of-the-art SR algorithms such as GP-GOMEA in terms of accuracy-complexity trade-off, while being more efficient in terms of number of expression evaluations.

Future work may concern the extension of the proposed approach in a meta-learning framework [Sch87; TP12; FAL17], where pre-training is performed, either via self-supervised or reinforcement learning, with the objective to reduce the required search budget on a broad family of target datasets.

Chapter 8

Symbolic Model-Based Reinforcement Learning

We investigate using symbolic regression (SR) to model dynamics with mathematical expressions in model-based reinforcement learning (MBRL). While the primary promise of MBRL is to enable sample-efficient learning, most popular MBRL algorithms rely, in order to learn their approximate world model, on black-box over-parametrized neural networks, which are known to be data-hungry and are prone to overfitting in low-data regime. We leverage the fact that a large collection of environments considered in RL is governed by physical laws that compose elementary operators e.g $\sin$, $\sqrt{}$, $\exp$, $\frac{\mathrm{d}}{\mathrm{dt}}$, and we propose to search a world model in the space of interpretable mathematical expressions with SR. We show empirically on simple domains that MBRL can benefit from the extrapolation capabilities and sample efficiency of SR compared to neural models.

8.1 Introduction

Most control systems, irrespective of the task to solve (i.e. rewards maximization) have in common the fact that their dynamics are governed by physical laws. They are usually expressed with mathematical equations connecting the next state with the system's past states and controller's actions with operators such as time-derivatives, trigonometric operators, power functions. Solving a task usually involves implicitly a good understanding of the dynamics, e.g. goal reaching.

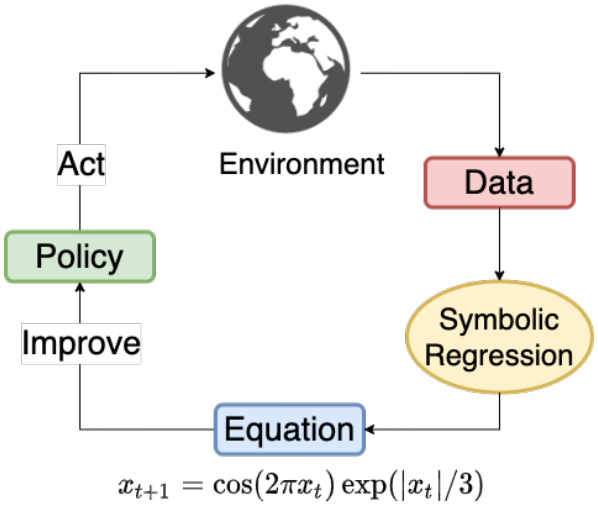


Figure 8.1 – Illustration of Symbolic-MBRL

For instance, in the classic CartPole environment [BSA83; Ope+21], the system's state is described by $(x, \dot{x}, \theta, \dot{\theta})$ where x (resp. $\dot{x}$) denotes the position (resp. speed) of the cart along the x-axis and θ (resp. $\dot{\theta}$) is the angle (resp. rotation) of the pole w.r.t the cart. The agent's action a affects the system's state according to the following equations [Ope+21; Flo07]:

$$\ddot{\theta} = \frac{g\sin\theta + \cos\theta\left(\frac{-K_{\text{mag}}a - m_p l\dot{\theta}^2\sin\theta}{m_c+m_p}\right)}{l\left(\frac{4}{3} - \frac{m_p\cos^2\theta}{m_c+m_p}\right)} \qquad \ddot{x} = \frac{K_{\text{mag}}a + m_p l\left(\dot{\theta}^2\sin\theta - \ddot{\theta}\cos\theta\right)}{m_c+m_p} \tag{8.1}$$

where $g, K_{\text{mag}}, m_p, m_c, l$ are all constants. Systems with impossible states, safety or physical constraints, e.g. the joint of a robotic arm cannot exceed a certain angle [TET12], can be expressed via piece-wise expressions. When the environment is stochastic, probabilistic distributions appear. Note that the parametrization of the state and action spaces has an impact on the symbols that are necessary to describe the system dynamics; for instance, if $\dot{x}$ and $\dot{\theta}$ were not in the CartPole agent's observation, derivative operators would have to appear in the equation. Generally, the larger the state-space parametrization, the "shorter" equation is.[1]

Model-Based Reinforcement Learning (MBRL) involves a two-step procedure repeated until task termination; a) learn from data a forward dynamics model (possibly stochastic) function f that maps current state s_t and action a_t to next state s_{t+1} and b) derive a policy from this model. Though they have been shown to learn faster

[1]The best parametrization to decrease the complexity of predicting an equation from data is an open problem.

than model-free algorithms theoretically [Efr+19] and in certain applications [DFR13; LA14], MBRL algorithms are often hard to train in practice [Pin+21]. In this work, we challenge the go-to approach of using over-parametrized feed-forward neural networks to approximate f as they are prone to overfitting when collected data does not have enough coverage of state and action spaces. We propose to leverage prior knowledge on operators that *could* appear in the environment dynamics equations, e.g. $\sin, \sqrt{}, \exp, \frac{\mathrm{d}}{\mathrm{dt}}$.

Manipulating expressions to fit data is exactly the objective of SR algorithms that select f within a large family of expressions through composition of operators, constants and variables, as opposed to gradient-descent of over-parametrized models, e.g. neural networks (NNs). The latter have more degrees of freedom and are easier to optimize, but prone to overfitting in low-data regimes, whereas SR has recently shown excellent extrapolation capabilities [La +21a; Kam+22; Kro+22]. Furthermore, SR provides an interpretable and differential model, interesting properties for RL, which we develop in Chapter 9.

Contributions. We propose a novel approach to dynamics modelling in control problems, which we call Symbolic-Model-Based RL, that uses mathematical expressions to model dynamics. To our knowledge, this is the first work that proposes to leverage SR to find an interpretable function f that best maps state-action pairs (s_t, a_t) to the next state s_{t+1}. We provide empirical evidences in simple domains, where our method largely outperforms the over-parametrized approaches, that SR provides faster better dynamics models that generalize to unseen states-action pairs.

8.2 Background

MBRL. MBRL algorithms ground control policies on a model of the dynamics of environment. Most of approaches alternate two steps repeatedly: i) collect data $\mathcal{D}$ with the current policy and learn an approximate model f of the environment's dynamics, fitting $\mathcal{D}$ as in supervised learning (SL), i.e.:

$$f^* = \arg\max_{f \in \mathcal{F}} \mathbb{E}_{(s_t, a_t, s_{t+1}) \sim \mathcal{D}} \mathcal{L}(s_{t+1}, f(s_t, a_t)) \tag{8.2}$$

where $\mathcal{F}$ is a family of functions, e.g. neural networks or Gaussian processes, and $\mathcal{L}$ a loss function that depends on the nature of f; ii) then, simulate transitions with

the model f and optimize the policy accordingly. Note that in this work we consider reward and termination functions learned in the same way as dynamics, even though MBRL algorithms [Chu+18; CBK20] often consider them provided to the agent.

We refer the reader to an exhaustive MBRL review [MBJ20]. Step i) usually faces classic under/over-fitting issues of SL (w.r.t the state and action space), causing sub-optimal task performance. The design of predictive model has proven to be very challenging; Gaussian Processes (GPs) can under-fit on complex dynamical systems [Cal+16], whereas over-parametrized functions, i.e. neural networks, can express complex (and high-dimensional) dynamics, but are prone to overfitting. To avoid overfitting, one can a) acquire more data, but this often comes with exploration challenges, b) use regularization, i.e. making the model simpler, in the form of priors, e.g. GPs' kernel function or Bayesian neural networks [Blu+15; GHK17], or c) use ensembles [ET93; Osb16; Kur+18]. [Chu+18] uses probabilistic NNs (b) in combination with ensembles (c) to make MBRL agents uncertainty-aware for better planning.

SR in RL. SR has been applied to model-free RL to learn interpretable policies explicitly [Lan+21; Pet+21; Vid+22] or via the associated value function [Kub+21], but to our knowledge we are the first to consider SR for MBRL. In addition to the immediate interpretability benefit, considering the dynamics model search space $\mathcal{F}$ in Eq. 8.2 to be the family of short mathematical expressions that contain constants, variables and operators from a given dictionary has the advantages of injecting prior knowledge, smoothness properties, as well as to significantly reduce the size of the search space.

8.3 Experiments

Description of the algorithm. We propose to use an expression optimized via SR, instead of the usual NN dynamics model, to fit data in a MBRL's algorithm (step i)); though all SR algorithms are applicable to most MBRL algorithm, we use:

- Probabilistic Ensembles with Trajectory Sampling (PETS) [Chu+18] implemented with *MBRL-lib* [Pin+21] as our base MBRL algorithm. PETS learns an ensemble of probabilistic world models; pratically NNs that condition on state and action pairs and predict the mean and variance of a distribution over the

next state and reward. At any given timestep, it selects the sequence of actions that maximize rewards using the cross-entropy method [Bot+13] and trajectory sampling on the learned dynamic models. As in the original PETS paper, we maintain an ensemble of 7 dynamics models. We call this model *Symbolic-PETS* and compare it to *MLP-PETS*, the original version of PETS.

- This work was done prior to previous contributions explained in Chapters 6 and 7, therefore we used Operon [BKK20a], the state-of-the-art GP algorithm, as our base SR algorithm. Preliminary experiments on the considered environments, showed that Operon had better performance that numerous SR algorithms, e.g. gplearn [Ste16], PS-Tree [Zha+22] and AI-Feynman [UT20]; they either had inaccurate predictions, overly complex expressions or inference was too long.

Our experiments investigate the following questions: **i**) Can SR algorithms help learn predictive models with less samples, **ii**) Do the obtained dynamics equations have interesting properties? **iii**) How is this manifesting in terms of performance (task solving)? We consider deterministic environments, which already represent a substantial part of environments in the RL literature [TET12; Tas+18; Fre+21].

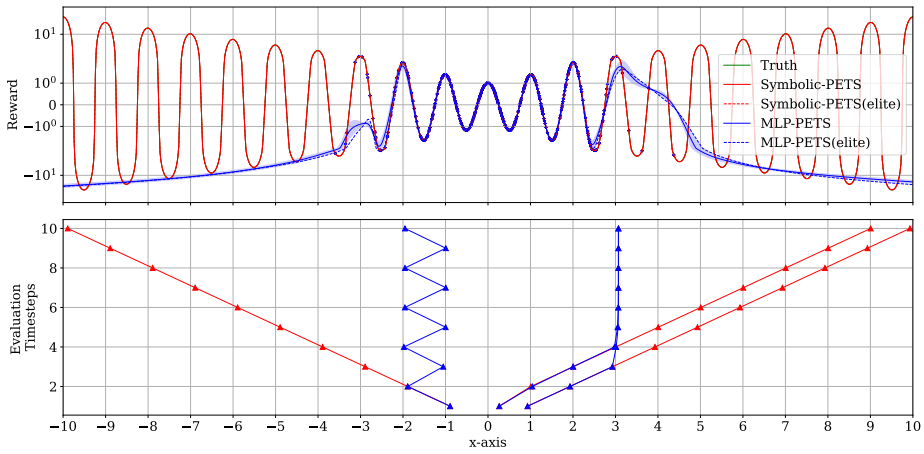


Figure 8.2 – We train the dynamics models on 500 transitions collected by a random (uniform $[-1, 1]$). The top row is the immediate reward function predicted by learned models (evaluated with $a_t = 0$ for clarity) and dots correspond to training data. Elite is the best dynamics model w.r.t to an evaluation set. The bottom row represents 3 evaluation roll-outs after the predictive model was updated:we observe that Symbolic-PETS allows agents to reach better rewarded states.

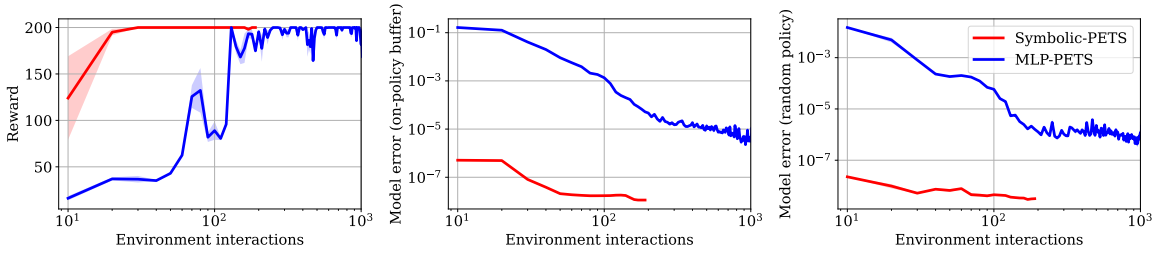


Figure 8.3 – Symbolic-PETS solves CartPole *very* fast. Agents are evaluated on 3 episodes with 3 random seeds every 10 transitions. MLP-PETS solves CartPole despite significant model error, suggesting that solving CartPole does not require a perfect understanding of the environment.

An illustrating example. For illustrative purposes, we consider a simple one-dimensional state MDP where an agent moves on the horizontal axis x while observing its position, with episode length 10 and the following dynamics:

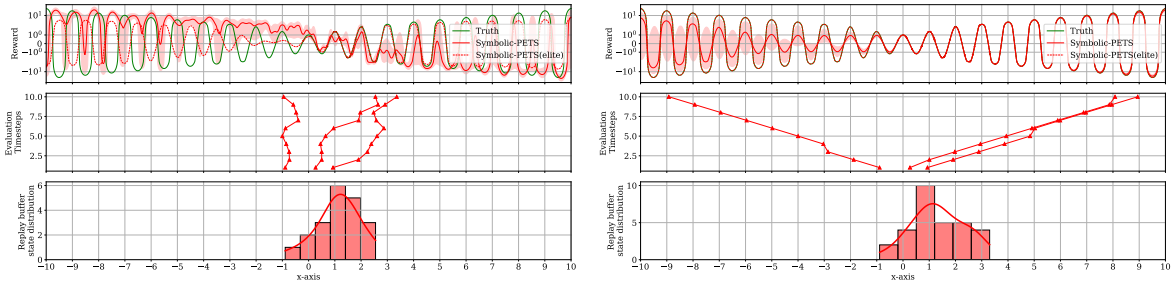
$$s_{t+1} = s_t + a_t, \quad r_t = \cos(2\pi s_{t+1}) \exp(|s_{t+1}|/3) \tag{8.3}$$

As illustrated on Fig. 8.2, solving this MDP is challenging as it requires sufficient exploration to learn the reward function and avoid falling into local minimas, i.e. staying in $x \in \mathbb{N}$. As shown in Fig. 8.2, MLP-PETS' dynamics models over-fits on the training distribution. Even using an ensemble of 7 models, the uncertainty of the ensemble is not good enough to be leveraged for efficient exploration, leading to a sub-optimal policy. On the other hand, Symbolic-PETS' dynamics models extrapolate really well on unseen states, thus achieving optimal behavior in just one update. Symbolic-PETS's predicted reward function is:

$$(1.0 \exp(|0.333s_t + 0.333a_t| + 2.14e^{-4}) \sin(6.283x_t + 6.283a_t - |0| - 4.712)$$

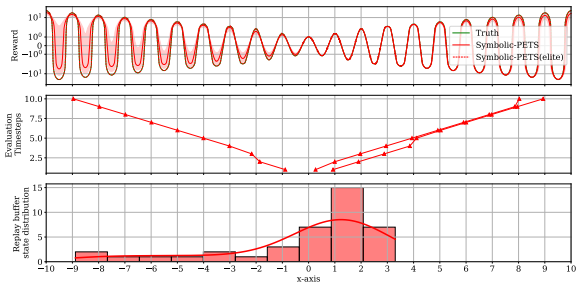
We present, in Figure 8.4, the evolution of Symbolic-PETS every episodes after just 20 observed random transitions.

CartPole. We consider the continuous CartPole [Ope+21], where the agent state is $s_t = (x_t, \theta_t, \dot{x}_t, \dot{\theta}_t)$. As in [Chu+18; Pin+21], the termination and reward function are made available to the agent, therefore control is restricted to be a problem of dynamics modelling. We define the model error $\mathcal{L}$ in Equation (8.2) as the MSE averaged over output dimensions.

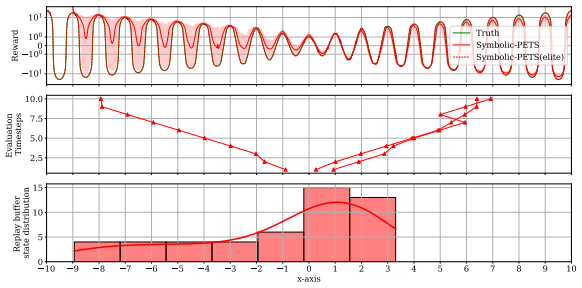


(**a**) 20 environments steps

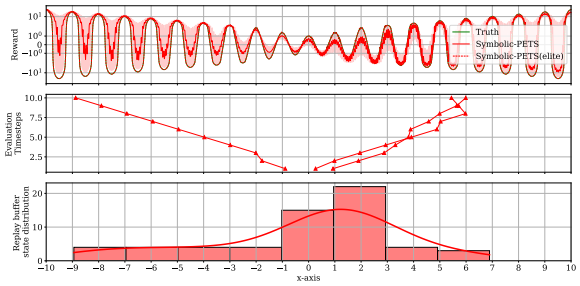
(**b**) 30 environments steps



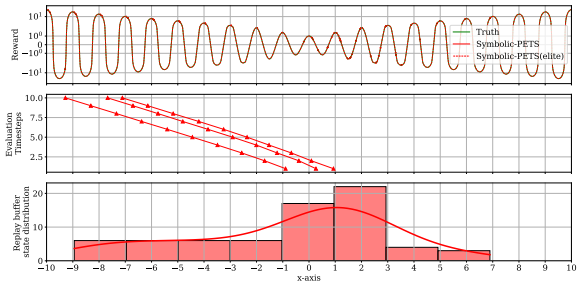
(**c**) 40 environments steps



(**d**) 50 environments steps



(**e**) 60 environments steps



(**f**) 70 environments steps

Figure 8.4 – Top row is the reward function evaluated with $a_t = 0$ for clarity learned by the Symbolic-PETS agents (elite is the best dynamics model w.r.t to an evaluation set). Middle rows represents 3 evaluation roll-outs after a predictive model update. Bottom row is the training replay buffer state distribution.

In a first experiment, we explore the capabilities of both symbolic and MLP regressors on the data generated by a random policy on CartPole. We collect an evaluation dataset of $5e^4$ transitions, enough to have good state-action coverage. We then train the two dynamics models on training datasets of growing sizes and plot the model error in Fig. E.1). The symbolic model predicts the following equations with the accuracy reached by the MLP in two order of magnitude less interactions:

$$
\begin{aligned}
x_{t+1} &= x_t + 0.02\dot{x}_t \\
\theta_{t+1} &= \theta_t + (0.02\dot{\theta}_t + 0.015)/\cos(0.035 * \dot{\theta}_t) - 0.015 \\
\dot{x}_{t+1} &= (0.002\theta_t + 2.34e^{-4}\dot{\theta}_t a_t + 1.0) \times (\dot{x}_t + 0.195a_t - \sin(0.015\theta_t) + 3.23e^{-5}) \\
\dot{\theta}_{t+1} &= \cos(0.195\theta_t)(0.314\theta_t + \dot{\theta}_t - 8.97e^{-1}a_t \times (-0.031\dot{\theta}_t - 2.014)\frac{(0.016\dot{\theta}_t - \cos(1.053\theta_t))}{(6.173 - 0.002\theta_t)})
\end{aligned}
\tag{8.4}
$$

Interestingly, predicted equations are a bit different than in Equation (8.1), though we can notice constants such as the time-discretization interval 0.02. What could be as missing terms can be explained by limited development as θ and x have small values because of CartPole's constraints. In Fig 8.3, we demonstrate that Symbolic-PETS is able to solve CartPole in just 20 interactions with the environment, a *state-of-the-art* performance to our knowledge.

8.4 Conclusion

We demonstrated on simple environments that SR can learn better predictive models with many less samples than neural network and that they extrapolate well to unseen state-action pairs much better. Though [La +21a] showed promising results on real-world regression datasets with input dimensions up to 124 (with a single output dimension), SR still remains to be scaled to higher output dimensions, with challenges including parallelizing regressor training of each output dimension or pixel observations. Note that it yet remains to try `DGSR` techniques including those introduced in previous chapters in the control setting. We will discuss in greater details in Chapter 9 interesting research directions that leverage interpretable world models.

Chapter 9

General Conclusion and Perspectives

9.1 Conclusions on our Contributions

This doctoral thesis addresses the distribution-shift problem encountered by RL agents within the meta-RL framework, aiming to equip them with the ability to adapt to unseen scenarios, whether they arise within the same environment or in different but similar ones. The research investigates two primary approaches for developing adaptable agents. In Part I, the focus lies on developping model-free adaptation mechanisms that facilitate learning at the time of testing. Chapter 3 leverages training tasks to establish an inductive bias on trajectory representations, enabling efficient expression of the encountered task. Chapter 4, on the other hand, concentrates on devising exploration strategies to achieve comprehensive coverage of the state-space, especially in situations where no extrinsic reward is accessible before test-time. Moving on to Part II, the study delves into the utilization of SR to develop predictive dynamics models that exhibit robustness, accuracy, and the capacity to learn from minimal interactions with the environment. This aspect is particularly applied to model-based agents and enrich the field of SR with a collection of novel generative models.

9.2 Perspectives

We conclude the present manuscript by trying answering a set of questions and raising new ones.

What are the underlying assumptions behind algorithms in Part I?

Episodic setting. Both chapters in Part I operate under the assumption that the environment is episodic, wherein the agent engages in a series of interactions before the state is reset to an initial state sampled from the distribution P^0. Notably, this assumption carries particular significance in the context of learning in the physical world, as it necessitates human supervision to intervene and return the agent to its initial state.

Access to a simulator. They also enable agents to undergo **online training** facilitated by access to a simulator and a set of training tasks, which may employ the same or different dynamics compared to the test environments. It is essential to highlight that the prevailing focus in meta-reinforcement learning research has been on the utilization of online data. Consequently, certain scenarios may restrict agents from engaging in unrestricted exploration.

How can we directly improve algorithms from Part I?

Intrinsic rewards for IMPORT. In Chapter 3, the IMPORT algorithm learns *task embeddings* by mapping the history to an embedding which is then used by the policy as an input. The mapping is implemented by a recurrent NN, thus making the overall policy a POMDP policy. The embedding is learned by minimizing the distance w.r.t a privileged embedding learned from an informed policy that conditions on a task identifier; being primarily learned with an inductive bias via an auxiliary loss, the recurrent policy lacks the incentive to explore to reconstruct the privileged embedding. To address this limitation, one approach is to introduce intrinsic rewards to encourage reconstruction. At test time, the agent does not have access to the privileged embedding, so it uses the policy that maximised the sum of intrinsic and extrinsic rewards at train time. The problem is informed policies can learn privileged embeddings (therefore intrinsic rewards) that collapsed, i.e. whose lost precious information that distinguishes training tasks. For instance, if the goal information is only available in the last state of a sequence, all the information on how to reach that state will not be adequately captured in the intrinsic reward signal. Therefore, one could construct privileged embeddings that capture the sequence of subgoals required to clearly identify each task.

Non-stationarities. As mentioned in the preceding question, IMPORT is trained with prior knowledge of the environment changes. Consequently, the model is not

explicitly trained to identify changes in tasks within an episode, assuming the task remains constant throughout the entire episode. To accommodate non-stationary environments within episodes, one could extend IMPORT's training process to include such scenarios.

Symbolic representations. Similarly to *task inference* approaches who explictly model reconstruction of the task identifier, it is worth exploring the utilization of symbolic representations of trajectories using SR. This entails learning model-free policies that condition their decisions on symbolic information extracted from the trajectory.

Offline training of UPSIDE. To relax the assumption of a simulator to learn online, one could leverage offline training in cases where agents may be granted access to offline data collected prior to the task. Effectively leveraging such data could be a valuable approach in mitigating the demand for costly online data collection during test-time.

Better local coverage. In UPSIDE (Chapter 4), a random policy is employed for the diffusing part, which is effective in locally covering the state-space for certain environments like 2D mazes but proves inadequate for more challenging control tasks like Hopper or Ant. An alternative approach would be to learn per-task local coverage policies specifically tailored for the diffusing phase.

Graph-based exploration strategies. To address the issue of resets discussed in the previous question, an extension of UPSIDE could involve the adoption of graph-based policies rather than tree-based policies. Graph-based policies would enable the agent not only to reach a specific goal from the initial state but also to reverse the process, allowing for more comprehensive problem-solving capabilities.

How can symbolic regression contribute to the field of Meta-Reinforcement Learning?

In Chapter 8, we have explored the application of symbolic world models in toy control problems and demonstrated that using SR to derive interpretable world models can significantly improve the sample efficiency of model-based RL agents. Additionally, the use of SR helps avoid suboptimal performance by leveraging its extrapolation capabilities.

However, the full potential of analytical expressions in Meta-RL is yet to be fully uncovered.

Agents Acquiring differentiable analytical expressions facilitates policy optimization by enabling gradient backpropagation through the world model. Moreover, in scenarios such as system identification of dynamical systems, i.e. where a world model structure is given, learning primarily comes down to identifying numerical parameters. SR removes the need for prior knowledge of the system dynamics, but once an accurate analytical expression has been found, improving the world model can reduce to system identification by reducing search to numerical parameters. Non-stationarities are a common phenomenon in real-world settings, such as robotics; various factors like wear and tear of actuators, temperature variations, sensor noise, and failures can occur. In such situations, parameter identification can further enhance the sample efficiency of SR compared to the efficiency of updating neural network weights when the structure of the expression is correct. This is particularly applicable to parametrized MDPs (as introduced in Chapter 2), where tasks differ based on factorizing parameters.

Acquiring dymamic simulators. Learning a world model from data offers significant advantages, even when an associated reward function is absent. Assuming that symbolic regression (SR) produces an accurate expression, from interactions, with strong generalization, one can exploit this interpretable expression in various ways, thereby contributing to the field of environment design [Par+22]. Firstly, a comparative analysis can be conducted between the predicted expression derived from SR and the ground-truth expression to identify unknown physics parameters. Secondly, by establishing a simulator that aligns with the learned world model, it becomes feasible to generate novel environments with diverse dynamics, incorporating different numerical parameters and additional factors such as friction. These varied environments, originating from the initial expression, play a pivotal role in the pursuit of developing agents with broad generalization abilities, excelling across a wide spectrum of tasks; we think a curriculum or adversarial learning could be useful in training agents on increasingly harder tasks. Notably, the field of sim2real has embraced this approach, employing domain randomization techniques [Tob+17] on learned neural network weights to bridge the gap between simulation and real-world performance. Lastly, the derived expression can serve as a foundation for establishing safety constraints within the system, ensuring secure and reliable operation.

What are the missing pieces for symbolic regression to be successfully applied to meta-RL?

Representation capacity. In chapter 8, the control environments were limited to a maximum of 4 features. Consequently, it is essential to investigate how SR can perform with higher-dimensional inputs, particularly when dealing with pixel-based observations, which are prevalent in the RL literature. There preliminary results were obtained using a GP-based algorithm, therefore one should evaluate how `DGSR` methods perform in these domains. Moreover, it is worth noting that SR has not undergone testing more complex dynamics with e.g. piece-wise definitions, matrix multiplication, or cross-products, which are commonly encountered in robotics. Expanding the set of operators to handle these complexities is necessary, though no tests have been conducted on this aspect so far. Many control environments of interest entail stochastic dynamic functions, and therefore, SR must address the inherent stochasticity of these environments to achieve robust performance. Additionally, when dealing with partially observable environments, where certain elements like velocity remain unobserved, augmenting the operator set with derivatives could prove beneficial.

Compute desirements. In MBRL, updating a world model parametrized by a NN consists of simple gradient steps on the newly acquired data. However, fitting a SR algorithm on a new dataset is order of magnitudes longer (especially as the dynamics is complex and required accuracy threshold is high). DGSR goes into this direction by leveraging inductive bias to restrict the search space, therefore reducing the fitting time, however this needs to be even made faster. We need efficient implementation for SR to parallelize search over the different output dimensions, as well as to improve SR search to let the model improve from the previous model when newly sampled transitions are collected. This approach allows the model to build upon the knowledge gained from the previous model, leading to meaningful improvements instead of starting the learning process from scratch each time.

How to improve deep generative symbolic regression?

We have identified several potential avenues for enhancing DGSR, warranting further exploration. Regarding the improvement of the base Transformer model, several strategies can be pursued:

1. Considering the adoption of more advanced and contemporary Transformer architectures [BPC20; Ain+23] which could enable the model to condition on

larger contexts. This can potentially enhance the model's capacity to capture complex patterns and dependencies in the data.

2. Exploring alternative tokenization strategies for real numbers and expressions which could lead to more efficient representations, thereby improving the model's ability to interpret numerical and symbolic information effectively.

3. Investigating the application of curriculum learning to put more emphasizis on examples that yield inaccurate predictions, by strategically introducing more challenging examples as training progresses.

4. Addressing the enhancement of test-time predictions. While MCTS was considered in Chapter 7, other RL techniques could be explored to update the expression sampling distribution, similarly to RLHF.

Regarding data improvements, we propose two avenues for consideration:

1. Devoting more effort to construct improved synthetic datasets. A well-designed and diverse synthetic dataset can be a valuable asset in training the model effectively, capturing a wide range of expression structures and improving generalization.

2. Tailoring synthetic datasets to specific problem domains of interest. For instance, in cases where the structure of expressions is roughly known, generating a more specialized synthetic dataset predominantly consisting of expressions with this structure can be beneficial. This approach can limit the search space, providing a performance boost compared to training on a highly diverse set of expressions.

These improvements hold promising directions in advancing DGSR and expanding its applicability to real-world problem-solving tasks. Specialists, in particular, may find the constrained search space approach to be highly valuable, contributing to more efficient and targeted model performance. Further research and experimentation in these areas are expected to enrich the potential of DGSR and foster the advancement of symbolic model-based Reinforcement Learning (RL) techniques, while simultaneously yielding interpretable world models as valuable by-products.

Part III

Appendix

Appendix A

Complements on Chapter 3

A.1 The IMPORT algorithm

The algorithm is described in details in Algorithm A.1. In our implementation, the value function network used for **(A)** and **(B)** is the same, i.e. shared. We specialize the input, i.e. for **(A)** the input will be $(s_t, f_H(\tau_t))$ and $(s_t, f_\mu(\mu_t))$ for **(B)**.

A.2 Implementation details

A.2.1 Data collection and optimization

We focus on on-policy training for which we use the actor-critic method A2C [Mni+16] algorithm with generalized advantage estimation. We use a distributed execution to accelerate experience collection. Several worker processes independently collect trajectories. As workers progress, a shared replay buffer is filled with trajectories and an optimization step happens when the buffer's capacity bs is reached. After model updates, replay buffer is emptied and the parameters of all workers are updated to guarantee synchronisation.

A.2.2 Network architectures

The architecture of the different methods remains the same in all our experiments, except that the number of hidden units changes across considered environments

Algorithm A.1: Details of IMPORT Training

Initialize $\sigma, \omega, \theta, \nu$ arbitrarily
Hyperparameters: Number of iterations K, Number of transitions per update steps M, discount factor γ, GAE parameter γ_{GAE}, Adam learning rate η, weighting of the (**C**) objective β, weighting of the entropy objective λ_h, weighting of the critic objective λ_c
Optim = Adam(η)
for $k = 1, \dots, K$ **do**
  if *k is odd* **then**
    Collect M transitions according to π_H in buffer B_H.
  else
    Collect M transitions according to π_μ in buffer B_μ.
  $\delta_\sigma, \delta_\omega, \delta_\theta = 0, 0, 0$
  $R^\mu \leftarrow \text{compute\_gae\_returns}(B_\mu, \gamma_{GAE})$
  $R^H \leftarrow \text{compute\_gae\_returns}(B_H, \gamma_{GAE})$
  $\delta_{\theta,\omega} \mathrel{+}= \frac{1}{|B_H|} \sum_{b \in B_H} \sum_{t=1}^{T} [R_t^{\mu,b} - V_\nu(s_t^b, z_t^b)] \nabla_{\theta,\omega} \log \pi_H(a_t^b | s_t^b, z_t^b)$
  $\delta_{\theta,\omega} \mathrel{+}= \frac{\lambda_h}{|B_H|} \sum_{b \in B_H} \sum_{t=1}^{T} \nabla_{\theta,\omega} H\Big(\pi_H(a_t^b | s_t^b, z_t^b)\Big)$
  $\delta_\omega \mathrel{-}= \frac{2\beta}{|B_H|} \sum_{b \in B_H} \sum_{t=1}^{T} [f_H^\omega(s_t^b, z_t^b) - f_\mu(s_t^b, \mu_t^b)] \nabla_\omega f_H^\omega(s_t^b, z_t^b)$
  $\delta_\nu \mathrel{-}= \frac{2\lambda_c}{|B_H|} \sum_{b \in B_H} \sum_{t=1}^{T} [R_t^{H,b} - V_\nu(s_t^b, z_t^b)] \nabla_\nu V_\nu(s_t^b, z_t^b)$
  $\delta_{\theta,\sigma} \mathrel{+}= \frac{1}{|B_\mu|} \sum_{b \in B_\mu} \sum_{t=1}^{T} [R_t^{H,b} - V_\nu(s_t^b, \mu_t^b)] \nabla_{\theta,\sigma} \log \pi_\mu(a_t^b | s_t^b, \mu_t^b)$
  $\delta_{\theta,\sigma} \mathrel{+}= \frac{\lambda_h}{|B_\mu|} \sum_{b \in B_\mu} \sum_{t=1}^{T} \nabla_{\theta,\sigma} H\Big(\pi_\mu(a_t^b | s_t^b, \mu_t^b)\Big)$
  $\delta_\nu \mathrel{-}= \frac{2\lambda_c}{|B_\mu|} \sum_{b \in B_\mu} \sum_{t=1}^{T} [R_t^{\mu,b} - V_\nu(s_t^b, \mu_t^b)] \nabla_\nu V_\nu(s_t^b, \mu_t^b)$
  $\theta \leftarrow Optim(\theta, \delta_\theta)$
  $\omega \leftarrow Optim(\omega, \delta_\omega)$
  $\sigma \leftarrow Optim(\sigma, \delta_\sigma)$
  $\nu \leftarrow Optim(\nu, \delta_\nu)$

and we consider convolutional neural networks for the Maze3d environment. A description of the architectures of each method is given in Fig. 3.2.
Unless otherwise specified, MLP blocks represent single linear layers activated with a $tanh$ function and their output size is hs. All methods aggregate the trajectory into an embedding z_t using a GRU with hidden size hs. Its input is the concatenation of representations of the last action a_{t-1} and current state s_t obtained separately. Actions are encoded as one-hot vectors. When episodes begin, we initialize the last action with a vector of zeros. For bandits environments, the current state corresponds to the previous reward. TS uses the same GRU architecture to aggregate the history into z_t.

All methods use a $softmax$ activation to obtain a probability distribution over actions.
The use of the hidden-state z_t differs across methods. While **RNNs** only use z_t as an input to the policy and critic, both **TS** and **TI** map z_t to a belief distribution that is problem-specific, e.g. Gaussian for control problems, Beta distribution for bandits, and a multinomial distribution for Maze and CartPole-task environments. For instance, z_t is mapped to a Gaussian distribution by using two MLPs whose outputs of size $|\mu|$ correspond to the mean and variance. The variance values are mapped to $[0, 1]$ using a $sigmoid$ activation.

IMPORT maps z_t to an embedding f_H, whereas the task embedding f_μ is obtained by using a $tanh$-activated linear mapping of μ_t. Both embeddings have size hs_μ, tuned by cross-validation onto a set of validation tasks. The input of the shared policy head ϕ is the embedding associated with the policy to use, i.e. either f_H when using π_H or f_μ when using f_μ.

For the Maze3d experiment and in all methods, we pre-process the pixel input s_t with three convolutional layers (with output channels 32, stride is 2 and respective kernel sizes are 5, 5 and 4) and LeakyReLU activation. We also use a batch-norm after each convolutional layer. The output is flattened, linearly mapped to a vector of size hs and $tanh$-activated.

A.3 Experiments

In this section, we explain in deeper details the environments and the set of hyper-parameters we considered. We add learning curves of all experiments to supplement results from Table 3.1, 3.2, 3.3 and A.2 in order to study sample efficiency.

Task descriptor. Note that for CartPole and Acrobot μ is normalized to be in $[-1, 1]^D$ where D is the task descriptor dimension. The task distribution q is always uniform, see the description of the environments for details. For experiments with task identifiers, we associate to each sampled task an integer value corresponding to the order of generation, and encode it usong a one-hot vector.

HPs	CartPole	Acrobot	Bandits	TMDP	Maze3d
E	16	128	16	16	32
Tr	20	20	20	20	20
hs	16	32	16	64	32
hs_μ	$\{2, 4, 8, 16\}$	$\{2, 4, 8, 16\}$	16	$\{16, 32\}$	$\{2, 16, 32\}$
γ	0.95	0.95	0.90	0.90	0.90
λ_h	$\{1., 1e^{-1}\}$				$\{1e^{-1}, 1e^{-2}, 1e^{-3}\}$
γ_{GAE}	$\{0.0, 1.0\}$				
clip gradient	40				
η	$\{1e^{-3}, 3e^{-4}\}$				
λ_c	$\{1., 1e^{-1}, 1e^{-2}\}$				
β	$\{1e^{-1}, 1e^{-2}, 0.\}$				

Table A.1 – Hyperparameters tested per environments. At each training epoch, we run our agent on E environments in parallel collecting Tr transitions on each of them resulting in batches of $M = E * Tr$ transitions.

Hyperparameters. Hyperparameter ranges are specified in Table A.1. For TS, we consider sampling μ from the posterior dynamics distribution every k steps with $k \in \{1, 5, 10, 20\}$.

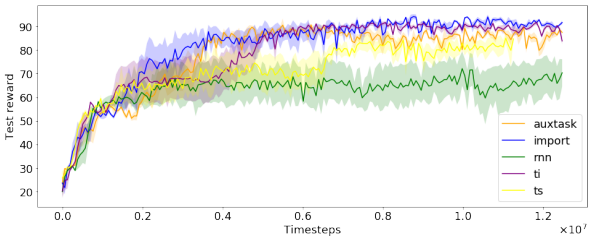
A.3.1 CartPole.

We consider the classic CartPole control environment where the environment dynamics change within a set $\mathcal{M}$ ($|\mu| = 5$) described by the following physical variables: gravity, cart mass, pole mass, pole length, magnetic force. Their respective pre-normalized domains are $[4.8, 14.8]$, $[0.5, 1.5]$, $[0.01, 0.19]$, $[0.2, 0.8]$, and $[-10, 10]$. The value of μ are uniformly sampled. Knowing some components of μ might not be required to behave optimally. The discrete action space is $\{-1, 1\}$.

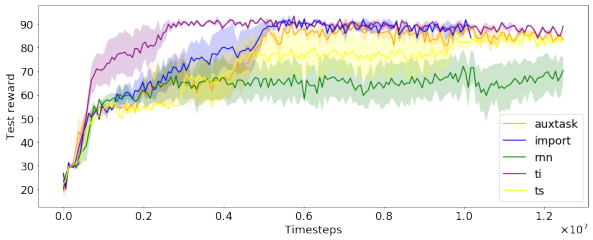
Episode length is $T = 100$.

Final performance and sample efficiency. Table 3.1 shows IMPORT's performance is marginally superior to other methods in most settings. Learning curves in Figure A.1 allow analyzing the sample efficiency of the different methods. Overall, IMPORT is more sample efficient than other methods in the privileged information μ setting. Moreover, the use of the auxiliary loss ($\beta > 0$) usually speed-up the learning conver-

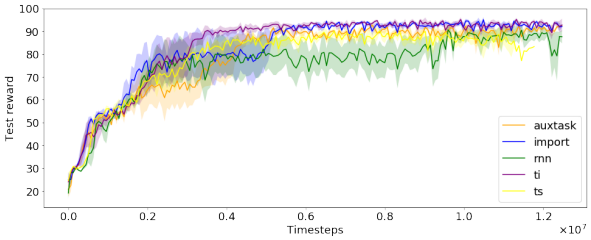
gence by enforcing the RNN to quickly produce a coherent embedding. We can see that only sharing parameters ($\beta = 0$) already helps improving over RNNs.



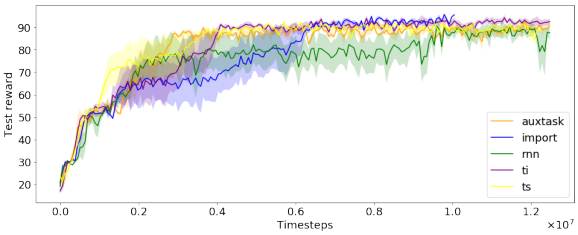
(**a**) CartPole with μ and $N = 10$



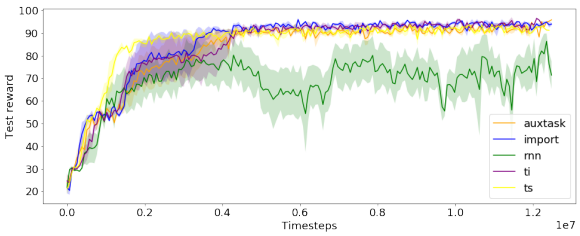
(**b**) CartPole with with TID and $N = 10$



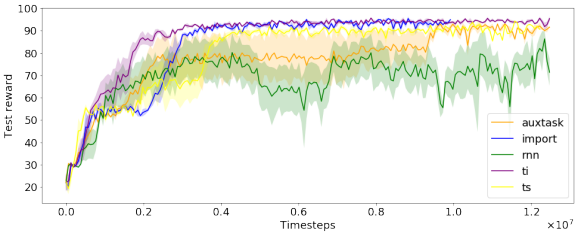
(**c**) CartPole with μ and $N = 20$



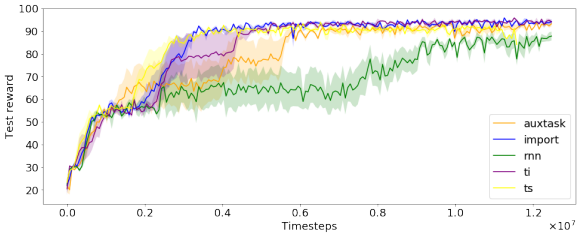
(**d**) CartPole with TID and $N = 20$



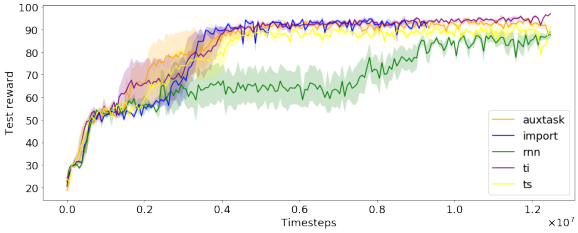
(**e**) CartPole with μ and $N = 50$



(**f**) CartPole with TID and $N = 50$



(**g**) CartPole with μ and $N = 100$



(**h**) CartPole with TID and $N = 100$

Figure A.1 – Evaluation on CartPole where the agent has access to μ or task descriptors (TID stands for task identifier)

Non-stationary environments. We consider the non-stationary version of CarPole environment where at each timestep, there is a probability $\rho = 0.05$ to sample a new dynamic μ. Table A.2 shows that the performance of IMPORT, AuxTask and `TI` are comparable in these settings.

Method	$N = 10$	$N = 100$
AuxTask	86.4(1.0)	93.0(0.3)
IMPORT	**91.7(0.5)**	92.7(0.8)
RNN	65.5(4.3)	89.5(0.6)
TI	88.2(3.9)	**95.5(0.8)**
TS	86.7(1.6)	92.2(0.7)

Figure A.2 – CartPole (non-stationary).

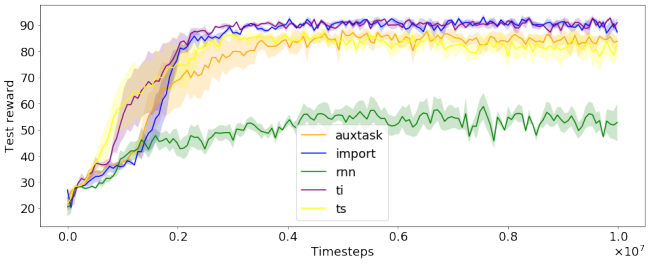


Figure A.3 – Non-stationary CartPole with $N = 10$

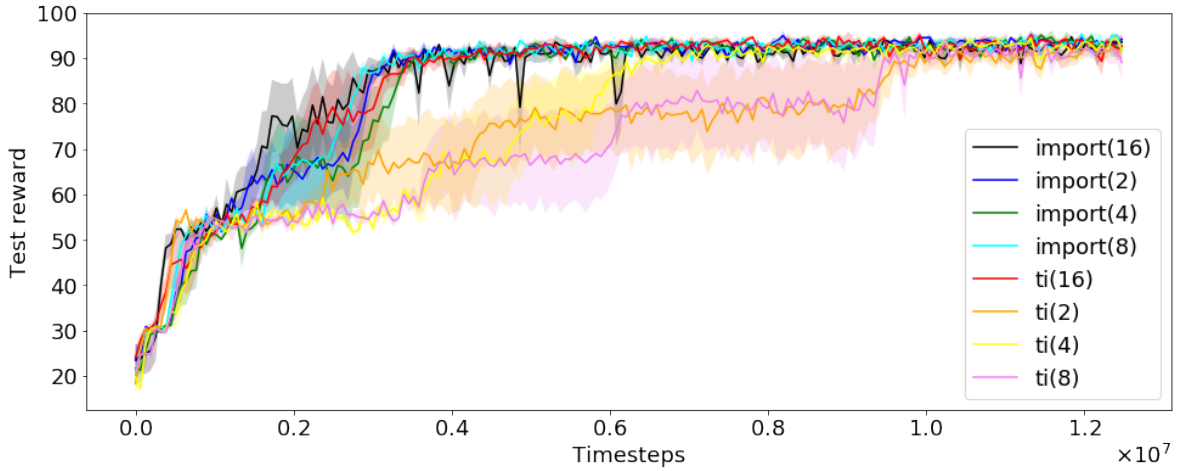
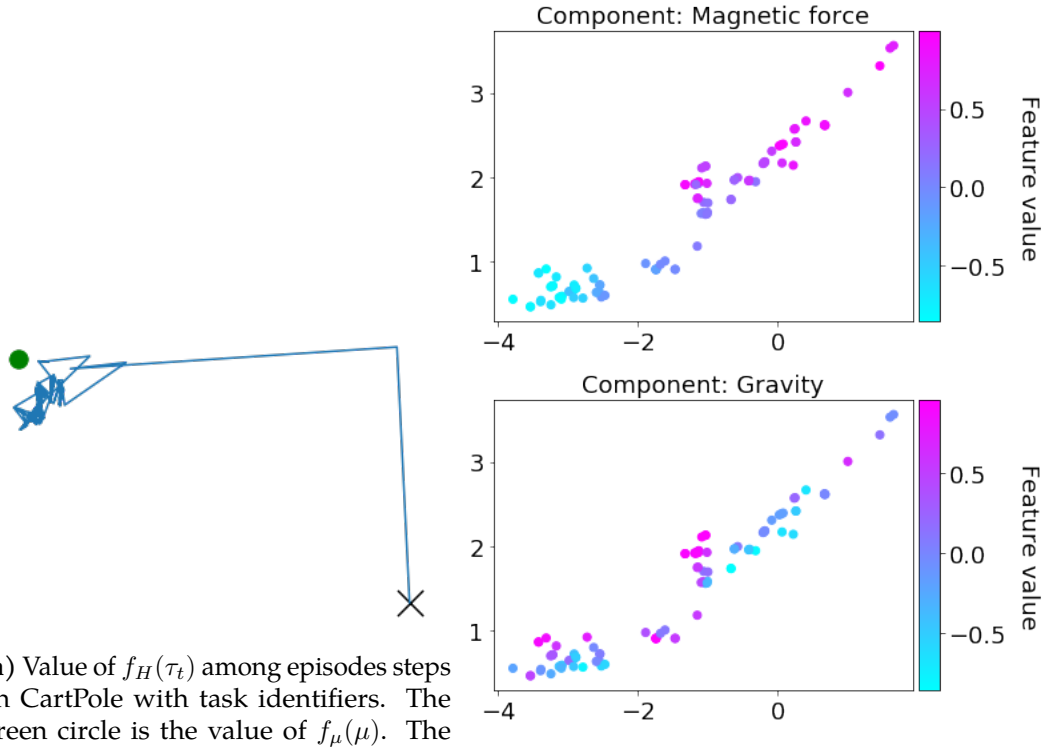


Figure A.4 – IMPORT and TI with different task embedding representation size on CartPole with $N = 20$

Size of built embeddings. We now study the impact of the task embedding representation size. As can be seen from Figure A.4, IMPORT's performance remains stable for different representation sizes in $\{2, 4, 8, 16\}$ whereas TI's sample efficiency decreases with this dimension.

Trajectory and task embeddings. In Figure A.5, we plot both the evolution of $f_H(\tau_t)$ during an episode of the final model obtained training IMPORT with two-dimensional task embeddings on CartPole with **task identifiers** (left) and task embedding $f_\mu(\mu)$ learnt by the informed policy (right). As expected, the history embedding gets close to the task embedding after just a few timesteps (left). Interestingly, task embeddings $f_\mu(\mu)$ are able to capture relevant information from the task. For instance, they are highly correlated with the *magnetic force* which is a very strong factor to "understand" from each new environment to control the system correctly. At the opposite, *gravity* is



(**a**) Value of $f_H(\tau_t)$ among episodes steps on CartPole with task identifiers. The green circle is the value of $f_\mu(\mu)$. The image shows that IMPORT starts with a random embedding, and is able to discover the task embedding with a reasonable performance in a few steps.

(**b**) Task embeddings $f_\mu(\mu)$ for Cartpole with task identifiers. The color of the point corresponds to the value of one of the 'real' physics component of the environment (unknown to the model).

Figure A.5 – Visualization of task embeddings upon Cartpole

less correlated since it does not influence the optimal policy – whatever the gravity is, if the pole is on the left, then you have to go right and vice-versa.

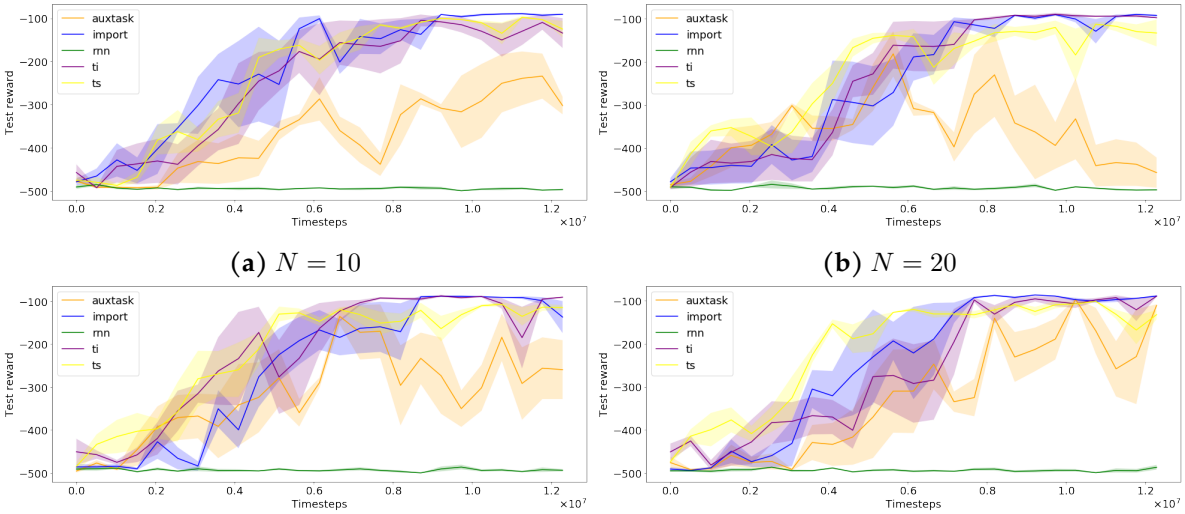
A.3.2 Acrobot

Acrobot consists of two joints and two links, where the joint between the two links is actuated. Initially, the links are hanging downwards, and the goal is to swing the end of the lower link up to a given height. Environment dynamics are determined by the length of the two links, their masses, their maximum velocity. Their respective pre-normalized domains are $[0.5, 1.5]$, $[0.5, 1.5]$, $[0.5, 1.5]$, $[0.5, 1.5]$, $[3\pi, 5\pi]$ and $[7\pi, 11\pi]$. Unlike CartPole, the environment is stochastic because the simulator applies noise to the applied force. The action space is $\{-1, 0, 1\}$. We also add an extra dynamics parameter which controls whether the action order is inverted, i.e. $\{1, 0, -1\}$, thus $|\mu| = 7$.

Episode length is 500.

	$N = 10$	$N = 20$	$N = 50$	$N = 100$
AuxTask	$-189.0(54.8)$	$-98.3(1.8)$	$-103.0(8.0)$	$-93.6(1.3)$
IMPORT	$\mathbf{-87.2(0.9)}$	$\mathbf{-92.5(1.3)}$	$-88.9(1.1)$	$-88.9(1.6)$
RNN	$-483.6(1.6)$	$-482.7(4.0)$	$-480.7(3.5)$	$-485.0(3.7)$
TaskInference	$-89.7(1.2)$	$-94.6(0.7)$	$\mathbf{-87.8(0.8)}$	$\mathbf{-87.3(1.2)}$
TS	$-101.4(2.0)$	$-102.1(6.0)$	$-102.4(2.0)$	$-102.3(0.8)$

Table A.2 – Acrobot



(a) $N = 10$ **(b)** $N = 20$

(c) $N = 50$ **(d)** $N = 100$

Figure A.6 – Performance on Acrobot

IMPORT outperforms all baselines in settings with small training task sets (Figure A.6 and Table A.2) and perform similarly to `TI` on larger training task sets.

A.3.3 Bandits

The **Bandit** environment is a standard Bernoulli multi-armed bandit problem with K arms. The vector $\mu \in \mathbb{R}^K$ denotes the probability of success of the independent Bernoulli distributions. Each dimension of μ is sampled uniformly between 0 and 0.5, the best arm is randomly selected and associated to a probability of 0.9. Although relatively simple, this environment assesses the ability of algorithms to learn nontrivial exploration/exploitation strategies.

Note that it is not surprising that UCB outperforms the other algorithms in this setting. UCB is an optimal algorithm for MAB and we have optimized it for achieving

the best empirical performance. Moreover, IMPORT cannot leverage correlations between tasks since, due to the generation process, tasks are independent.

We visualize the task embeddings learnt by the informed policy in A.7.

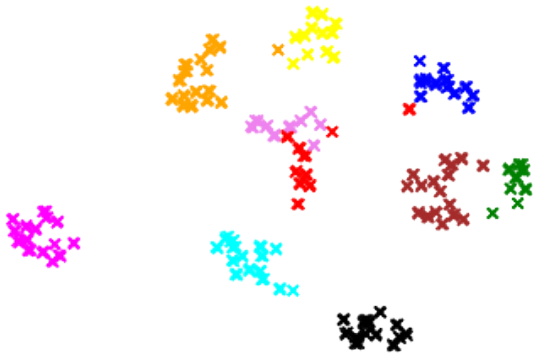
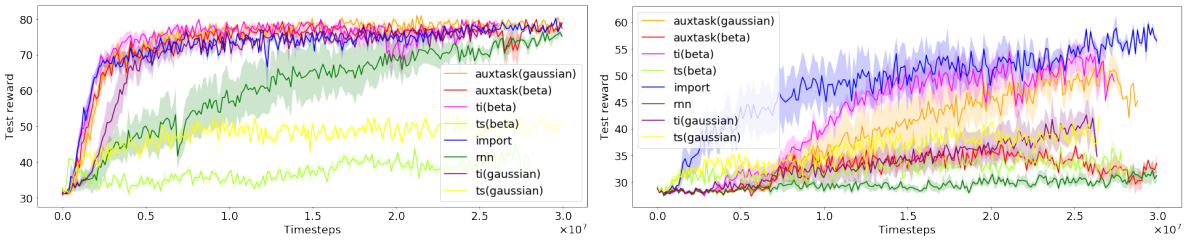


Figure A.7 – t-SNE of the task embeddings on the bandit problem with $K = 10$.



(**a**) Bandits with $K = 10$ arms

(**b**) Bandits with $K = 20$ arms

Figure A.8 – Learning curves on the bandit problem.

A.3.4 Maze3d environment

The **Maze 3D** environment (Figure 3.3) is a continuous maze problem implemented using gym-miniworld [Che18], with 3 discrete actions (forward, left, right) where the objective is to reach one of the two possible goals, resulting in a reward of $+1$ (resp. -1) when the correct (resp. wrong) goal is reached. If a box is touched, the episode ends. The maze's axis range from -40 to 40, the two turn actions (*left*, *right*) modify the angle by 45 degrees, and the *forward action* is a 5 length move. The agent starts in a random position with a random orientation. The information about which goal to reach at each episode is encoded by the use of two different textures on the wall located on the opposite side of the boxes. In this way, the agent cannot simultaneously observe both boxes and the "informative" wall.

This environment allows to evaluate the models in a setting where the observation is a high dimensional space (3x60x60 RGB image). The mapping between the RGB image and the task target in $\{-1, 1\}$ is challenging and the informed policy should provide better auxiliary task targets than `TI` thanks to the "easy" training of the informed policy.

IMPORT outperformsTI on this environment (Figure A.9) in both final performance and sample efficiency.

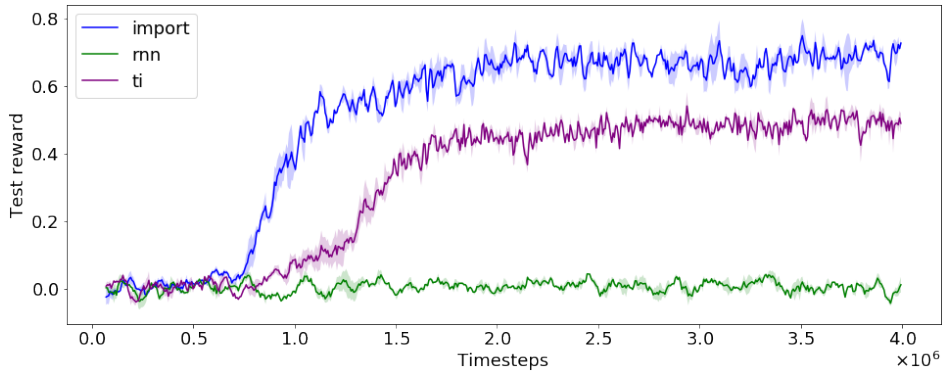


Figure A.9 – Learning curves on the Maze 3D environment

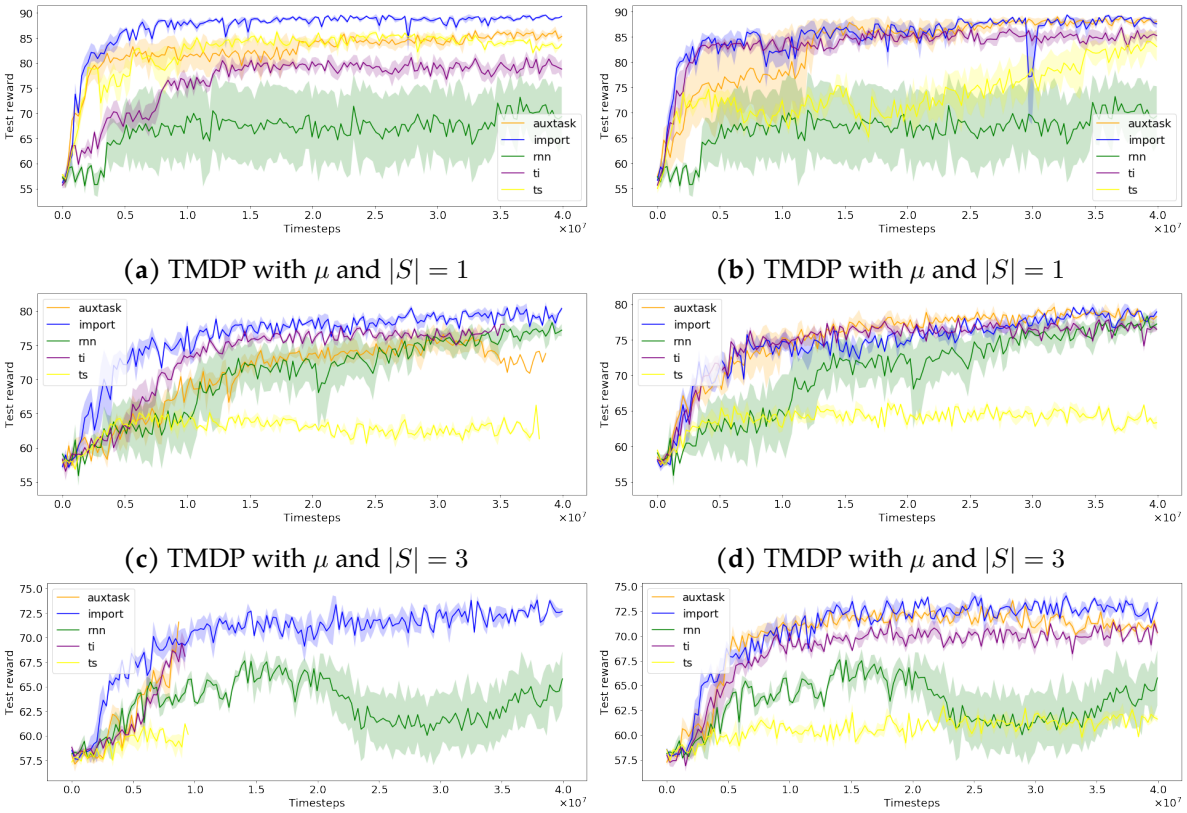
A.3.5 Tabular MDPs

Tabular MDP [Dua+16] is a MDP with S discrete states and A actions such that the transition matrix is sampled from a flat Dirichlet distribution, and the reward function is sampled from a uniform distribution in $[0, 1]$. The task identifier μ is a concatenation of the transition and reward functions resulting in a vector of size $S^2A + SA$, allowing to test the models with high-dimensional μ.

IMPORT outperforms all baselines in all settings (Figure A.10 and Table 3.2).

A.4 Impact of the β hyperparameter

We study the sensibility of the β parameter on IMPORT. Figure A.11 clearly shows the benefits of using the auxiliary objective. On all but the Tabular-MDP environments, the recurrent policy successfully leverages the auxiliary objective to improve both sample efficiency and final performance for Acrobot.



(**a**) TMDP with μ and $|S| = 1$

(**b**) TMDP with μ and $|S| = 1$

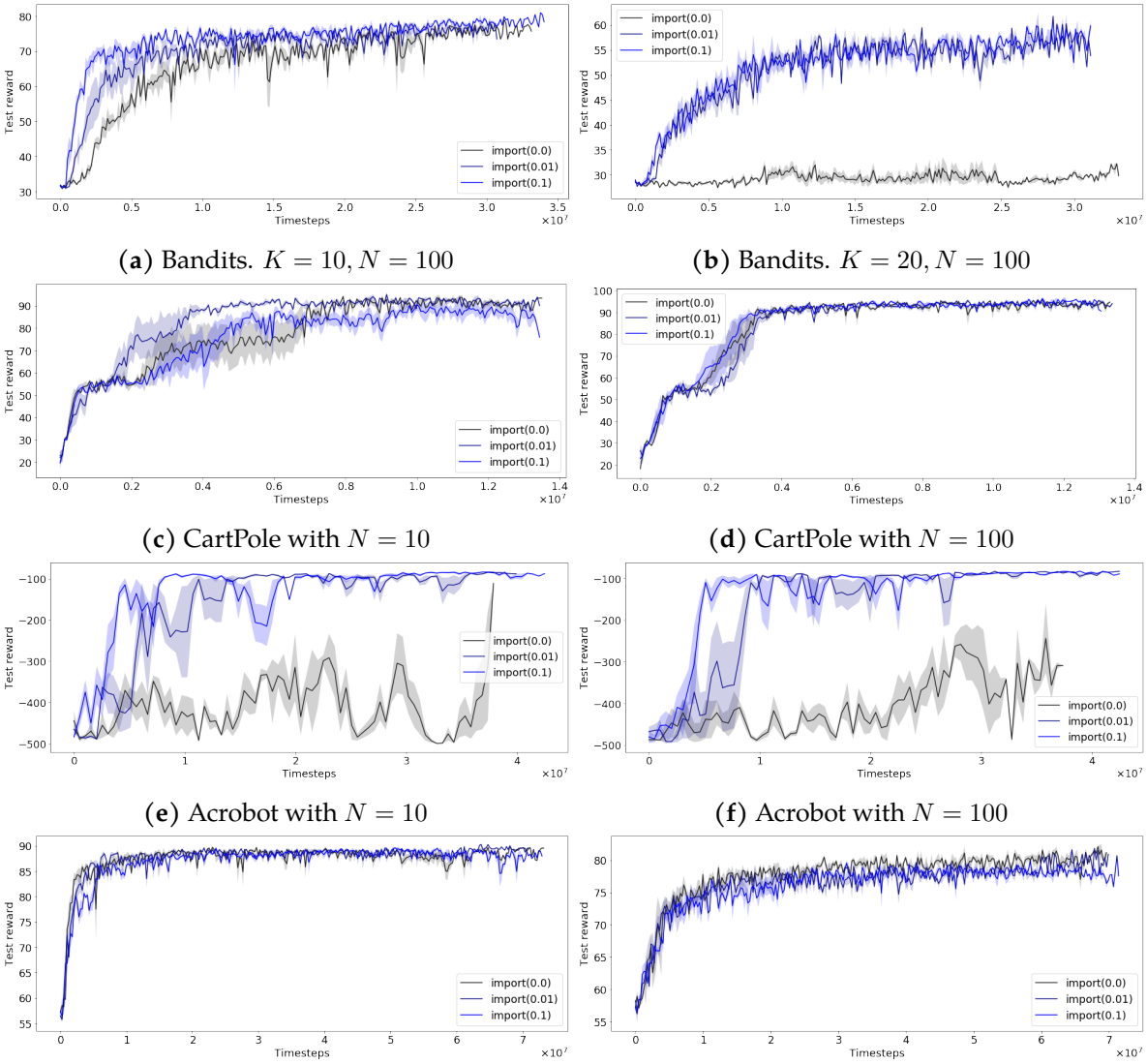
(**c**) TMDP with μ and $|S| = 3$

(**d**) TMDP with μ and $|S| = 3$

(**e**) TMDP with μ and $|S| = 5$

(**f**) TMDP with μ and $|S| = 5$

Figure A.10 – Evaluation on Tabular-MDP with different parameters and task descriptors (TID stands for task identifier).



(**a**) Bandits. $K = 10, N = 100$

(**b**) Bandits. $K = 20, N = 100$

(**c**) CartPole with $N = 10$

(**d**) CartPole with $N = 100$

(**e**) Acrobot with $N = 10$

(**f**) Acrobot with $N = 100$

(**g**) TMDP with $|S| = 1, |A| = 5, N = 100$

(**h**) TMDP with $|S| = 3, |A| = 5, N = 100$

Figure A.11 – Test performance of IMPORT for different β parameters (auxiliary supervised objective). We only report performance on informative μ task descriptors.

Appendix B

Complements on Chapter 4

B.1 Theoretical Details on Section 4.4

B.1.1 Proofs of Lemmas 1 and 2

Restatement of Lemma 1. There exists a value $\eta^\dagger \in (0, 1)$ such that solving $(\mathcal{P}_{\eta^\dagger})$ is equivalent to maximizing a lower bound on the mutual information objective $\max_{N_Z,\rho,\pi,\phi} \mathcal{I}(S_{\text{diff}}; Z)$.

Proof. We assume that the number of available skills is upper bounded, i.e., $1 \leq N_Z \leq N_{\max}$. We begin by lower bounding the negative conditional entropy by using the well known lower bound of Barber and Agakov [BA04] on the mutual information

$$\begin{aligned} -\mathcal{H}(Z|S_{\text{diff}}) &= \sum_{z\in Z} \rho(z)\mathbb{E}_{s_{\text{diff}}}\left[\log p(z|s_{\text{diff}})\right] \\ &\geq \sum_{z\in Z} \rho(z)\mathbb{E}_{s_{\text{diff}}}\left[\log q_\phi(z|s_{\text{diff}})\right]. \end{aligned}$$

We now use that any weighted average is lower bounded by its minimum component, which yields

$$-\mathcal{H}(Z|S_{\text{diff}}) \geq \min_{z\in Z} \mathbb{E}_{s_{\text{diff}}}\left[\log q_\phi(z|s_{\text{diff}})\right].$$

Thus the following objective is a lower bound on the original objective of maximizing $\mathcal{I}(S_{\text{diff}}; Z)$

$$\max_{N_Z=N,\rho,\pi,\phi} \left\{ \mathcal{H}(Z) + \min_{z\in[N]} \mathbb{E}_{s_{\text{diff}}} \left[\log q_\phi(z|s_{\text{diff}})\right] \right\}. \tag{B.1}$$

Interestingly, the second term in eq. (B.1) no longer depends on the skill distribution ρ, while the first entropy term $\mathcal{H}(Z)$ is maximized by setting ρ to the uniform distribution over N skills (i.e., $\max_\rho \mathcal{H}(Z) = \log(N)$). This enables to simplify the optimization which now only depends on N. Thus eq. (B.1) is equivalent to

$$\max_{N_Z=N} \left\{ \log(N) + \max_{\pi,\phi} \min_{z\in[N]} \mathbb{E}_{s_{\text{diff}}} \left[\log q_\phi(z|s_{\text{diff}})\right] \right\}. \tag{B.2}$$

We define the functions

$$f(N) := \log(N), \qquad g(N) := \max_{\pi,\phi} \min_{z\in[N]} \mathbb{E}_{s_{\text{diff}}} \left[\log q_\phi(z|s_{\text{diff}})\right].$$

Let $N^\dagger \in \arg\max_N f(N) + g(N)$ and $\eta^\dagger := \exp g(N^\dagger) \in (0, 1)$. We now show that any solution of $(\mathcal{P}_{\eta^\dagger})$ is a solution of eq. (B.2). Indeed, denote by $N^\star$ the value that optimizes $(\mathcal{P}_{\eta^\dagger})$. First, by validity of the constraint, it holds that $g(N^\star) \geq \log \eta^\dagger = g(N^\dagger)$. Second, since $N^\dagger$ meets the constraint and $N^\star$ is the maximal number of skills that satisfies the constraint of $(\mathcal{P}_{\eta^\dagger})$, by optimality we have that $N^\star \geq N^\dagger$ and therefore $f(N^\star) \geq f(N^\dagger)$ since f is non-decreasing. We thus have

$$\begin{cases} g(N^\star) \geq g(N^\dagger) \\ f(N^\star) \geq f(N^\dagger) \end{cases} \implies f(N^\star) + g(N^\star) \geq f(N^\dagger) + g(N^\dagger).$$

Putting everything together, an optimal solution for $(\mathcal{P}_{\eta^\dagger})$ is an optimal solution for eq. (B.2), which is equivalent to eq. (B.1), which is a lower bound of the MI objective, thus concluding the proof. □

Restatement of Lemma 2. The function g is non-increasing in N.

Proof. We have that $g(N) := \max_{\pi,q} \min_{z\in[N]} \mathbb{E}_{s\sim\pi(z)}[\log(q(z|s)]$, where throughout the proof we write s instead of s_{diff} for ease of notation. Here the optimization variables are $\pi \in (\Pi)^N$ (i.e., a set of N policies) and $q : \mathcal{S} \to \Delta(N)$, i.e., a classifier of states to N possible classes, where $\Delta(N)$ denotes the N-simplex. For $z \in [N]$, let

$$h_N(\pi, q, z) := \mathbb{E}_{s\sim\pi(z)}[\log(q(z|s)], \qquad f_N(\pi, q) := \min_{z\in[N]} h_N(\pi, q, z).$$

Let $(\pi', q') \in \arg\max_{\pi,q} f_{N+1}(\pi, q)$. We define $\widetilde{\pi} := (\pi'(1), \dots, \pi'(N)) \in (\Pi)^N$ and $\widetilde{q} : \mathcal{S} \to \Delta(N)$ such that $\widetilde{q}(i|s) := q'(i|s)$ for any $i \in [N-1]$ and $\widetilde{q}(N|s) := q'(N|s) + q'(N+1|s)$. Intuitively, we are "discarding" policy $N+1$ and "merging" class $N+1$ with class N.

Then it holds that

$$g(N+1) = f_{N+1}(\pi', q') = \min_{z \in [N+1]} h_{N+1}(\pi', q', z) \leq \min_{z \in [N]} h_{N+1}(\pi', q', z).$$

Now, by construction of $\widetilde{\pi}, \widetilde{q}$, we have for any $i \in [N-1]$ that $h_{N+1}(\pi', q', i) = h_N(\widetilde{\pi}, \widetilde{q}, i)$. As for class N, since $\widetilde{\pi}(N) = \pi'(N)$, by definition of $\widetilde{q}(N|\cdot)$ and from monotonicity of the log function, it holds that $h_{N+1}(\pi', q', N) = \mathbb{E}_{s \sim \pi'(N)}[\log(q'(N|s)]$ satisfies

$$h_{N+1}(\pi', q', N) \leq \mathbb{E}_{s \sim \widetilde{\pi}(N)}[\log(\widetilde{q}(N|s)] = h_N(\widetilde{\pi}, \widetilde{q}, N).$$

Hence, we get that

$$\min_{z \in [N]} h_{N+1}(\pi', q', z) \leq \min_{z \in [N]} h_N(\widetilde{\pi}, \widetilde{q}, z) = f_N(\widetilde{\pi}, \widetilde{q}) \leq g(N).$$

Putting everything together gives $g(N+1) \leq g(N)$, which yields the desired result. □

B.1.2 Simple Illustration of the Issue with Uniform-ρ MI Maximization

This section complements Sect. 4.4.2: we show a simple scenario where **1)** considering both a uniform ρ prior and a fixed skill number N_Z provably leads to suboptimal MI maximization, and where **2)** the `UPSIDE` strategy of considering a uniform ρ restricted to the η-discriminable skills can provably increase the MI for small enough η.

Consider the simple scenario (illustrated on Fig. B.1) where the agent has N skills indexed by n and must assign them to M states indexed by m. We assume that the execution of each skill deterministically brings it to the assigned state, yet the agent may assign stochastically (i.e., more than one state per skill). (A non-RL way to interpret this is that we want to allocate N balls into M boxes.) Denote by $p_{n,m} \in [0, 1]$ the probability that skill n is assigned to state m. It must hold that $\forall n \in [N], \sum_m p_{n,m} = 1$. Denote by $\mathcal{I}$ the MI between the skill variable and the assigned state variable, and by $\overline{\mathcal{I}}$ the MI under the prior that the skill sampling distribution ρ is uniform, i.e.,

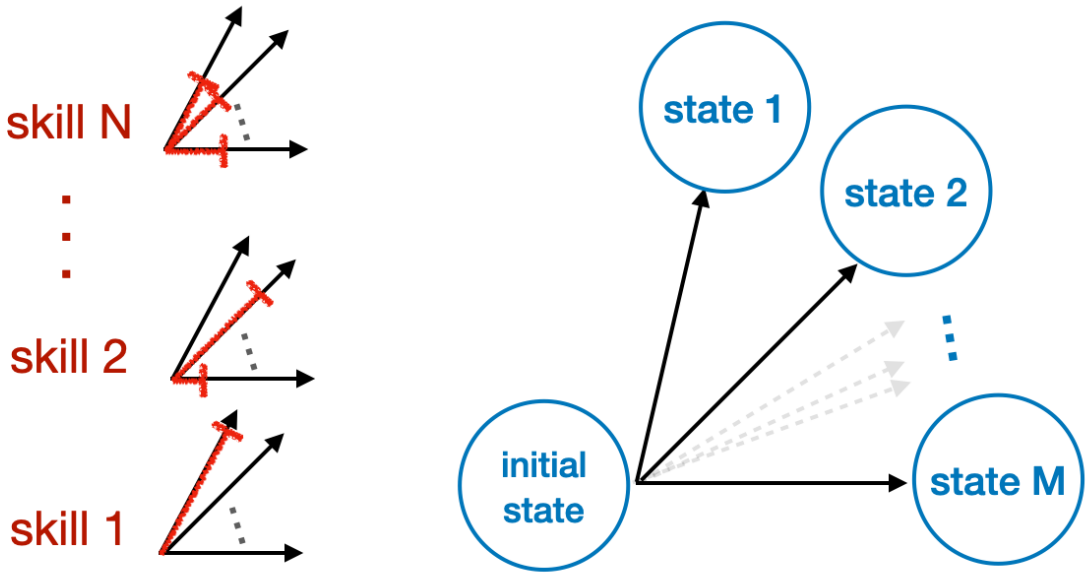


Figure B.1 – The agent must assign (possibly stochastically) N skills to M states: *under the prior of uniform skill distribution, can the MI be increased by varying the number of skills N?*

$\rho(n) = 1/N$. It holds that

$$\overline{\mathcal{I}}(N, M) = -\sum_n \frac{1}{N} \log \frac{1}{N} + \sum_{n,m} \frac{1}{N} p_{n,m} \log \frac{\frac{1}{N} p_{n,m}}{\sum_n \frac{1}{N} p_{n,m}} = \log N + \frac{1}{N} \sum_{n,m} p_{n,m} \log \frac{p_{n,m}}{\sum_n p_{n,m}}.$$

Let $\overline{\mathcal{I}}^\star(N, M) := \max_{\{p_{n,m}\}} \overline{\mathcal{I}}(N, M)$ and $\{p^\star_{n,m}\} \in \arg\max_{\{p_{n,m}\}} \overline{\mathcal{I}}(N, M)$. We also define the *discriminability* of skill n in state m as

$$q_{n,m} := \frac{p_{n,m}}{\sum_n p_{n,m}},$$

as well as the *minimum discriminability* of the optimal assignment as

$$\eta := \min_n \max_m q^\star_{n,m}.$$

Lemma 3. *There exist values of N and M such that the uniform-ρ MI can be improved by removing a skill (i.e., by decreasing N).*

Proof. The following example shows that with $M = 2$ states, it is beneficial for the uniform-ρ MI maximization to go from $N = 3$ to $N = 2$ skills. Indeed, we can

numerically compute the optimal solutions and we obtain for $N = 3$ and $M = 2$ that

$$\overline{\mathcal{I}}^\star(N=3,M=2) \approx 0.918, \qquad p^\star_{n,m} = \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad q^\star_{n,m} = \begin{pmatrix} 0 & 0.5 \\ 0 & 0.5 \\ 1 & 0 \end{pmatrix}, \qquad \eta = 0.5,$$

whereas for $N = 2$ and $M = 2$,

$$\overline{\mathcal{I}}^\star(N=2,M=2) = 1, \qquad p^\star_{n,m} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad q^\star_{n,m} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad \eta = 1.$$

As a result, $\overline{\mathcal{I}}^\star(N=2,M=2) > \overline{\mathcal{I}}^\star(N=3,M=2)$, which concludes the proof. The intuition why $\overline{\mathcal{I}}^\star$ is increased by decreasing N is that for $N = 2$ there is one skill per state whereas for $N = 3$ the skills must necessarily overlap. Note that this contrasts with the original MI (that also optimizes ρ) where decreasing N cannot improve the optimal MI. □

The previous simple example hints to the fact that the value of the minimum discriminability of the optimal assignment η may be a good indicator to determine whether to remove a skill. The following more general lemma indeed shows that a sufficient condition for the uniform-ρ MI to be increased by removing a skill is that η is small enough.

Lemma 4. *Assume without loss of generality that the skill indexed by N has the minimum discriminability η, i.e., $N \in \arg\min_n \max_m q^\star_{n,m}$. Define*

$$\Delta(N,\eta) := \log N - \frac{N-1}{N}\log(N-1) + \frac{1}{N}\log\eta.$$

If $\Delta(N,\eta) \leq 0$ — which holds for small enough η — then removing skill N results in a larger uniform-ρ optimal MI, i.e., $\overline{\mathcal{I}}^\star(N,M) < \overline{\mathcal{I}}^\star(N-1,M)$.

Proof. It holds that

$$\begin{aligned}
\overline{\mathcal{I}}^\star(N,M) &= \log N + \frac{1}{N}\left(\sum_{n\in[N-1]}\sum_{m\in[M]} p^\star_{n,m}\log q^\star_{n,m} + \sum_{m\in[M]} p^\star_{n,m}\log\eta\right) \\
&= \log N - \frac{N-1}{N}\log(N-1) \\
&\quad + \frac{N-1}{N}\left(\log(N-1) + \frac{1}{N-1}\sum_{n\in[N-1]}\sum_{m\in[M]} p^\star_{n,m}\log q^\star_{n,m}\right) + \frac{1}{N}\log\eta
\end{aligned}$$

$$= \Delta(N, \eta) + \frac{N-1}{N}\overline{\mathcal{I}}^{\star}(N-1, M).$$

As a result, if $\Delta(N, \eta) \leq 0$ then $\overline{\mathcal{I}}^{\star}(N, M) < \overline{\mathcal{I}}^{\star}(N-1, M)$. □

B.2 UPSIDE Algorithm

B.2.1 Visual illustrations of UPSIDE's Design Mentioned in Section 4.4

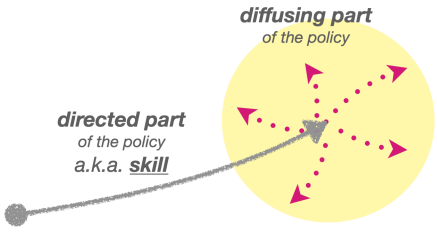


Figure B.2 – Decoupled structure of an UPSIDE policy: a directed skill followed by a diffusing part.

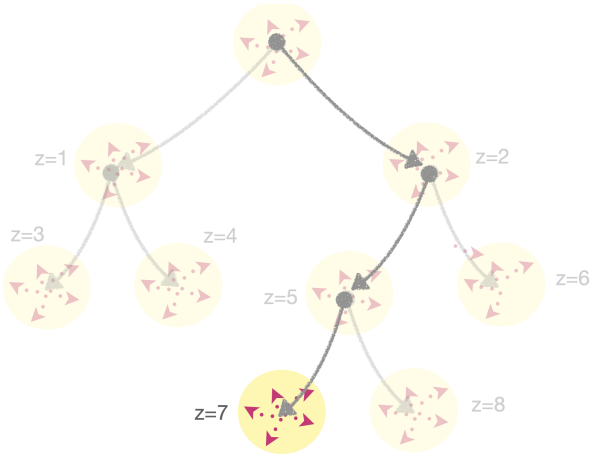


Figure B.3 – In the above UPSIDE tree example, executing policy $z = 7$ means sequentially composing the skills of policies $z \in \{2, 5, 7\}$ and then deploying the diffusing part of policy $z = 7$.

B.2.2 High-Level Approach of UPSIDE

B.2.3 Details of Algorithm algorithm B.1

We give in algorithm B.1 a more detailed version of ,algorithm 4.1 and we list some additional explanations below.

- When optimizing the discriminator, rather than sampling (state, policy) pairs with equal probability for all nodes from the tree $\mathcal{T}$, we put more weight (e.g. $3\times$) on already consolidated policies, which seeks to avoid the new policies from invading the territory of the older policies that were previously correctly learned.

Algorithm B.1: Detailed UPSIDE

Parameters: Discriminability threshold $\eta \in (0, 1)$, branching factor N^{start}, $N^{\max}$
Initialize: Tree $\mathcal{T}$ initialized as a root node index by 0, policy candidates $\mathcal{Q} = \{0\}$, state buffers $\mathcal{B}_Z = \{0 : [\,]\}$
while $\mathcal{Q} \neq \emptyset$ **do** `// tree expansion`
  Dequeue a policy $z \in \mathcal{Q}$ and create $N = N^{\text{start}}$ nodes $\mathcal{C}(z)$ rooted at z and add new key z to B_Z
  Instantiate new replay buffer $\mathcal{B}_{\text{RL}}$
  PolicyLearning($\mathcal{B}_{\text{RL}}, \mathcal{B}_Z, \mathcal{T}, \mathcal{C}(z)$)
  if $\min_{z' \in \mathcal{C}(z)} d(z') > \eta$ **then** `// Node addition`
    while $\min_{z' \in \mathcal{C}(z)} d(z') > \eta$ **and** $N < N^{max}$ **do**
      Increment $N = N + 1$ and add one policy to $\mathcal{C}(z)$
      PolicyLearning($\mathcal{B}_{\text{RL}}, \mathcal{B}_Z, \mathcal{T}, \mathcal{C}(z)$)
    end
  end
  else `// Node removal`
    while $\min_{z' \in \mathcal{C}(z)} d(z') < \eta$ **and** $N > 1$ **do**
      Reduce $N = N - 1$ and remove least discriminable policy from $\mathcal{C}(z)$
      PolicyLearning($\mathcal{B}_{\text{RL}}, \mathcal{B}_Z, \mathcal{T}, \mathcal{C}(z)$)
    end
  end
  Enqueue in $\mathcal{Q}$ the η-discriminable nodes $\mathcal{C}(z)$
end
PolicyLearning(Replay buffer $\mathcal{B}_{\text{RL}}$, State buffers $\mathcal{B}_Z$, Tree $\mathcal{T}$, policies to update Z_U)
Optimization parameters: patience K, policy-to-discriminator update ratio J, K_{discr} discriminator update epochs, K_{pol} policy update epochs
Initialize: Discriminator q_ϕ with $|\mathcal{T}|$ classes
for K iterations **do** `// Training loop`
  For all $z' \in Z_U$, clear $\mathcal{B}_Z[z']$ then collect and add B states from the diffusing part of $\pi(z')$ to it
  Train the discriminator q_ϕ for K_{discr} steps with dataset $\bigcup_{z' \in \mathcal{T}} \mathcal{B}_Z[z']$.
  Compute discriminability $d(z') = \widehat{q}^{\mathcal{B}}_\phi(z') = \frac{1}{|\mathcal{B}_{z'}|} \sum_{s \in \mathcal{B}_{z'}} q_\phi(z'|s)$ for all $z' \in Z_U$
  if $\min_{z' \in Z_U} d(z') > \eta$ **then** `// Early stopping`
    Break
  end
  for J iterations **do**
    For all $z' \in Z_U$, sample a trajectory from $\pi(z')$ and add to replay buffer $\mathcal{B}_{\text{RL}}$
    For all $z' \in Z_U$, update policy π'_z for K_{pol} steps on replay buffer $\mathcal{B}_{\text{RL}}$ to optimize the discriminator reward as in Sect. 4.4.1 keeping skills from parent policies fixed
  end
end
Compute discriminability $d(z')$ for all $z' \in Z_U$

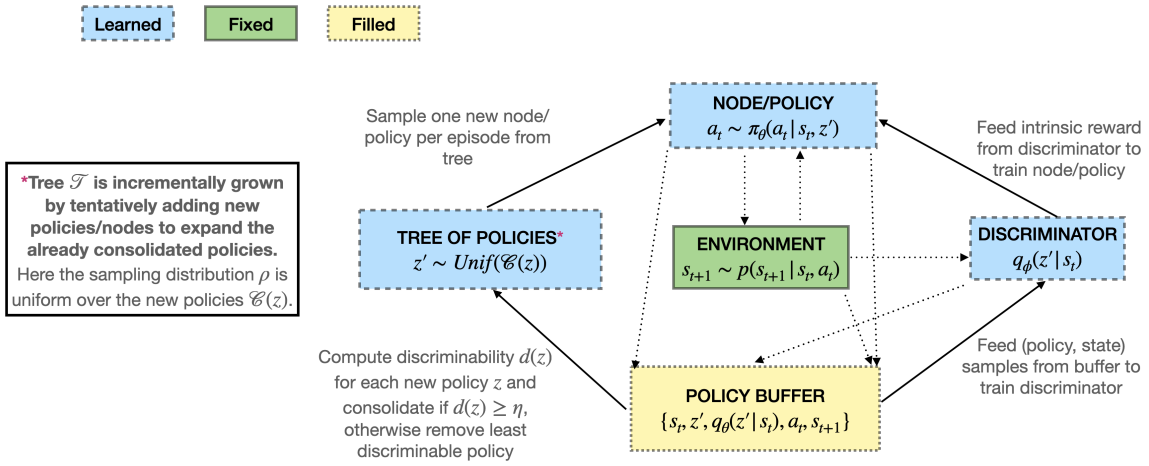


Figure B.4 – High-level approach of `UPSIDE`.

- A replay buffer $\mathcal{B}_{\text{RL}}$ is instantiated at every new expansion (line 5), thus avoiding the need to start collecting data from scratch with the new policies at every PolicyLearning call.
- J (line 31) corresponds to the number of policy updates ratio w.r.t. discriminator updates, i.e. for how long the discriminator reward is kept fixed, in order to add stationarity to the reward signal.
- Instead of using a number of iterations K to stop the training loop of the PolicyLearning function (line 24), we use a maximum number of environment interactions K_{steps} for node expansion. Note that this is the same for `DIAYN-hier` and `DIAYN-curr`.
- The state buffer size B needs to be sufficiently large compared to H so that the state buffers of each policy represent well the distribution of the states generated by the policy's diffusing part. In practice we set $B = 10H$.
- In PolicyLearning, we add K_{initial} random (uniform) transitions to the replay buffer for each newly instantiated policies.
- Moreover, in PolicyLearning, instead of sampling uniformly the new policies we sample them in a round robin fashion (i.e., one after the other), which can be simply seen as a variance-reduced version of uniform sampling.

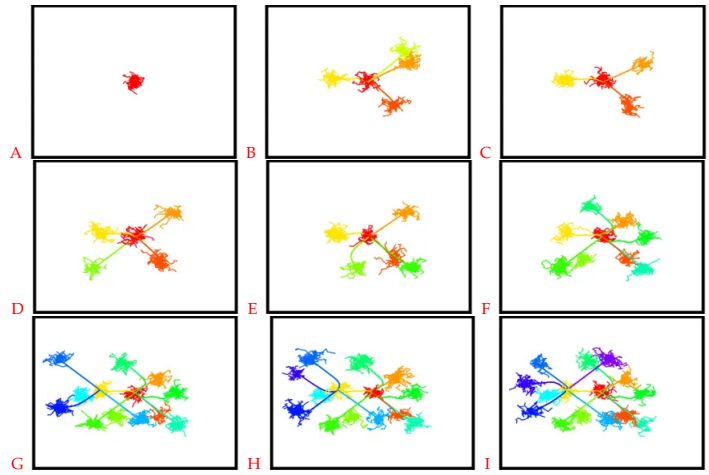


Figure B.5 – Fine-grained evolution of the tree structure on a wall-free maze with $N^{\text{start}} = 4$ and $N^{\text{max}} = 8$. The environment is a wall-free continuous maze with initial state s_0 located at the center of the maze. Image A represents the diffusing part around s_0. In image B, $N^{\text{start}} = 4$ policies are trained, yet one of them (in lime yellow) is not sufficiently discriminable, thus it is pruned, resulting in image C. A small number of interactions is enough to ensure that the three policies are η-discriminable (image C). In image D, a fourth policy (in green) is able to become η-discriminable. New policies are added, trained and η-discriminated from 5 policies (image E) to $N^{\text{max}} = 8$ policies (image F). Then a policy (in yellow) is expanded with $N^{\text{start}} = 4$ policies (image G). They are all η-discriminable so additional policies are added (images H, I, ...). The process continues until convergence or until time-out (as done here). On the left, we plot the number of active policies (which represents the number of policies that are being trained at the current level of the tree) as well as the average discriminator accuracy over the active policies.

B.2.4 Illustration of the Evolution of UPSIDE's Tree on a Wall-Free Maze

See fig. B.5.

B.2.5 Illustration of Evolution of UPSIDE's Tree on the Bottleneck Maze

See Fig. B.6.

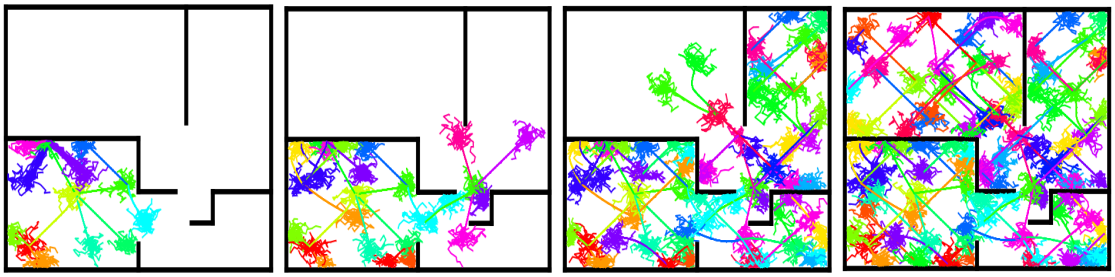


Figure B.6 – Incremental expansion of the tree learned by UPSIDE towards unexplored regions of the state space in the Bottleneck Maze.

B.3 Experimental Details

B.3.1 Baselines

DIAYN-N_Z. This corresponds to the original DIAYN algorithm [Eys+19] where N_Z is the number of skills to be learned. In order to make the architecture more similar to UPSIDE, we use distinct policies for each skill, i.e. they do not share weights as opposed to [Eys+19]. While this may come at the price of sample efficiency, it may also help put lesser constraint on the model (e.g. gradient interference).

DIAYN-curr. We augment DIAYN with a curriculum that enables to be less dependent on an adequate tuning of the number of skills of DIAYN. We consider the curriculum of UPSIDE where we start learning with N^{start} policies during a period of time/number of interactions. If the configuration satisfies the discriminablity threshold η, a skill is added, otherwise a skill is removed or learning stopped (as in Alg. B.1, lines 5-12). Note that the increasing version of this curriculum is similar to the one proposed in VALOR [Ach+18]. In our experiments, we use $N^{\text{start}} = 1$.

DIAYN-hier. We extend DIAYN through the use of a hierarchy of directed skills, built following the UPSIDE principles. The difference between DIAYN-hier and UPSIDE is that the discriminator reward is computed over the entire directed skill trajectory, while it is guided by the diffusing part for UPSIDE. This introduced baseline can be interpreted as an ablation of UPSIDE without the decoupled structure of policies.

SMM. We consider SMM [Lee+19b] as it is state-of-art in terms of coverage, at least on long-horizon control problems, although [Cam+20] reported its poor performance

in hard-to-explore bottleneck mazes. We tested the regular SMM version, i.e., learning a state density model with a VAE, yet we failed to make it work on the maze domains that we consider. As we use the cartesian (x, y) positions in maze domains, learning the identity function on two-dimensional input data is too easy with a VAE, thus preventing the benefits of using a density model to drive exploration. In our implementation, the exploration bonus is obtained by maintaining a multinomial distribution over "buckets of states" obtained by discretization (as in our coverage computation), resulting in a computation-efficient implementation that is more stable than the original VAE-based method. Note that the state distribution is computed using states from past-but-recent policies as suggested in the original paper.

EDL. We consider EDL [Cam+20] with the strong assumption of an available state distribution oracle (since replacing it by SMM does not lead to satisfying results in presence of bottleneck states as shown in [Cam+20, page 7]: "We were unable to explore this type of maze effectively with SMM"). In our implementation, the oracle samples states uniformly in the mazes avoiding the need to handle a complex exploration, but this setting is not realistic when facing unknown environments.

B.3.2 Architecture and Hyperparameters

The architecture of the different methods remains the same in all our experiments, except that the number of hidden units changes across considered environments. For UPSIDE, flat UPSIDE (i.e., UPSIDE with a tree depth of 1), DIAYN, DIAYN-curr, DIAYN-hier and SMM the multiple policies do not share weights, however EDL policies all share the same network because of the constraint that the policy embedding z is learnt in a supervised fashion with the VQ-VAE rather than the unsupervised RL objective. We consider decoupled actor and critic optimized with the TD3 algorithm [FHM18] though we also tried SAC [Haa+18] which showed equivalent results than TD3 with harder tuning.[1] The actor and the critic have the same architecture that processes observations with a two-hidden layers (of size 64 for maze environments and 256 for control environments) neural networks. The discriminator is a two-hidden (of size 64) layer model with output size the number of skills in the tree.

[1]For completeness, we report here the performance of DIAYN-SAC in the continuous mazes: DIAYN-SAC with $N_Z = 10$ on Bottleneck maze: 21.0 ($\pm$ 0.50); on U-maze: 17.5 ($\pm$ 0.75), to compare with DIAYN-TD3 with $N_Z = 10$ on Bottleneck maze: 17.67 ($\pm$ 0.57); on U-maze: 14.67 ($\pm$ 0.42). We thus see that DIAYN-SAC fails to cover the state space, performing similarly to DIAYN-TD3 (albeit over a larger range of hyperparameter search, possibly explaining the slight improvement).

Common (for all methods and environments) optimization hyper-parameters:

- Discount factor: $\gamma = 0.99$
- $\sigma_{\text{TD3}} = \{0.1, 0.15, 0.2\}$
- Q-functions soft updates temperature $\tau = 0.005$
- Policy Adam optimizer with learning rate $lr_{\text{pol}} = \{1e^{-3}, 1e^{-4}\}$
- policy inner epochs $K_{\text{pol}} = \{10, 100\}$
- policy batch size $B_{\text{pol}} = \{64\}$
- Discriminator delay: $J = \{1, 10\}$
- Replay buffer maximum size: $1e6$
- $K_{\text{initial}} = 1e3$

We consider the same range of hyper-parameters in the downstream tasks.

`UPSIDE`, `DIAYN` and `SMM` variants (common for all environments) optimization hyper-parameters:

- Discriminator batch size $B_{\text{discr}} = 64$
- Discriminator Adam optimizer with learning rate $lr_{\text{discr}} = \{1e^{-3}, 1e^{-4}\}$
- discriminator inner epochs $K_{\text{discr}} = \{10, 100\}$
- Discriminator delay: $J = \{1, 10\}$
- State buffer size $B = 10H$ where the diffusing part length H is environment-specific.

`EDL` optimization hyper-parameters: We kept the same as [Cam+20]. The VQ-VAE's architecture consists of an encoder that takes states as an input and maps them to a code with 2 hidden layers with 128 hidden units and a final linear layer, and the decoder takes the code and maps it back to states also with 2 hidden layers with 128 hidden units. It is trained on the oracle state distribution, then kept fixed during policy learning. Contrary to `UPSIDE`, `DIAYN` and `SMM` variants, the reward is stationary.

- $\beta_{\text{commitment}} = \{0.25, 0.5\}$
- VQ-VAE's code size 16
- VQ-VAE batch size $B_{\text{vq-vae}} = \{64, 256\}$
- total number of epochs: 5000 (trained until convergence)
- VQ-VAE Adam optimizer with learning rate $lr_{\text{vq-vae}} = \{1e^{-3}, 1e^{-4}\}$

Maze specific hyper-parameters:

- $K_{\text{steps}} = 5e4$ (and in time 10 minutes)

- $T = H = 10$
- Max episode length $H_{\text{max}} = 200$
- Max number of interactions $T_{\text{max}} = 1e^7$ during unsupervised pre-training and downstream tasks.

Control specific optimization hyper-parameters:

- $K_{\text{steps}} = 1e5$ (and in time 1 hour)
- $T = H = 50$
- Max episode length $H_{\text{max}} = 250$
- Max number of interactions $T_{\text{max}} = 1e^7$ during unsupervised pre-training and downstream tasks.

  Note that hyperparameters are kept fixed for the downstream tasks too.

B.3.3 Experimental protocol

We now detail the experimental protocol that we followed, which is common for both `UPSIDE` and baselines, on all environments. It consists in the following three stages:

Unsupervised pre-training phase. Given an environment, each algorithm is trained without any extrinsic reward on $N_{\text{unsup}} = 3$ seeds which we call *unsupervised seeds* (to account for the randomness in the model weights' initialization and environment stochasticity if present). Each training lasts for a maximum number of T_{max} environment steps (split in episodes of length H_{max}). This protocol actually favors the baselines since by its design, `UPSIDE` may decide to have fewer environment interactions than T_{max} thanks to its termination criterion (triggered if it cannot fit any more policies); for instance, all baselines where allowed $T_{\text{max}} = 1e7$ on the maze environments, but `UPSIDE` finished at most in $1e6$ environment steps fitting in average 57 and 51 policies respectively for the Bottleneck Maze and U-Maze.

Model selection. For each unsupervised seed, we tune the hyper-parameters of each algorithm according to a certain performance metric. For the baselines, we consider the cumulated intrinsic reward (as done in e.g., [Str+21]) averaged over stochastic roll-outs. For `UPSIDE`, `DIAYN-hier` and `DIAYN-curr`, the model selection criterion is the number of consolidated policies, i.e., how many policies were η-discriminated

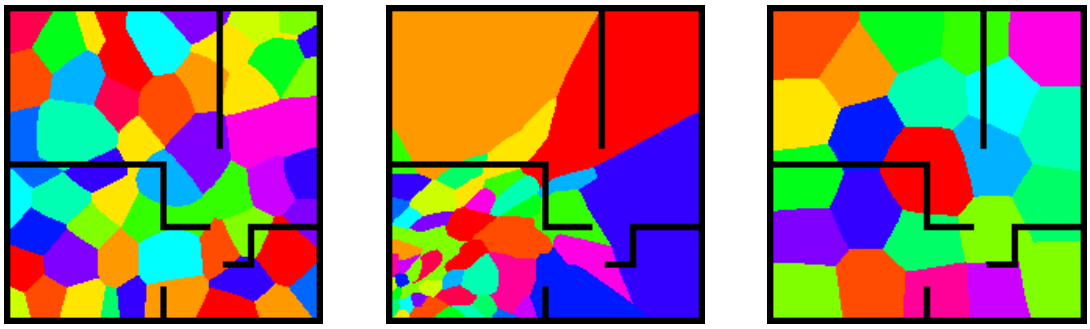
during their training stage. For each method, we thus have as many models as seeds, i.e. N_{unsup}.

Downstream tasks. For each algorithm, we evaluate the N_{unsup} selected models on a set of tasks. All results on downstream tasks will show a performance averaged over the N_{unsup} seeds.

- **Coverage.** We evaluate to which extent the state space has been covered by discretizing the state space into buckets (10 per axis on the continuous maze domains) and counting how many buckets have been reached. To compare the global coverage of methods (and also to be fair w.r.t. the amount of injected noise that may vary across methods), we roll-out for each model its associated deterministic policies.

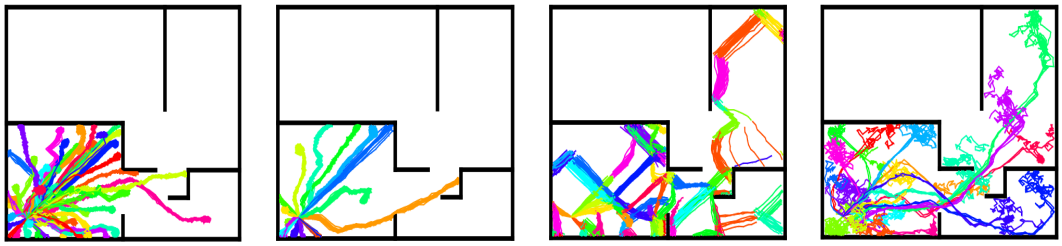
- **Fine-tuning on goal-reaching task.** We consider goal-oriented tasks in the discounted episodic setting where the agent needs to reach some unknown goal position within a certain radius (i.e., the goal location is unknown until it is reached once) and with sparse reward signal (i.e., reward of 1 in the goal location, 0 otherwise). The environment terminates when goal is reached or if the number of timesteps is larger than H_{max}. The combination of *unknown goal location and sparse reward* makes the exploration problem very challenging, and calls upon the ability to first cover (for goal finding) and then navigate (for reliable goal reaching) the environment efficiently. To evaluate performance in an exhaustive manner, we discretize the state space into $B_{\text{goal}} = 14$ buckets and we randomly sample $N_{\text{goal}} = 3$ from each of these buckets according to what we call *goal seeds* (thus there are $B_{\text{goal}} \times N_{\text{goal}} = 10$ different goals in total). For every goal seed, we initialize each algorithm with the set of policies learned during the unsupervised pre-training. We then roll-out each policy during N_{explo} episodes to compute the cumulative reward of the policy, and select the best one to fine-tune. On `UPSIDE`, we complete the selected policy (of length denoted by L) by replacing the diffusing skill with a skill whose length is the remaining number of interactions left, i.e. $H_{\text{max}} - L$. The ability of selecting a good policy is intrinsically linked to the coverage performance of the model, but also to few-shot adaptation. Learning curves and performance are averaged over *unsupervised seeds*, *goal seeds*, and over roll-outs of the stochastic policy. Since we are in the discounted episodic setting, fine-tuning makes sense, to reach as fast as possible the goal. This is particularly important as `UPSIDE`, because of its tree policy structure, can reach the goal sub-optimally w.r.t the discount. On the maze environments, we consider all unsupervised pre-training baselines as well as "vanilla" baselines trained from scratch during the downstream tasks: `TD3`

[FHM18] and `ICM` [Pat+17]. In the Ant environment, we also consider $N_{\text{goal}} = 3$ and $B_{\text{goal}} = 14$ in the $[-8, 8]^2$ square.



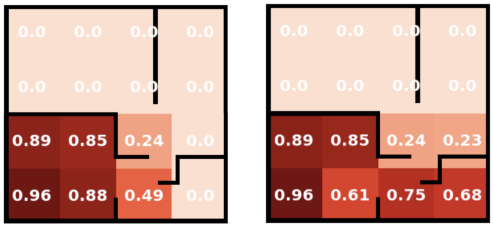
(a) UPSIDE discriminator (b) DIAYN discriminator (c) EDL VQ-VAE

Figure B.7 – Environment divided in colors according to the most likely latent variable Z, according to (*from left to right*) the discriminator learned by UPSIDE, the discriminator learned by DIAYN and the VQ-VAE learned by EDL. Contrary to DIAYN, UPSIDE's optimization enables the discriminator training and the policy training to catch up to each other, thus nicely clustering the discriminator predictions across the state space. EDL's VQ-VAE also manages to output good predictions (recall that we consider the EDL version with the strong assumption of the available state distribution oracle, see [Cam+20]), yet the skill learning is unable to cover the entire state space due to exploration issues and sparse rewards.



(a) SMM (b) DIAYN-curr (c) DIAYN-hier (d) Flat UPSIDE

Figure B.8 – Complement to Figure 4.2: Visualization of the policies learned on the Bottleneck Maze for the remaining methods.



(a) SMM (b) TD3

Figure B.9 – Complement of Fig. 4.5: Heatmaps of downstream task performance after fine-tuning for the remaining methods.

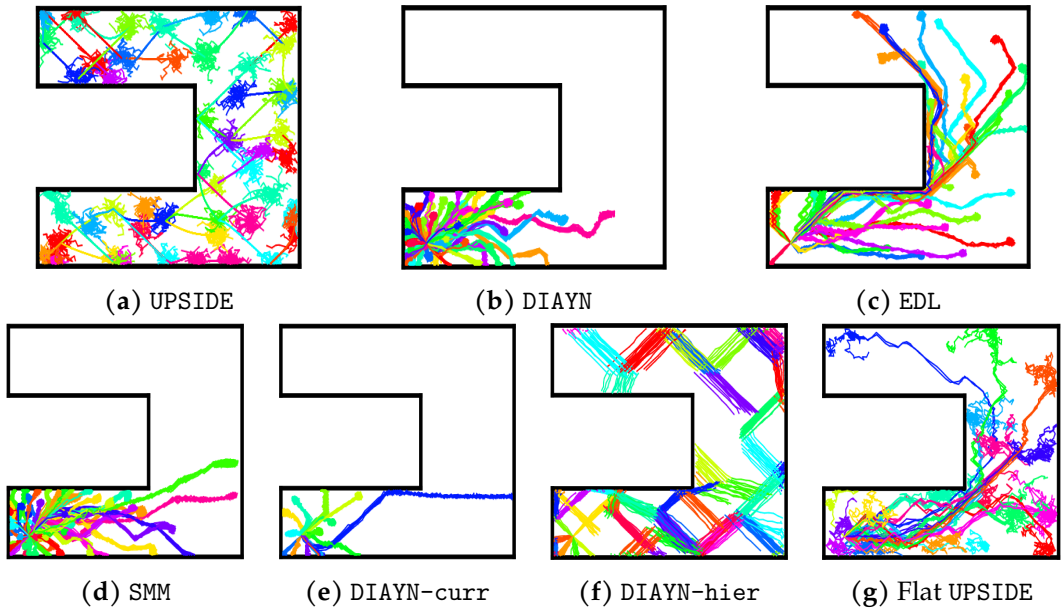


Figure B.10 – Visualization of the policies learned on U-Maze. This is the equivalent of Fig. 4.2 for U-Maze.

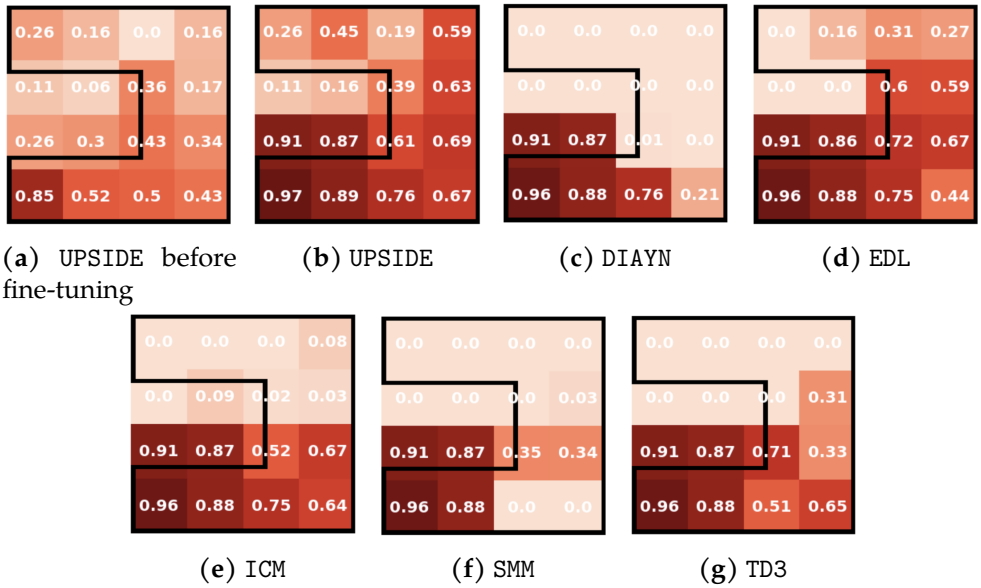
B.4 Additional Experiments

B.4.1 Additional results on Bottleneck Maze

Here we include **1)** Fig. B.7 for an analysis of the predictions of the discriminator (see caption for details); **2)** Fig. B.8 for the policy visualizations for the remaining methods (i.e., those not reported in Fig. 4.2; **3)** Fig. B.9 for the downstream task performance for the remaining methods (i.e., those not reported in Fig. 4.5).

B.4.2 Additional Results on U-Maze

Fig. B.10 visualizes the policies learned during the unsupervised phase (i.e., the equivalent of Fig. 4.2 for the U-Maze), and Fig. B.11 reports the heatmaps of downstream task performance (i.e., the equivalent of Fig. 4.5 for the U-Maze). The conclusion is the same as on the Bottleneck Maze described in Sect. 4.5: UPSIDE clearly outperforms all the baselines, both in coverage (Fig. B.10) and in unknown goal-reaching performance (Fig. B.11) in the downstream task phase.



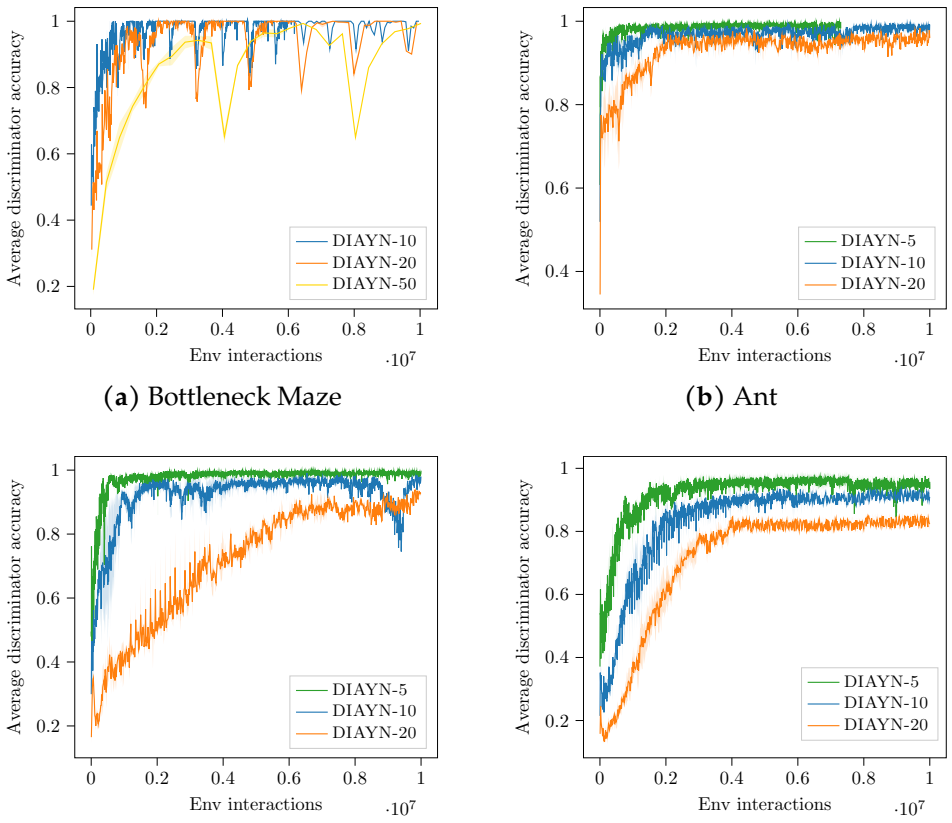
(**a**) UPSIDE before fine-tuning (**b**) UPSIDE (**c**) DIAYN (**d**) EDL

(**e**) ICM (**f**) SMM (**g**) TD3

Figure B.11 – Heat maps of downstream task performance on U-Maze. This is the equivalent of Fig. 4.5 for U-Maze.

B.4.3 Analysis of the discriminability

In Fig. B.12 (see caption) we investigate the average discriminability of DIAYN depending on the number of policies N_Z.



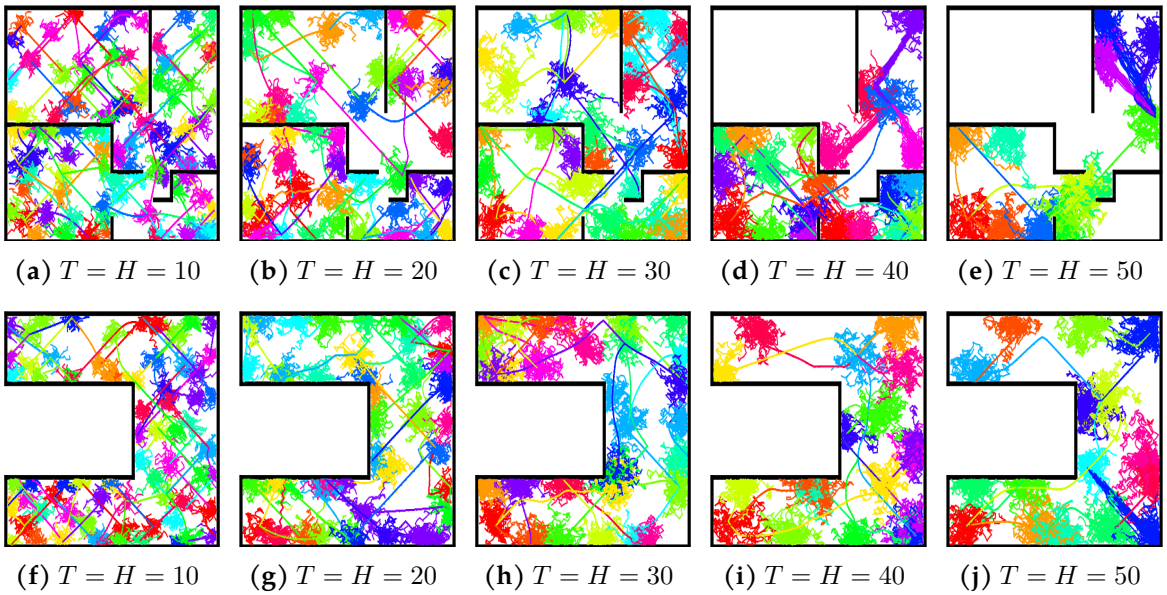
(**a**) Bottleneck Maze (**b**) Ant

(**c**) Half-Cheetah (**d**) Walker2d

Figure B.12 – Average discriminability of the DIAYN-N_Z policies. The smaller N_Z is, the easier it is to obtain a close-to-perfect discriminability. However, even for quite large N_Z (50 for mazes and 20 in control environments), DIAYN is able to achieve a good discriminator accuracy, most often because policies learn how to "stop" in some state.

B.4.4 Ablation on the lengths T and H of the UPSIDE policies

Our ablation on the mazes in Fig. B.13 investigates the sensitiveness of UPSIDE w.r.t. T and H, the lengths of the directed skills and diffusing parts of the policies. For the sake of simplicity, we kept $T = H$. It shows that the method is quite robust to reasonable choices of T and H, i.e., equal to 10 (as done in all other experiments) but also 20 or 30. Naturally, the performance degrades if T, H are chosen too large w.r.t. the environment size, in particular in the bottleneck maze which requires "narrow" exploration, thus composing disproportionately long skills hinders the coverage. For $T = H = 50$, we recover the performance of flat UPSIDE.



(a) $T = H = 10$ (b) $T = H = 20$ (c) $T = H = 30$ (d) $T = H = 40$ (e) $T = H = 50$

(f) $T = H = 10$ (g) $T = H = 20$ (h) $T = H = 30$ (i) $T = H = 40$ (j) $T = H = 50$

Figure B.13 – Ablation on the length of UPSIDE policies (T, H): Visualization of the policies learned on the Bottleneck Maze (*top*) and the U-Maze (*bottom*) for different values of T, H. (*Right table*) Coverage values (according to the same procedure as in Table 4.2). Recall that T and H denote respectively the lengths of the directed skill and of the diffusing part of an UPSIDE policy.

UPSIDE	Bottleneck Maze	U-Maze
$T = H = 10$	85.67 (±1.93)	71.33 (±0.42)
$T = H = 20$	87.33 (±0.42)	67.67 (±1.50)
$T = H = 30$	77.33 (±3.06)	68.33 (±0.83)
$T = H = 40$	59.67 (±1.81)	57.33 (±0.96)
$T = H = 50$	51.67 (±0.63)	58.67 (±1.26)

B.4.5 Fine-tuning Results on Half-Cheetah and Walker2d

In Sect. 4.5, we reported the fine-tuning results on Ant. We now focus on Half-Cheetah and Walker2d, and similarly observe that UPSIDE largely outperforms the baselines:

	UPSIDE	TD3	DIAYN
Half-Cheetah	174.93 (±1.45)	108.67 (±25.61)	0.0 (±0.0)
Walker2d	46.29 (±3.09)	14.33 (±1.00)	2.13 (±0.74)

We note that the fine-tuning experiment on Half-Cheetah exactly corresponds to the standard Sparse-Half-Cheetah task, by rewarding states where the x-coordinate is larger than 15. Meanwhile, Sparse-Walker2d provides a reward as long as the x-coordinate is larger than 1 and the agent is standing up. Our fine-tuning task on Walker2d is more challenging as we require the x-coordinate to be larger than 4. Yet the agent can be rewarded even if it is not standing up, since our downstream task

is purely goal-reaching. We indeed interestingly noticed that UPSIDE may reach the desired goal location yet not be standing up (e.g., by crawling), which may occur as there is no incentive in UPSIDE to remain standing up, only to be discriminable.

Appendix C

Complements on Chapter 6

C.1 Details on the training data

In Tab. C.1 we provide the detailed set of parameters used in our data generator. The probabilities of the unary operators were selected to match the natural frequencies appearing in the Feynman dataset.

In Fig. C.1, we show the statistics of the data generation.The number of expressions diminishes with the input dimension and number of unary operators because of the higher likelihood of generating out-of-domain inputs. One could easily make the distribution uniform by enforcing to retry as long as a valid example is not found, however we find empirically that having more easy examples than hard ones eases learning and provides better out-of-domain generalization, which is our ultimate goal.

In Fig. C.2, we show some examples of the input distributions generated by our multimodal approach. Notice the diversity of shapes obtained by this procedure.

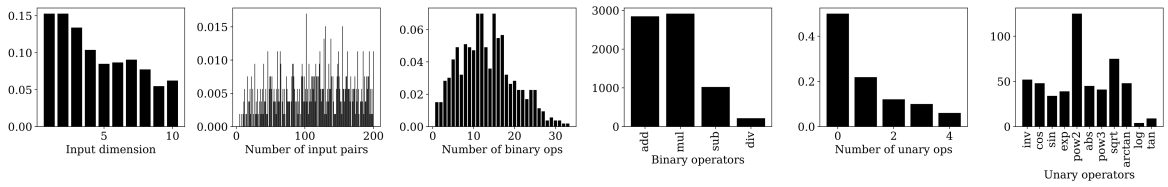


Figure C.1 – Statistics of the synthetic data. We calculated the latter on $10,000$ generated examples.

Table C.1 – Parameters of our generator.

Parameter	Description	Value
D_{max}	Max input dim	10
$\mathcal{D}_{\text{aff}}$	Distrib of (a,b)	sign $\sim \mathcal{U}\{-1,1\}$, mantissa $\sim \mathcal{U}(0,1)$, exponent $\sim \mathcal{U}(-2,2)$
b_{max}	Max binary ops	$5+D$
O_b	Binary operators	add:1, sub:1, mul:1
u_{max}	Max unary ops	5
O_u	Unary operators	inv:5, abs:1, sqr:3, sqrt:3, sin:1, cos:1, tan:0.2, atan:0.2, log:0.2, exp:1
N_{min}	Min number of points	$10d_{\text{in}}$
N_{max}	Max number of points	200
k_{max}	Max num clusters	10

C.2 Examples of expressions generated

Below, we give some typical examples of the expressions generated by our random generator for dimensions between 1 and 3 (Tab. C.2), as well as some examples of equations from the Feynman dataset (Tab. C.3).

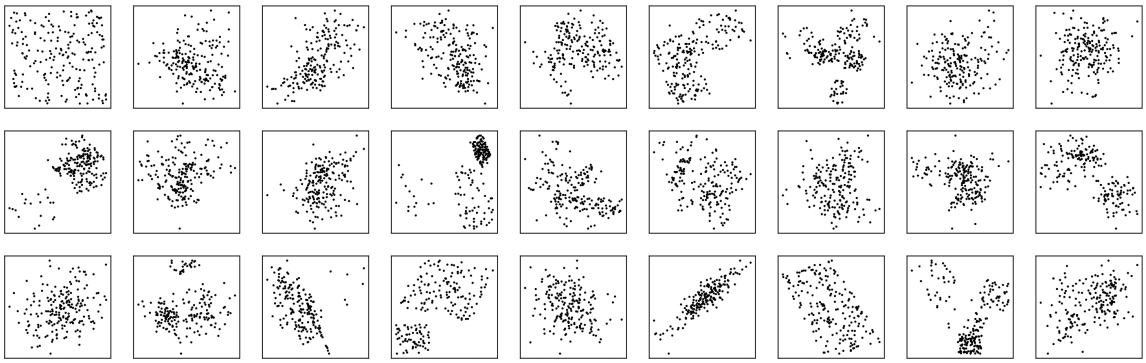


Figure C.2 – Diversity of the input distributions generated by the multimodal approach. Here we show distributions obtained for $D = 2$.

$Cx_0 - \frac{Cx_0}{Cx_0+C} - C\left(Cx_0\left(Cx_0+C\right)^3+C\right)^3+C$
$Cx_0 + C\left\|Cx_0+C\right\| + C$
$Cx_0x_1\left(Cx_0 + C\log\left(Cx_0+C\right)\right)+C$
$C + x_0\left(-Cx_1+Cx_2\right)^2$
$C + x_0x_2\left(Cx_0+Cx_1 - C\sin\left(Cx_0 - C\sin\left(Cx_1x_2+C\right)\right)\right)$
$Cx_0 + Cx_2 + C + \frac{C}{Cx_1\sqrt{Cx_0+C}+Ce^{Cx_0-\frac{C}{Cx_2+C}}}$
$Cx_0 + Cx_1 + C$
$C\left(Cx_0+C\right)^3 + C - \frac{C}{\frac{Cx_1}{Cx_2+C}+C}$
$C\sin\left(Cx_0+C\right)+C+\frac{C}{Cx_1+C}$
$-Cx_1\left(\frac{Cx_2}{Cx_0+C}+C\right)^2 + Cx_2 + C\sin\left(Cx_1+C\right) + C\tan\left(Cx_0+C\right)+C$
$C + x_0\left(Cx_0 - Cx_1 + C\left\|Cx_1+C\right\|\right)$
$C + x_2\left(Cx_0+Cx_1\right)\left(Cx_0+\frac{C}{Cx_0+C}\right)$
$C - x_2\sin\left(Cx_0+Cx_1-\frac{Cx_2}{Cx_2+C}\right)$
$-Cx_0 + C\left(\frac{Cx_0}{Cx_0+C}+C\right)^2+C$
$Cx_0\left(Cx_0x_1x_2+\frac{C}{Cx_2+C}\right)+Cx_1+C\left(Cx_1+C\right)^2+C$

Table C.2 – A few examples of equations from our random generator.

C.3 Does memorization occur?

It is natural to ask the following question: due to the large amount of data seen during training, is our model simply memorizing the training set ? Answering this question involves computing the number of possible functions which can be generated. To estimate this number, calculating the number of possible skeleton N_s is insufficient, since a given skeleton can give rise to very different functions according to the sampling of the constants, and even for a given choice of the constants, the input points $\{x\}$ can be sampled in many different ways.

Nonetheless, we provide the lower bound N_s as a function of the number of nodes in Fig. C.3, using the equations provided in [LC19]. For small expressions (up to four operators), the number of possible expressions is lower or similar to than the number of expressions encountered during training, hence one cannot exclude the possibility that some expressions were seen several times during training, but with different realizations due to the initial conditions. However, for larger expressions, the number of possibilities is much larger, and one can safely assume that the expressions encountered at test time have not been seen during training.

$\frac{\sqrt{2}e^{-\frac{\theta^2}{2}}}{2\sqrt{\pi}}$	$\frac{\sqrt{2}e^{-\frac{\theta^2}{2\sigma^2}}}{2\sqrt{\pi}\sigma}$	$\frac{\sqrt{2}e^{-\frac{(\theta-\theta_1)^2}{2\sigma^2}}}{2\sqrt{\pi}\sigma}$
$\sqrt{\left(-x_1+x_2\right)^2+\left(-y_1+y_2\right)^2}$	$\frac{Gm_1m_2}{(-x_1+x_2)^2+(-y_1+y_2)^2+(-z_1+z_2)^2}$	$\frac{m_0}{\sqrt{1-\frac{v^2}{c^2}}}$
$x_1y_1+x_2y_2+x_3y_3$	$Nn\mu$	$\frac{q_1q_2}{4\pi\varepsilon r^2}$
$\frac{q_1}{4\pi\varepsilon r^2}$	$Ef q_2$	$q\left(Bv\sin\left(\theta\right)+Ef\right)$
$\frac{m\left(u^2+v^2+w^2\right)}{2}$	$Gm_1m_2\left(\frac{1}{r_2}-\frac{1}{r_1}\right)$	gmz
$\frac{k_{spring}x^2}{2}$	$\frac{-tu+x}{\sqrt{1-\frac{u^2}{c^2}}}$	$\frac{t-\frac{ux}{c^2}}{\sqrt{1-\frac{u^2}{c^2}}}$
$\frac{m_0v}{\sqrt{1-\frac{v^2}{c^2}}}$	$\frac{u+v}{1+\frac{uv}{c^2}}$	$\frac{m_1r_1+m_2r_2}{m_1+m_2}$
$Fr\sin\left(\theta\right)$	$mrv\sin\left(\theta\right)$	$\frac{mx^2\left(\omega^2+\omega_0^2\right)}{4}$
$\frac{q}{C}$	$\arcsin\left(n\sin\left(\theta_2\right)\right)$	$\frac{1}{\frac{n}{d_2}+\frac{1}{d_1}}$
$\frac{\omega}{c}$	$\sqrt{x_1^2-2x_1x_2\cos\left(\theta_1-\theta_2\right)+x_2^2}$	$\frac{Int_0\sin^2\left(\frac{n\theta}{2}\right)}{\sin^2\left(\frac{\theta}{2}\right)}$
$\arcsin\left(\frac{lambd}{dn}\right)$	$\frac{a^2q^2}{6\pi c^3\varepsilon}$	$\frac{4\pi Ef^2c\varepsilon\omega^4r^2}{3\left(\omega^2-\omega_0^2\right)^2}$
$\frac{Bqv}{p}$	$\frac{\omega_0}{1-\frac{v}{c}}$	$\frac{\omega_0\left(1+\frac{v}{c}\right)}{\sqrt{1-\frac{v^2}{c^2}}}$
$\frac{h\omega}{2\pi}$	$I_1+I_2+2\sqrt{I_1I_2}\cos\left(\delta\right)$	$\frac{\varepsilon h^2}{\pi mq^2}$

Table C.3 – A few examples of equations from the Feynman dataset.

C.4 Attention maps

A natural question is whether self-attention based architectures are optimally suited for symbolic regression tasks. In Fig. C.4, we show the attention maps produced by the encoder of our Transformer model, which contains 4 layers avec 16 attention heads (we only keep the first 8 for the sake of space). In order to make the maps readable, we consider one-dimensional inputs and sort them in ascending order.

The attention plots demonstrate the complementarity of the attention heads. Some focus on specific regions of the input, whereas others are more spread out. Some are concentrated along the diagonal (focusing on neighboring points), whereas others are concentrated along the anti-diagonal (focusing on far-away points.

Most strikingly, the particular features of the functions studied clearly stand out in the attention plots. Focus, for example, on the 7th head of layer 2. For the exponential function, it focuses on the extreme points (near -1 and 1); for the inverse function, it focuses on the singularity around the origin; for the sine function, it reflects the peri-

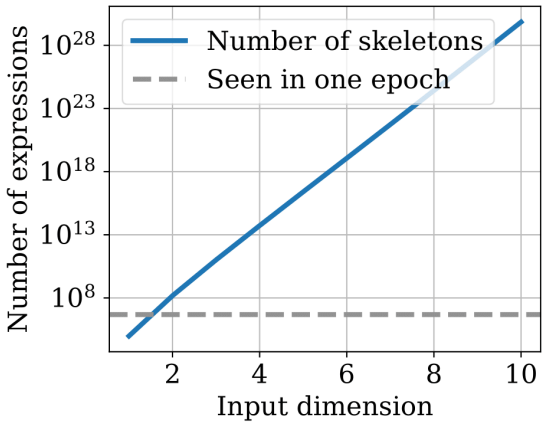


Figure C.3 – Our models only see a small fraction of the possible expressions during training. We report the number of possible skeletons for each number of operators. Even after a hundred epochs, our models have only seen a fraction of the possible expressions with more than 4 operators.

odicity, with evenly spaces vertical lines. The same phenomenology can be acrossed is several other heads.

C.5 Additional in-domain results

Fig. C.5, we present a similar ablation as Fig. 6.4 of the main text but using the R^2 score as metric rather than accuracy.

C.6 Additional out-of-domain results

Complexity-accuracy In Fig. C.6, we display a Pareto plot comparing accuracy and formula complexity on SRBench datasets.

Jin benchmark In Fig. C.7, we show the predictions of our model on the functions provided in [Jin+20a]. Our model gets all of them correct except for one.

Black-box datasets In Fig. C.8, we display the results of our model on the black-box problems of SRBench.

Strogatz datasets Each of the 14 datasets from the ODE-Strogatz benchmark is the trajectory of a 2-state system following a first-order ordinary differential equation (ODE). Therefore, the input data has a very particular, time-ordered distribution, which differs significantly from that seen at train time. Unsurprisingly, Fig. C.9 shows that our model performs somewhat less well to this kind of data in comparison with GP-based methods.

Ablation on input dimension In Fig. C.10, we show how the performance of our model depends on the dimensionality of the inputs on Feynamn and black-box datasets.

Ablation on decoding strategy In Fig. C.11, we display the difference in performance using two decoding strategies.

Ablation on the use of i) mixture of distributions during training, ii) scaling during inference It is generally observed that Transformers struggle to generalize out-of-distribution, especially in mathematical tasks [Wel+22]. We demonstrate that both i) and ii) are necessary to handle datasets involving input distributions that are (i) neither gaussian nor uniform, and (ii) vary across wide ranges of scales.

For (i), we provide in Fig. C.12 a qualitative example on a model trained with Gaussian and uniform distributions that a distribution-shift at test time can cause failure. Consider the function $y = x_1 \cos(x_0 + x_1)$. Recall the model was trained on datapoints sampled from distributions either $N(0, 1)$ or $\mathcal{U}([-1, 1])$. As we sample 100 datapoints from $\mathcal{U}([0, 6])$, we see the E2E model makes good predictions, whereas, adding 100 datapoints, sampled uniformly between $\mathcal{U}([-7, 5])$, degrades the model prediction.

For (ii), we also provide in Fig. C.13 a qualitative example of failure on the same function $y = x_1 \cos(x_0 + x_1)$ and datapoints when scaling is not used. We additionally report results on SRBench evaluation for our E2E model without scaling in Table C.4 and show that changes in scales during inference put transformers outside their comfort zone.

Table C.4 – The importance of scaling at inference for transformer-based approaches.

Refinement	Scaler	Feynman [mean R2>0.99]	Black-box [median R2]
With	With	**0.84**	**0.87**
Without	With	0.78	0.70
With	Without	0.53	0.64
Without	Without	0.06	0.46

Results on SRBench show that scaling is necessary to achieve competitive results. Note that refinement of constants can improve the performance of the unscaled prediction, however it is not enough to even catch up with the E2E without refinement model.

C.7 Extended comparison with prior work.

SymbolicGPT [Val+21] and NSRS [Big+21] both train Transformers to predict function skeletons with other tokenization strategies. SymbolicGPT is prone to training instabilities when considering functions with high value variations and NSRS' architecture is not able to scale to high dimensions because the tokenized input grows linearly with the input dimension. [dAs+22] also predicts skeletons but focus on the problem of inferring one-dimensional recurrence relations from small sets of points in case only, while we estimate functions of many variables over larger sets of points.

Only NSRS [Big+21] provides a pre-trained model, but it was only trained on problems with dimensions ≤ 3, corresponding to a very small subset of SRBench. Note however that even at these low dimensions, NSRS seems to perform less well than our model: the authors report an accuracy (defined at R2> 0.95) on the Feynman datasets of ≈ 0.75 in their appendix (Fig. 9), whereas we get ≈ 0.84 on R2> 0.99 on all dimensions.

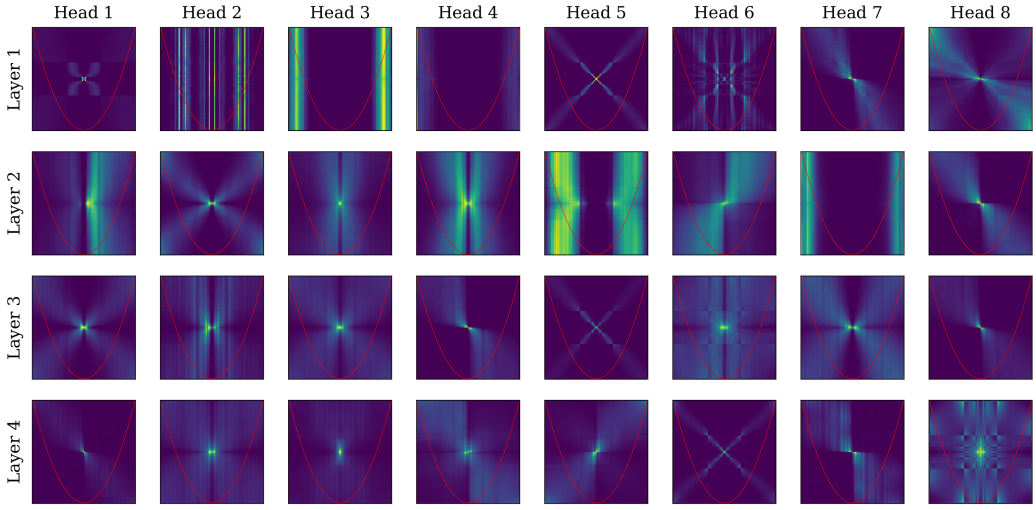
The benchmark we used for our comparison, SRBench, is currently the most extensive and up-to-date benchmark for SR, and provides comparisons with other DL-based methods such as DSR [Pet+19]. Note also that the ablation of Tables 6.1 and 6.2 and Fig. 6.4 (in-domain), as well as Figures 6.5 and C.8 (out-of-domain) are provided to show the benefit of the E2E approach over skeleton approaches.

Note that an other end-to-end approach[Vas+22b], very similar to ours, was released two months after us.

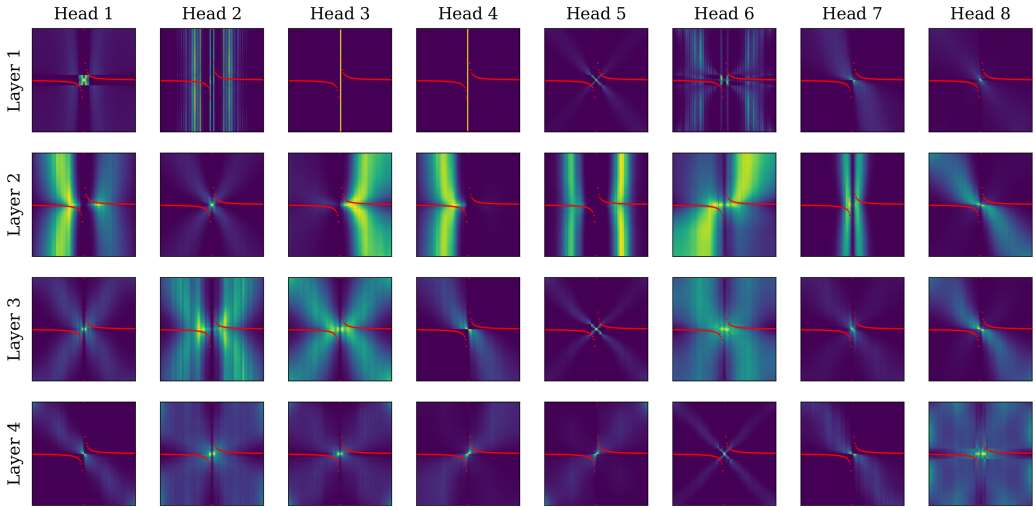
C.8 Extension of our model to dimension > 10.

Our method still remains improvable in scaling to larger dimensions. The reason we restricted our model to dimension ≤ 10 is that the input sequence length becomes prohibitively long beyond, and that generating high-dimensional functions in an unbiased way becomes increasingly tricky. Nonetheless, since the objective of SR is to output interpretable formulas, we argue that SR is most useful for moderately low dimensional problems. For example, $1-10$ dimensional problems already cover a large class of physical systems : for instance, point objects can be represented by their position, speed and mass, 7 parameters. Additionally, in many real world problems where more than 10 features are available, some of the features are often irrelevant or heavily correlated. To mitigate this, one typically carries out feature selection before modeling the data.

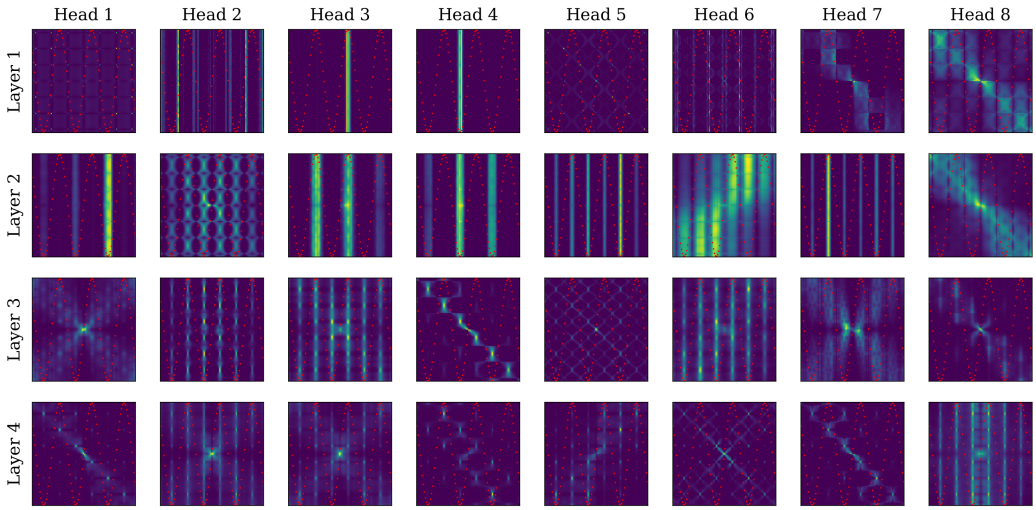
We tested our model on the high-dimensional problems of SRBench (up to 1000 input dimensions), by feeding to our model only the 10 features most correlated with the output. This naive strategy already obtained encouraging results (with a median R2 score of 0.72, to compare with 0.58 for DSR and 0.55 for gplearn, but still well below Operon which stands at 0.91).



(a) $f(x) = x^2$



(b) $f(x) = 1/x$



(c) $f(x) = \sin(10x)$

Figure C.4 – Attention maps reveal distinctive features of the functions considered. We presented the model 1-dimensional functions with 100 input points sorted in ascending order, in order to better visualize the attention. We plotted the self-attention maps of the first 8 (out of 16) heads of the Transformer encoder, across all four layers. We see very distinctive patterns appears: exploding areas for the exponential, the singularity at zero for the inverse function, and the periodicity of the sine function.

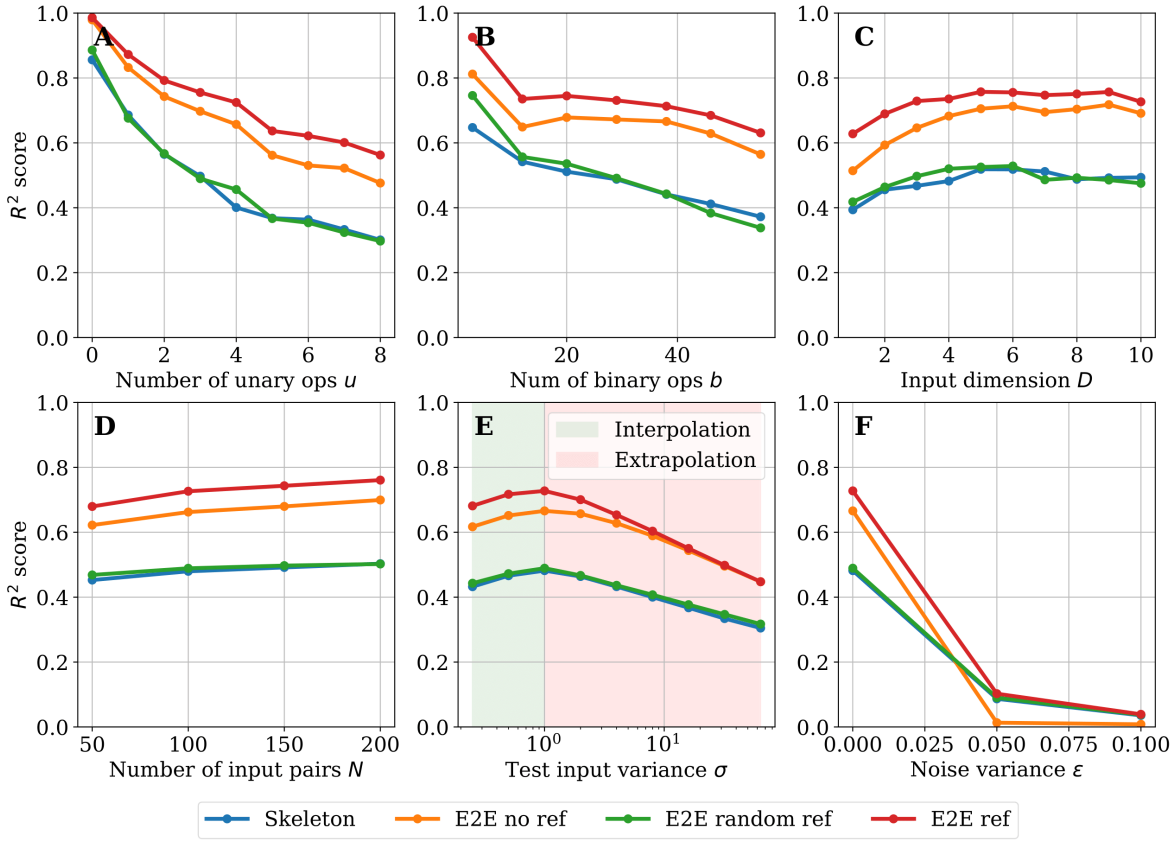


Figure C.5 – Ablation over the function difficulty (top row) and input difficulty (bottom row). We plot the R^2 score (Eq. 6.1). **A:** number of unary operators. **B:** number of binary operators. **C:** input dimension. **D:** Low-resource performance, evaluated by varying the number of input points. **E:** Extrapolation performance, evaluated by varying the variance of the inputs. **F:** Robustness to noise, evaluated by varying the multiplicative noise added to the labels.

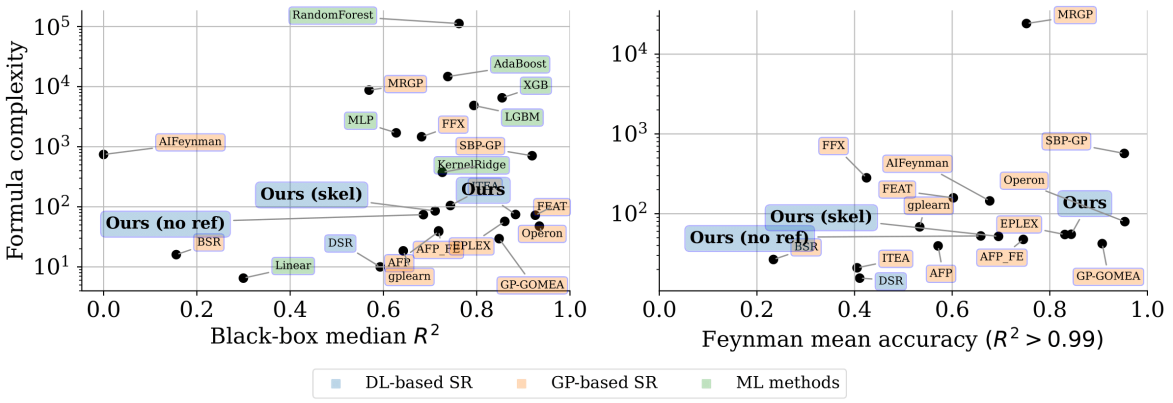
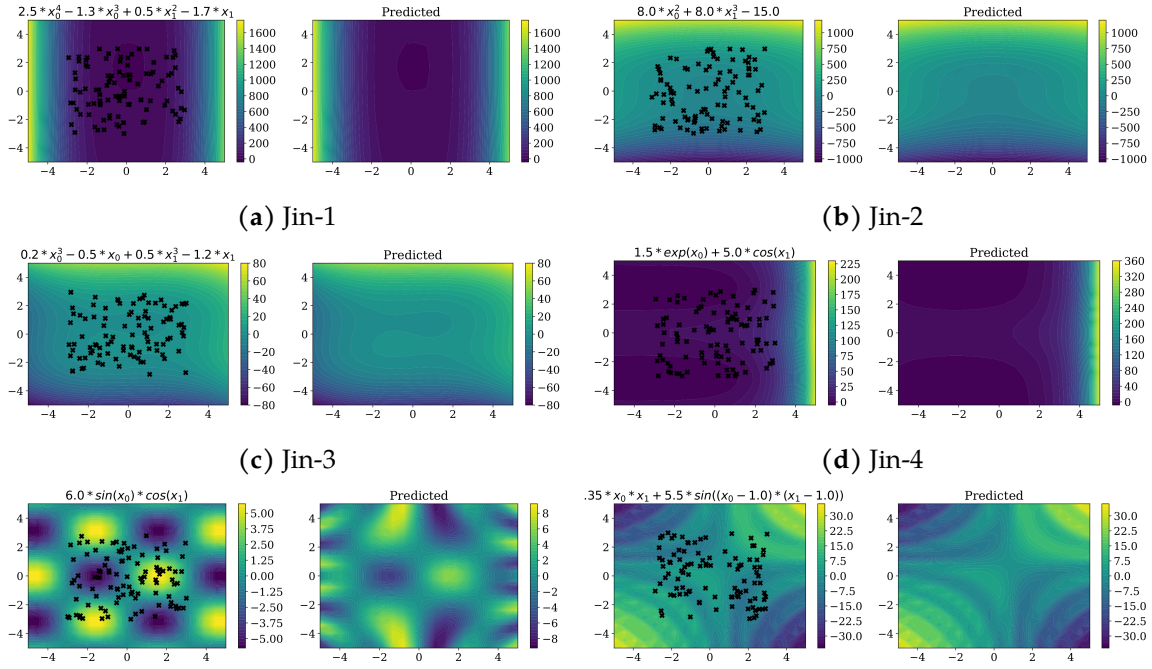


Figure C.6 – Complexity-accuracy pareto plot. Pareto plot comparing the average test performance and formula complexity of our models with baselines provided by the SRbench benchmark [La +21a], both on Feynman SR problems [UT20] and black-box regression problems. We use colors to distinguish three families of models: deep-learning based SR, genetic programming-based SR and classic machine learning methods (which do not provide an interpretable solution).



(a) Jin-1 (b) Jin-2

(c) Jin-3 (d) Jin-4

(e) Jin-5 (f) Jin-6

Figure C.7 – Illustration of our model on a few benchmark datasets from the litterature. We show the prediction of our model on six 2-dimensional datasets presented in [Jin+20a] and used as a comparison point in a few recent works [Mun+21]. The input points are marked as black crosses. Our model retrieves the correct expression in all but one of the cases: in Jin5, the prediction matches the input points correctly, but extrapolates badly.

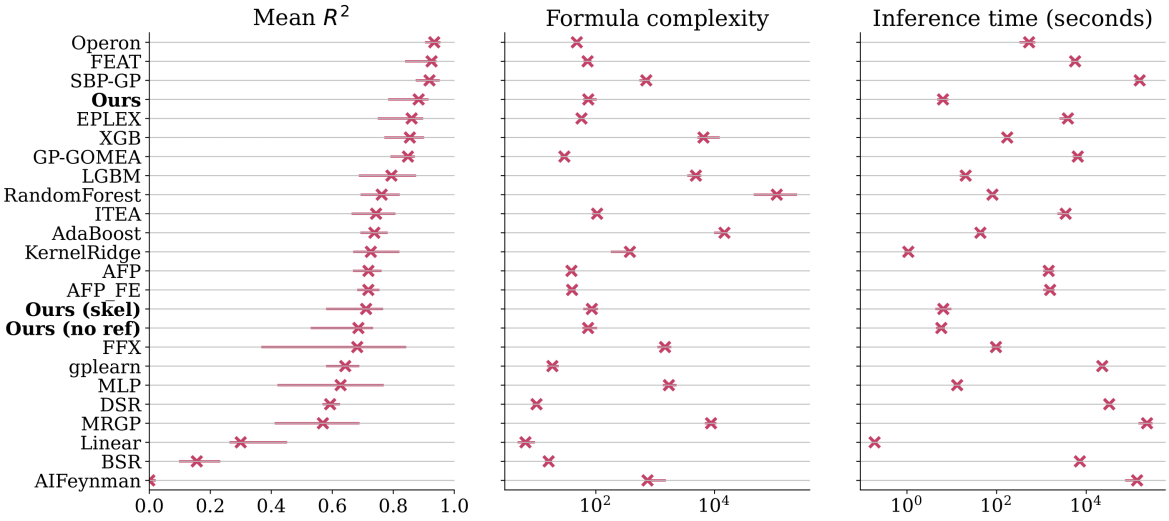


Figure C.8 – Performance metrics on black-box datasets.

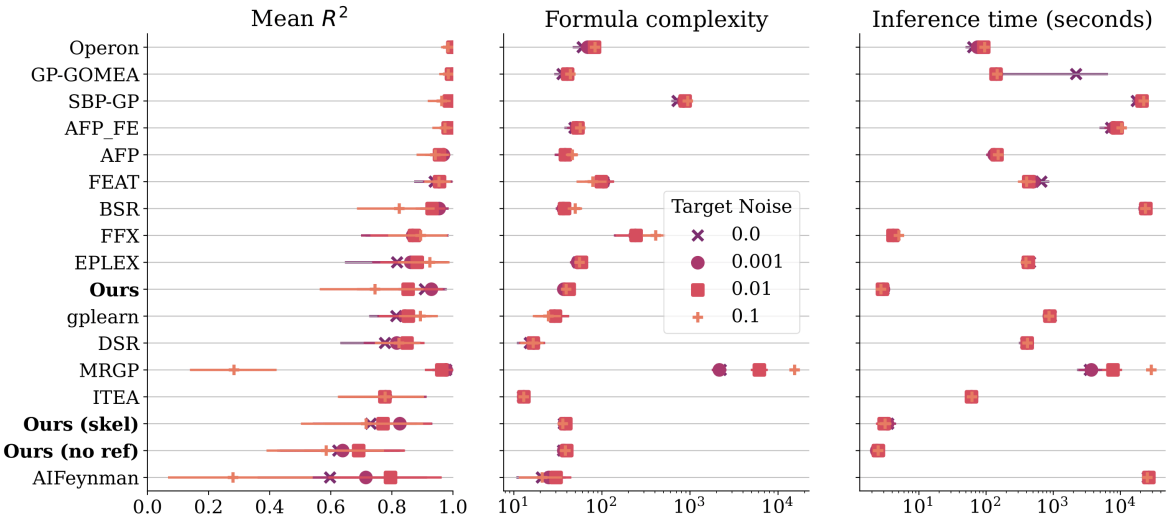
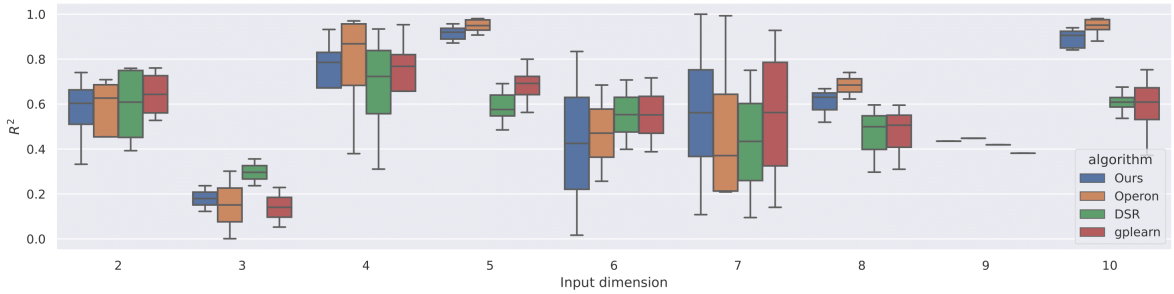
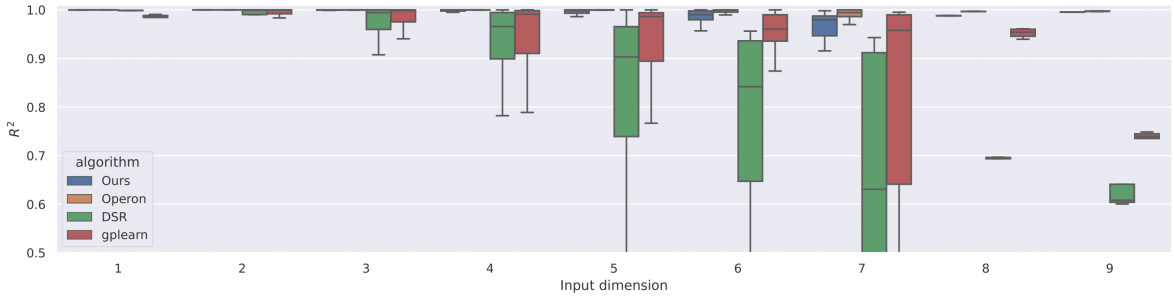


Figure C.9 – Performance metrics on Strogatz datasets.



(**a**) Black-box



(**b**) Feynman

Figure C.10 – Performance metrics on SRBench, separated by input dimension.

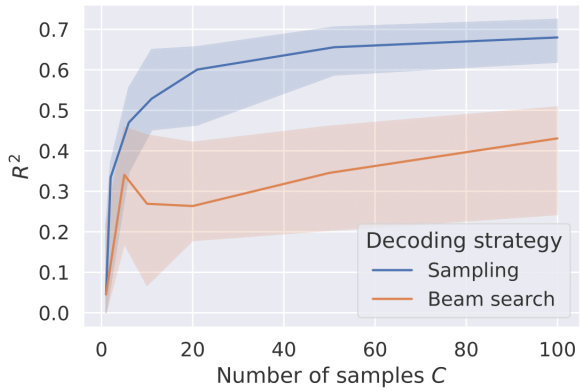


Figure C.11 – Median R^2 of our method without refinement on black-box datasets when $B = 1$, varying the number of decoded function samples. The beam search [WR16] used in [Big+21] leads to low-diversity candidates in our setup due to expressions differing only by small modifications of the coefficients.

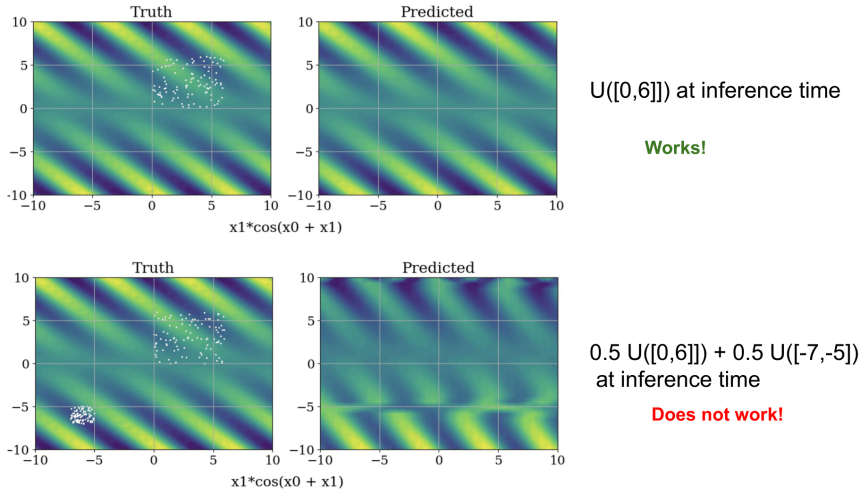


Figure C.12 – Transformers do not generalize well to distribution-shift.

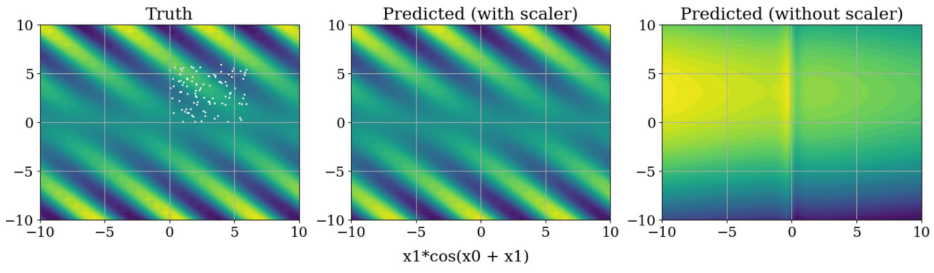


Figure C.13 – Transformers do not generalize well to scale-shift.

Appendix D

Complements on Chapter 7

D.1 Data generation

Ground-truth expressions. To generate a large synthetic dataset with examples $(\mathcal{D}, f^*)$, we first sample N observations/features $\mathbf{X} \in \mathrm{R}^{N \times D}$ where D is uniformly sampled between 1 and 10 with a mixture of Gaussians as in [Kam+22], then consider sampling: a) an empty unary-binary tree from [LC19] generator with between 5 and 25 internal nodes, b) assign a random operator on nodes and either a variable $\{x_d\}_{d \leq D}$ or float constant drawn from a normal distribution on leaves. We then simplify the ground-truth expression with SymPy.

The only difference with [Kam+22] lies in the fact that the generated expressions are much smaller by not enforcing all variables to appear sampled expressions, therefore letting the model learn the ability to do feature selection, as well as to having much less constants (they apply linear transformations with probability 0.5 on all nodes/leaves), therefore providing more interpretable expressions (model size is divided by 2).

Example mutations. From a ground-truth expression f^*, we generate a sequence of example mutations that go from the empty expression to f^* by iteratively removing parts of the expression tree. To do so, at any given step, we randomly pick an internal node such that the size of the subtree argument B is large enough, and apply the backward mutation, i.e. adding an edge between the parent node and the remaining child node A. When the remaining expression is too small, the mutation's operator becomes $0 \to B$ where B is the remaining expression.

D.2 Data representation

As done in most [Kam+22], floats are represented in base 10 foating-point notation, rounded to four significant digits, and encoded as sequences of 3 tokens: their sign, mantissa (between 0 and 9999), and exponent (from `E-100` to `E100`). Expressions are represented by their Polish notations, i.e. the breadth-first search walk, with numerical constants are represented as explained above. A dataset is represented by the concatenation of all tokenized (x_i, y_i) pairs where vectors representation is just the flattenized tokens of each dimension value. The combination of both the expression and dataset yields the representation of states by concatenating both representations and using special separators between the expression f and dataset $\mathcal{D}$. Actions are represented by the concatenation (with special separators) of i) the node index (integer in base 10) on which to apply the mutation, ii) the operator token, iii) the tokenized expression B if the operator is binary.

D.3 Model details

Since the number of tokens transformers can use as context is limited by memory considerations and possible learnable long-range dependencies, we restrict to 100 the number of input data points used as input to M, the subset being sampled at each expansion. We also train our model on datasets with at most 10 variables, as in [Kam+22].

D.4 Exceptions detected ablations

Earlier `DGSR` work has showed that Transformer models [Vas+17] were able to learn almost perfect semantics of symbolic expressions, resulting in 99% of valid expressions in [Kam+22]. In this work, we noticed that our policy model was also able to manipulate the expression structure quite well as around 90% of sampled mutations resulted in valid expressions. Similarly we noticed that malformed mutations, e.g., invalid index on which expression node to apply an operation, argument B that cannot be parsed or or absent B when operator is binary, represent less than 1% of errors.

As mentioned in section 7.2.3, we constrain our model to discard expressions that are too complex (more than 60 operators/variables/constants) or have nested

complicated operators (e.g. $\cos(\cos(X))$, $\log(\log(X))$), in order to promote simpler expressions as explained, resulting in a great trade-off between accuracy and complexity as shown in section 7.3.2. Note that it would be possible to enforce a greater trade-off between accuracy and complexity, e.g. using strategies mentioned in [La +21a]. This results in 9% of expressions being discarded because these constraints are violated.

D.5 Search details

D.5.1 Definition of satisfactory expressions

We concluded from preliminary experiments that considering f^* to be satisfactory if $R^2 \geq 0.99$ performed best as it provided high-quality samples while dramatically reducing search times compared to perfect fitting. Systematically estimating what accuracy can be achieved with a given complexity is not possible without, e.g., resorting to another algorithm that operates under the same complexity constraints and can act as an oracle. We also tried estimating accuracy on validation set of points by running XGBoost on a train set, however for `SRBench` datasets, accuracy can greatly vary according to the way the dataset is split.

D.5.2 Constant optimization

We use Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS) with batch size 256, early stopping if accuracy does not improve after 10 iterations and a timeout of 1 second.

D.5.3 Search hyper-parameters

In this work, we employ a distributed learning architecture, similar to that proposed in [Lam+22]. Since the optimal hyper-parameters of search are not necessarily the same for all datasets, the controller samples these hyper-parameters from pre-defined ranges for each different search trial:

The proposed model depends on many of hyper-parameters, specifically those pertaining to the decoding of the mutation model and the search process. Determining the optimal values for these hyper-parameters poses a significant challenge in practice for several reasons. Firstly, the model is in a state of continual evolution, and thus

the optimal hyper-parameter values may also change over time. For instance, if the model exhibits an excessive level of confidence in its predictions, it may be necessary to increase the decoding temperature to promote diversity in mutations. Secondly, the optimal values of the hyper-parameters may be specific to the dataset under consideration. Finally, the sheer number of hyper-parameters to be tuned, coupled with the high computational cost of each experiment, renders the task of determining optimal values infeasible. To circumvent these issues, rather than fixing the hyper-parameters to a specific value, they are sampled from predefined ranges at the beginning of each search trial. The specific decoding parameters and the distribution utilized are as follows:

- Number of samples K per expansion. Distribution: uniform on range [8,16].
- Temperature used for decoding. Distribution: uniform on range [0.5, 1.0].
- Length penalty: length penalty used for decoding. Distribution: uniform on range [0, 1.2].
- Depth penalty: an exponential value decay during the backup-phase, decaying with depth to favor breadth or depth. Distribution: uniform on discrete values [0.8, 0.9, 0.95, 1].
- Exploration: the exploration constant p_{uct}. 1

Appendix E

Complements on Chapter 8

E.1 Operon

We used Operon [BKK20b] with the following allowed operators `add`,`sub`,`mul`,`div`,`sin`,`cos`, `pow` for nodes. Leaves are either variables, i.e. a state feature or a numerical constant. We use the following hyper-parameters: 5 local iterations, a population of size 5000, a total of 10000 generations and 10 threads.

E.2 Experiment details

We perform our experiments with 1 GPU and 1 CPU.

E.2.1 Toy example

Model hyper-parameters.

MLP-specific hyper-parameters. We consider a deterministic neural network with 4 hidden layers of size 200 with SiLU activation, trained with Adam optimizer during a maximum of 2000 epochs with batch size 256 and patience epochs 25 (meaning training stops when loss/ evaluation score does not progress more than 0.01 relatively), learning rate $7.5e-4$ and weight decay $3e-5$. Inputs are normalized.

Symbolic-specific hyper-parameters. We use Operon with 5 local iterations, 10000 generations, 10 threads, population size 5000 and allowed symbols are

```
add,sub,mul,div,constant,variable,sin,exp,abs
```

.

Action optimizer. We use CEM with planning horizon 3, 10 iterations, elite ratio 0.1, population size 1000, alpha 0.1 and clipped normal action distribution. Out of the ensemble of 7 predictive models, only the 3 elite (w.r.t evaluation set) ones are used.

E.3 Additional results

E.3.1 CartPole

Model hyper-parameters.

MLP-specific hyper-parameters. Same as for the toy example.

Symbolic-specific hyper-parameters. We use Operon with 5 local iterations, 10000 generations, 10 threads, population size 5000 and allowed symbols are

```
add,sub,mul,div,constant,variable,sin,cos,pow
```

.

Action optimizer. We use CEM with planning horizon 15, 5 iterations, elite ratio 0.1, population size 350, alpha 0.1. Out of the ensemble of 7 predictive models, only the 3 elite (w.r.t evaluation set) ones are used.

Analysis of results

Interestingly, one can notice in Fig. 8.3 the performance (in terms of reward and model error) fluctuate a bit as the number of interactions grows (contrary to Fig. E.1). This can be explained by the fact that random samples are the most informative transitions in terms on environment understanding, whereas on-policy transitions

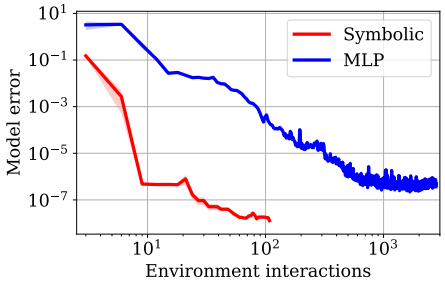


Figure E.1 – Model error of the MLP and symbolic regressors (averaged over different seeds) on data generated by a random policy.

(whose proportion grows during learning) are all located in a very narrow part of the state-space (pole standing still)

Appendix F

Predicting recurrences

F.1 Introduction

Given the sequence [1,2,4,7,11,16], what is the next term? Humans usually solve such riddles by noticing patterns in the sequence. In the easiest cases, one can spot a function: [1,4,9,16,25] are the first five squares, so the n-th term in the series is $u_n = n^2$, and the next one is 36. Most often however, we look for relations between successive terms: in the sequence [1,2,4,7,11,16], the differences between successive values are 1, 2, 3, 4, and 5, which makes it likely that the next term will be $16 + 6 = 22$. Mathematically, we are inferring the recurrence relation $u_n = u_{n-1} + n$, with $u_0 = 1$.

In all cases, we handle such problems as symbolic regressions: starting from a sequence of numbers, we try to discover a function or a recurrence relation that they satisfy, and use it to predict the next terms. This can lead to very challenging problems as the complexity of the unknown recurrence relation $u_n = f(n, \{u_i\}_{i<n})$ increases, e.g. $u_n = \tan^{-1}(u_{n-3}) \exp(\cos(n^2))$.

In this work, we train neural networks to infer the recurrence relation f from the observation of the first terms of the sequence. The majority of studies in machine learning for symbolic regression have focused on non-recurrent functions, i.e. expressions of the form $y = f(x)$. Recurrence relations provide a more general setting, which gives a deeper account of the underlying process: if the sequence corresponds to successive time steps, the recurrence relation is a discrete time differential equation for the system considered.

OEIS	Description	First terms	Predicted recurre
A000792	$a(n) = \max\{(n-i)a(i), i < n\}$	1, 1, 2, 3, 4, 6, 9, 12, 18, 27	$u_n = u_{n-1} + u_{n-3} - u_n$
A000855	Final two digits of 2^n	1, 2, 4, 8, 16, 32, 64, 28, 56, 12	$u_n = (2u_{n-1})\%1$
A006257	Josephus sequence	0, 1, 1, 3, 1, 3, 5, 7, 1, 3	$u_n = (u_{n-1} + n)\%(n$
A008954	Final digit of $n(n+1)/2$	0, 1, 3, 6, 0, 5, 1, 8, 6, 5	$u_n = (u_{n-1} + n)\%$
A026741	$a(n) = n$ if n odd, $n/2$ if n even	0, 1, 1, 3, 2, 5, 3, 7, 4, 9	$u_n = u_{n-2} + n//(u_n$
A035327	n binary, switch 0's and 1's, then decimal	1, 0, 1, 0, 3, 2, 1, 0, 7, 6	$u_n = (u_{n-1} - n)\%($
A062050	n-th chunk contains numbers $1, ..., 2^n$	1, 1, 2, 1, 2, 3, 4, 1, 2, 3	$u_n = (n\%(n - u_{n-1}$
A074062	Reflected Pentanacci numbers	5, -1, -1, -1, -1, 9, -7, -1, -1, -1	$u_n = 2u_{n-5} - u_n$

Table F.1 – Our integer model yields exact recurrence relations on a variety of interesting OEIS sequences. Predictions are based on observing the first 25 terms of each sequence.

Constant	Approximation	Rel. error
0.3333	$(3 + \exp(-6))^{-1}$	10^{-5}
0.33333	$1/3$	10^{-5}
3.1415	$2\arctan(\exp(10))$	10^{-7}
3.14159	π	10^{-7}
1.6449	$1/\arctan(\exp(4))$	10^{-7}
1.64493	$\pi^2/6$	10^{-7}
0.123456789	$10/9^2$	10^{-9}
0.987654321	$1 - (1/9)^2$	10^{-11}

Table F.2 – Our float model learns to approximate out-of-vocabulary constants with its own vocabulary. We obtain the approximation of each constant C by feeding our model the 25 first terms of $u_n = Cn$.

F.1.1 Contributions

We show that transformers can learn to infer a recurrence relation from the observation of the first terms of a sequence. We consider both sequences of integers and floats, and train our models on a large set of synthetic examples.

We first demonstrate that our symbolic regression model can predict complex recurrence relations that were not seen during training. We also show that those recurrence relations can be used to extrapolate the next terms of the sequence with better precision than a "numeric" model of similar architecture, trained specifically for extrapolation.

We then test the out-of-domain generalization of our models. On a subset of the Online Encyclopedia of Integer Sequences, our integer models outperform built-in Mathematica functions, both for sequence extrapolation and recurrence prediction, see Table F.1 for some examples. We also show that our symbolic float model is

Expression u_n	Approximation $\hat{u}_n$	Comment
$\operatorname{arcsinh}(n)$	$\log(n+\sqrt{n^2+1})$	Exact
$\operatorname{arccosh}(n)$	$\log(n+\sqrt{n^2-1})$	Exact
$\operatorname{arctanh}(1/n)$	$\frac{1}{2}\log(1+2/n)$	Asymptotic
$\operatorname{catalan}(n)$	$u_{n-1}(4-6/n)$	Asymptotic
$\operatorname{dawson}(n)$	$\frac{n}{2n^2-u_{n-1}-1}$	Asymptotic
$\operatorname{j0}(n)$ (Bessel)	$\frac{\sin(n)+\cos(n)}{\sqrt{\pi n}}$	Asymptotic
$\operatorname{i0}(n)$ (mod. Bessel)	$\frac{e^n}{\sqrt{2\pi n}}$	Asymptotic

Table F.3 – Our float model learns to approximate out-of-vocabulary functions with its own vocabulary. For simple functions, our model predicts an exact expression in terms of its operators; for complex functions which cannot be expressed, our model manages to predict the first order of the asymptotic expansion.

capable of predicting approximate expressions for out-of-vocabulary functions and constants (e.g. $\operatorname{bessel0}(x) \approx \frac{\sin(x)+\cos(x)}{\sqrt{\pi x}}$ and $1.644934 \approx \pi^2/6$), see Tables F.2, F.3 for more examples.

We conclude by discussing potential limitations of our models and future directions.

Demonstration We provide an interactive demonstration of our models at https://symbolicregression.metademolab.com.

Code An open-source implementation of our code will be released publicly at https://github.com/facebookresearch/recur.

F.1.2 Related work

AI for maths The use of neural networks for mathematics has recently gained attention: several works have demonstrated their surprising reasoning abilities [Sax+19; Cob+21], and have even sparked some interesting mathematical discoveries [Dav+21]. In particular, four types of tasks have recently been tackled in the deep learning literature.

First, converting a symbolic expression to another symbolic expression. Direct examples are integration and differentation [LC19], expression simplification [All+17] or equation solving [ASA18]. Second, converting a symbolic expression to numerical

data. This includes predicting quantitative properties of a mathematical structure, for example the local stability of a differential system [CHL20]. Third, converting numerical data to numerical data using mathematical rules. Examples range from learning basic arithmetics [KS15; Tra+18] to linear algebra [Cha21]. Fourth, converting numerical data to a symbolic expression: this is the framework of symbolic regression, which we will be focusing on.

Symbolic regression Two types of approaches exist for symbolic regression, which one could name selection-based and pretraining-based.

In selection-based approaches, we only have access to one set of observations: the values of the function we are trying to infer. Typical examples of this approach are evolutionnary programs, which have long been the *de facto* standard for symbolic regression [AB00; SL09; Mur+14; MWB95], despite their computational cost and limited performance. More recently, neural networks were used following this approach [SLM18; Kim+20; Pet+19].

In pretraining-based approaches, we train neural networks on a large dataset containing observations from many different function [Big+21; Val+21], hoping that the model can generalize to new expressions. Although the pretraining is computationally expensive, inference is much quicker as one simply needs to perform a forward pass, rather than search through a set of functions. In this work, we choose this approach.

Recurrent sequences All studies on symbolic regression cited above consider the task of learning non-recurrent functions from their values at a set of points. Our contribution is, to the best of our knowledge, the first to tackle the setup of recurrent expressions. Although this includes non-recurrent expressions as a special case, one slight restriction is that the inputs need to be sampled uniformly. Hence, instead of feeding the model with input-output pairs $\{(x_i, y_i)\}$, we only need to feed it the terms of the sequence $\{u_i\}$. Another important difference is that order of these terms matters, hence permutation invariant representations [Val+21] cannot be used.

Integer sequences, in particular those from the Online Encyclopedia of Integer Sequences (OEIS) [Slo07], have been studied using machine learning methods in a few recent papers. Wu [Wu18] trains various classifiers to predict the label of OEIS sequences, such as "interesting", "easy", or "multiplicative". Ryskina and Knight [RK21] use embeddings trained on OEIS to investigate the mathematical properties of integers. Ragni and Klein [RK11] use fully connected networks to predict the next

term of OEIS sequences (a numeric rather than symbolic regression task). Nam et al. [NKJ18] investigate different architectures (most of them recurrent networks) for digit-level numeric regression on sequences such as Fibonacci, demonstrating the difficulty of the task.

F.2 Methods

Broadly speaking, we want to solve the following problem: given a sequence of n points $\{u_0, \dots, u_{n-1}\}$, find a function f such that for any $i \in \mathbb{N}$, $u_i = f(i, u_{i-1}, \dots, u_{i-d})$, where d is the recursion degree. Since we cannot evaluate at an infinite number of points, we declare that a function f is a solution of the problem if, given the first n_{input} terms in the sequence, it can predict the next n_{pred} following ones.

Under this form, the problem is underdetermined: given a finite number of input terms, an infinite number of recurrence relations exist. However in practice, one would like the model to give us the simplest solution possible, following Occam's principle. This is generally ensured by the fact that simple expressions are more likely to be generated during training.

Integer vs. float We consider two settings for the numbers in the sequences: integers and floats. For integer sequences, the recurrence formula only uses operators which are closed in $\mathbb{Z}$ (e.g. $+, \times, \text{abs}$, modulo and integer division). For float sequences, additional operators and functions are allowed, such as real division, $\exp$ and $\cos$ (see table F.4for the list of all used operators). These two setups are interesting for different reasons. Integer sequences are an area of strong interest in mathematics, particular for their relation with arithmetics. The float setup is interesting to see how our model can generalize to a larger set of operators, which provides more challenging problems.

Symbolic vs. numeric We consider two tasks: symbolic regression and numeric regression. In symbolic regression, the model is tasked to predict the recurrence relation the sequence was generated from. At test time, this recurrence relation is evaluated by how well it approximates the n_{pred} following terms in the sequence.

In numeric regression, is tasked to directly predict the values of the n_{pred} following terms, rather than the underlying recurrence relation. At test time, the model predictions are compared with the true values of the sequence.

	Integer	Float
Unary	`abs, sqr, sign, relu`	`abs, sqr, sqrt, inv, log, exp, sin, cos, tan, atan`
Binary	`add, sub, mul, intdiv, mod`	`add, sub, mul, div`

Table F.4 – Operators used in our generators.

F.2.1 Data generation

All training examples are created by randomly generating a recurrence relation, randomly sampling its initial terms, then computing the next terms using the recurrence relation. More specifically, we use the steps below:

1. Sample the **number of operators** o between 1 and o_{max}, and build a unary-binary tree with o nodes, as described in [LC19]. The number of operators determines the difficulty of the expression.

2. Sample the **nodes** of the tree from the list of operators in table F.4. Note that the float case uses more operators than the integer case, which makes the task more challenging by expanding the problem space.

3. Sample the **recurrence degree** d between 1 and d_{max}, which defines the recurrence depth: for example, a degree of 2 means that u_{n+1} depends on u_n and u_{n-1}.

4. Sample the **leaves** of the tree: either a constant, with probability p_{const}, or the current index n, with probability p_n, or one of the previous terms of the sequence u_{n-i}, with $i \in [1, d]$, with probability p_{var}.

5. Recalculate the true recurrence degree d considering the deepest leaf u_{n-i} sampled during the previous step, then sample d **initial terms** from a random distribution $\mathcal{P}$.

6. Sample l between l_{min} and l_{max} and compute the next l terms of the sequence using the initial conditions. The total **sequence length** is hence $n_{input} = d_{\text{eff}} + l$.

We provide the values of the parameters of the generator in Table F.5. Note that in the last step, we interrupt the computation if we encounter a term larger than 10^{100}, or outside the range of one of the operators: for example, a division by zero, or a negative square root.

Parameter	Description	Value
d_{max}	Max degree	6
o_{max}	Max number of operators	10
l_{min}	Min length	5
l_{max}	Max length	30
p_{const}	Prob of constant leaf	$1/3$
p_{index}	Prob of index leaf	$1/3$
p_{var}	Prob of variable leaf	$1/3$
$\mathcal{P}$	Distrib of first terms	$\mathcal{U}(-10, 10)$

Table F.5 – Hyperparameters of our generator.

Limitations One drawback of our approach is poor out-of-distribution generalization of deep neural networks, demonstrated in a similar context to ours by [CHL20]: when shown an example which is impossible to be generated by the procedure described above, the model will inevitably fail. For example, as shown in App. F.6, our model performs poorly when the initial terms of the sequence are sampled outside their usual range, which is chosen to be $[-10, 10]$ in these experiments. One easy fix is to increase this range; are more involved one is to use a normalization procedure as described in our follow-up work [Kam+22].

F.2.2 Encodings

Model inputs are sequences of integers or floats. The outputs are recurrence relations for the symbolic task, and sequences of numbers for the numeric task. To be processed by transformers, inputs and outputs must be represented as sequences of tokens from a fixed vocabulary. To this effect, we use the encodings presented below.

Integers We encode integers using their base b representation, as a sequence of $1 + \lceil \log_b |x| \rceil$ tokens: a sign (which also serves as a sequence delimiter) followed by $\lceil \log_b |x| \rceil$ digits from 0 to $b - 1$, for a total vocabulary of $b + 2$ tokens. For instance, $x = -325$ is represented in base $b = 10$ as the sequence `[-, 3, 2, 5]`, and in base $b = 30$ as `[-, 10, 25]`. Choosing b involves a tradeoff between the length of the sequences fed to the Transformer and the vocabulary size of the embedding layer. We choose $b = 10,000$ and limit integers in our sequences to absolute values below 10^{100}, for a maximum of 26 tokens per integer, and a vocabulary of order 10^4 words.

Floats Following Charton [Cha21], we represent float numbers in base 10 floating-point notation, round them to four significant digits, and encode them as sequences of 3 tokens: their sign, mantissa (between `0` and `9999`), and exponent (from `E-100` to `E100`). For instance, $1/3$ is encoded as `[+, 3333, E-4]`. Again, the vocabulary is of order 10^4 words.

For all operations in floating-point representation, precision is limited to the length of the mantissa. In particular, when summing elements with different magnitudes, sub-dominant terms may be rounded away. Partly due to this effect, when approximating complex functions, our symbolic model typically only predicts the largest terms in its asymptotic expansion, as shown in Table F.3. We discuss two methods for increasing precision when needed in Section F.9 of the Appendix.

Recurrence relations To represent mathematical trees as sequences, we enumerate the trees in prefix order, i.e. direct Polish notation, and encode each node as a single autonomous token. For instance, the expression $\cos(3x)$ is encoded as `[cos,mul,3,x]`.

Note that the generation procedure implies that the recurrence relation is not simplified (i.e. expressions like $1 + u_{n-1} - 1$ can be generated). We tried simplifying them using Sympy before the encoding step (see Appendix F.7), but this slows down generation without any benefit on the performance of our models, which turn out to be surprisingly insensitive to the syntax of the formulas.

F.2.3 Experimental details

Similarly to Lample and Charton [LC19], we use a simple Transformer architecture [Vas+17] with 8 hidden layers, 8 attention heads and an embedding dimension of 512 both for the encoder and decoder.

Training and evaluation The tokens generated by the model are supervised via a cross-entropy loss. We use the Adam optimizer, warming up the learning rate from 10^{-7} to 2.10^{-4} over the first 10,000 steps, then decaying it as the inverse square root of the number of steps, following [Vas+17]. We train each model for a minimum of 250 epochs, each epoch containing 5M equations in batches of 512. On 16 GPU with Volta architecture and 32GB memory, one epoch is processed in about an hour.

After each epoch, we evaluate the in-distribution performance of our models on a held-out dataset of 10,000 equations. Unless specified otherwise, we generate expres-

sions using greedy decoding. Note that nothing prevents the symbolic model from generating an invalid expression such as `[add,1,mul,2]`. These mistakes, counted as invalid predictions, tend to be very rare: in models trained for more than a few epochs, they occur in less than 0.1% of the test cases.

Hypothesis ranking To predict recurrence relations for OEIS sequences (Section F.4.1), or the examples displayed in Tables F.1,F.2,F.3, we used a beam size of 10. Usually, hypotheses in the beam are ranked according to their log-likelihood, normalized by the sequence length. For our symbolic model, we can do much better, by ranking beam hypotheses according to how well they approximate the initial terms of the original sequence. Specifically, given a recurrence relation of degree d, we recompute for each hypothesis the n_{input} first terms in the sequence (using the first d terms of the original sequence as initial conditions), and rank the candidates according to how well they approximate these terms. This ranking procedure is impossible for the numerical model, which can only perform extrapolation.

In some sense, this relates our method to evolutionary algorithms, which use the input terms for candidate selection. Yet, as shown by the failure modes presented in Section F.11 of the Appendix, our model is less prone to overfitting since the candidates are not directly chosen by minimizing a loss on the input terms.

F.3 In-domain generalization

We begin by probing the in-domain accuracy of our model, i.e. its ability to generalize to unseen sequences generated with the same procedure as the training data. As discussed in Section F.8 of the Appendix, the diversity of mathematical expressions and the random sampling of the initial terms ensures that almost all the examples presented at test time have not been seen during training: one cannot attribute the generalization to mere memorization.

Prediction accuracy Due to the large number of equivalent ways one can represent a mathematical expression, one cannot directly evaluate the accuracy by comparing (token by token) the predicted recurrence relation to the ground truth. Instead, we use the predicted expression to compute the next n_{pred} terms of the sequence $\{\hat{u}_i\}$, and compare them with those computed from the ground truth, $\{u_i\}$. The prediction

Model	Integer		Float	
	$n_{op} \leq 5$	$n_{op} \leq 10$	$n_{op} \leq 5$	$n_{op} \leq 10$
Symbolic	**92.7**	**78.4**	**74.2**	**43.3**
Numeric	83.6	70.3	45.6	29.0

Table F.6 – Average in-distribution accuracies of our models. We set $\tau = 10^{-10}$ and $n_{pred} = 10$.

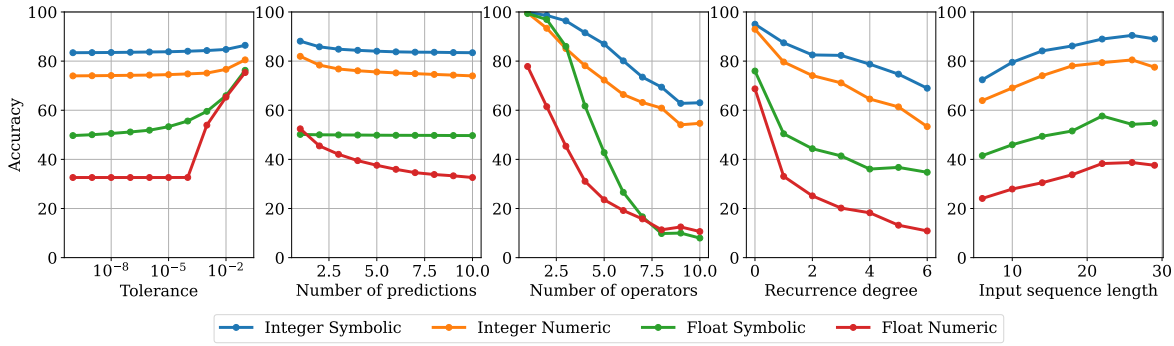


Figure F.1 – The symbolic model extrapolates further and with higher precision than the numeric model. From left to right, we vary the tolerance τ, the number of predictions n_{pred}, the number of operators o, the recurrence degree d and the number of input terms l. In each plot, we use the following defaults for quantities which are not varied: $\tau = 10^{-10}$, $n_{pred} = 10$, $o \in [\![1, 10]\!]$, $d \in [\![1, 6]\!]$, $l \in [\![5, 30]\!]$.

accuracy is then defined as:

$$acc(n_{pred}, \tau) = \mathbb{P}\left(\max_{1 \leq i \leq n_{pred}} \left| \frac{\hat{u}_i - u_i}{u_i} \right| \leq \tau \right) \tag{F.1}$$

By choosing a small enough $\tau \geq 0$ and a large enough n_{pred}, one can ensure that the predicted formula matches the true formula. In the float setup, $\tau = 0$ must be avoided for two reasons: (i) equivalent solutions represented by different expressions may evaluate differently because due to finite machine precision, (ii) setting $\tau = 0$ would penalize the model for ignoring sub-dominant terms which are indetectable due to the finite precision encodings. Hence, we select $\tau = 10^{-10}$ and $n_{pred} = 10$ unless specified otherwise[1]. Occasionally, we will consider larger values of τ, to assess the ability of our model to provide approximate expressions.

[1]For the float numeric model, which can only predict values up to finite precision ε, we round the values of target function to the same precision. This explains the plateau of the accuracy at $\tau < \varepsilon$ in Figure F.1.

Results The average in-distribution accuracies of our models are reported in Table F.6 with $\tau = 10^{-10}$ and $n_{pred} = 10$. Although the float setup is significantly harder that the integer setup, our symbolic model reaches good accuracies in both cases. In comparison, the numeric model obtains worse results, particularly in the float setup. The interested reader may find additional training curves, attention maps and visualizations of the model predictions in Appendix F.11.

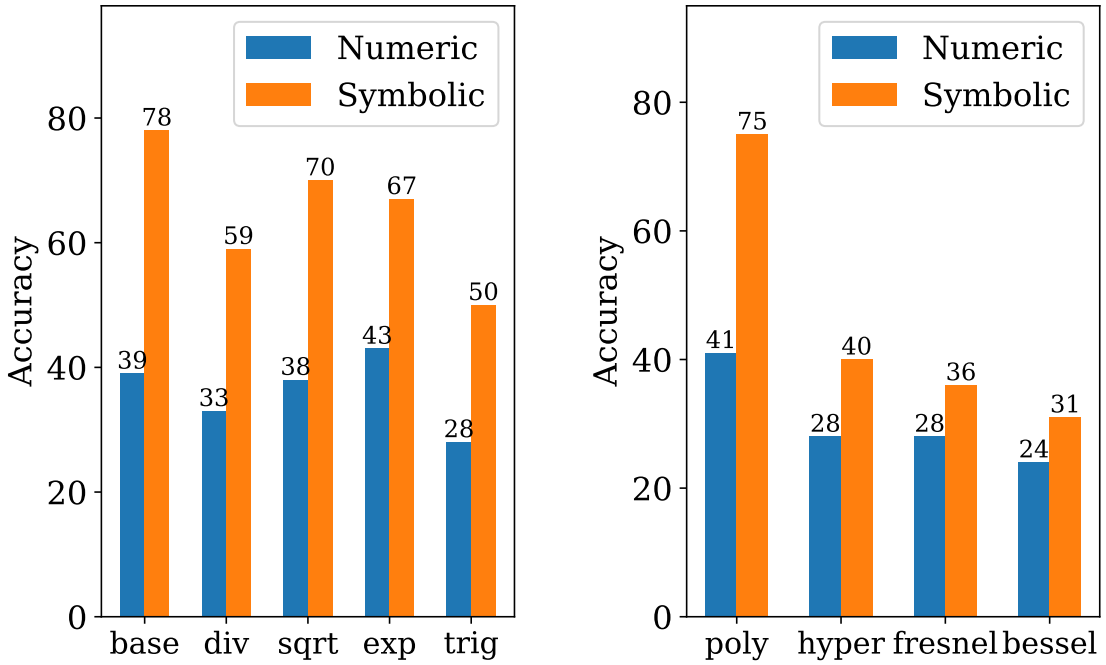
Ablation over the evaluation metric The two first panels of Figure F.1 illustrate how the accuracy changes as we vary the tolerance level τ and the number of predictions n_{pred}. The first important observation is that the symbolic model performs much better than the numeric model at low tolerance. At high tolerance, it still keeps an advantage in the integer setup, and performs similarly in the float setup. This demonstrates the advantage of a symbolic approach for high-precision predictions.

Second, we see that the accuracy of both models degrades as we increase the number of predictions n_{pred}, as one could expect. However, the decrease is less important for the symbolic model, especially in the float setup where the curve is essentially flat. This demonstrates another strong advantage of the symbolic approach: once it has found the correct formula, it can predict the whole sequence, whereas the precision of the numeric model deteriorates as it extrapolates further.

Ablation over the example difficulty The three last panels of Figure F.1 decompose the accuracy of our two models along three factors of difficulty: the number of operators o[2], the recurrence degree d and the sequence length n_{input} (see Section F.2.1).

Unsurprisingly, accuracy degrades rapidly as the number of operators increases, particularly in the float setting where the operators are more diverse: the accuracy of the symbolic model drops from 100% for $o = 1$ to 10% for $o = 10$. We attempted a curriculum learning strategy to alleviate the drop, by giving higher probabilities to expressions with many operators as training advances, but this did not bring any improvement. Increasing the recurrence degree has a similar but more moderate effect: the accuracy decreases from 70% for $d = 0$ (non-recurrent expressions) to 20% for $d = 6$ in the float setup. Finally, we observe that shorter sequences are harder to predict as they give less information on the underlying recurrence; however, even when fed with less than 10 terms, our models achieve surprisingly high accuracies.

[2]Since expressions are not simplified, o may be overestimated.



(**a**) In-domain

(**b**) Out-of-domain

Figure F.2 – Accuracy of our models on various in-domain and out-of-domain groups. We set $\tau = 10^{-10}$, $n_{pred} = 10$.

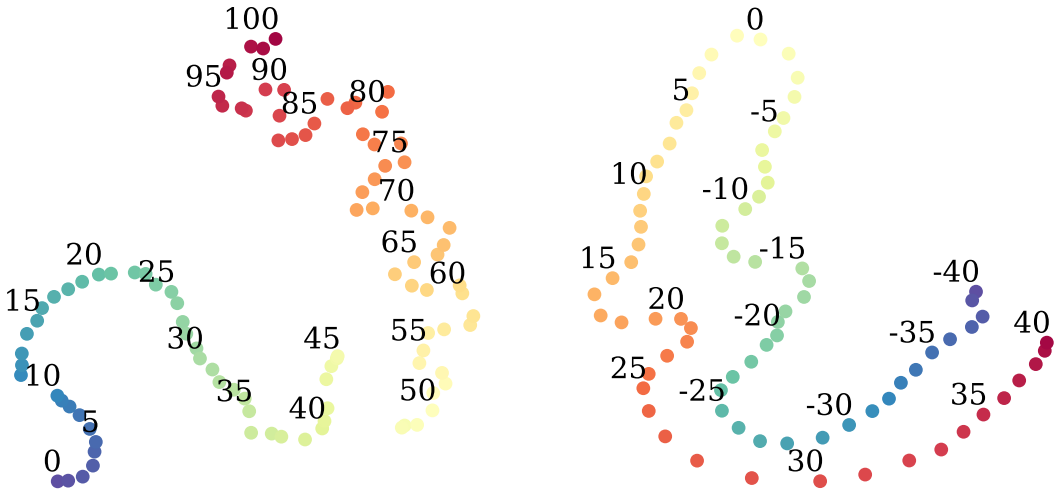
Ablation over operator families To understand what kind of operators are the hardest for our float model, we bunch them into 5 groups:

- **base**: {add, sub, mul}.
- **division**: base + {div, inv}.
- **sqrt**: base + {sqrt}.
- **exponential**: base + {exp, log}.
- **trigonometric**: base + {sin, cos, tan, arcsin, arccos, arctan}.

Results are displayed in Figure F.2a. We see that the main difficulties lie in division and trigonometric operators, but the performance of both models stays rather good in all categories.

Visualizing the embeddings To give more intuition on the inner workings of our symbolic models, we display a t-SNE [MH08a] projection of the embeddings of the integer model in Figure F.3a and of exponent embeddings of the float model in Figure F.3b.

Both reveal a sequential structure, with the embeddings organized in a clear order, as highlighted by the color scheme. In Appendix F.10, we study in detail the pairwise distances between embeddings, unveiling interesting features such as the fact that the integer model naturally learns a base-6 representation.



(**a**) Integer embeddings (**b**) Exponent embeddings

Figure F.3 – The number embeddings reveal intriguing mathematical structure. We represented the t-SNE of the embeddings of the integer model and the exponent embeddings of the float model. We depicted the first 100 integer embeddings (10,000 in the model), and the exponent embeddings -40 to 40 (-100 to 100 in the model).

F.4 Out-of-domain generalization

In this section, we evaluate the ability of our model to generalize out-of-domain. Recurrence prediction being a previously unexplored branch of symbolic regression, there are no official benchmarks we can compare our models to. For integer sequences, we use a subset of OEIS as our out-of-domain benchmark; for float sequences, we use a generator with out-of-vocabulary constants and operators. In Appendix F.5, we also show that our models can and can be made robust noise in the inputs.

F.4.1 Integer sequences: OEIS dataset

The Online Encyclopedia of Integer Sequences (OEIS) is an online database containing over 300,000 integer sequences. It is tempting to directly use OEIS as a testbed for

prediction; however, many sequences in OEIS do not have a closed-form recurrence relation, such as the stops on the New York City Broadway line subway (A000053). These will naturally cause our model to fail.

Preprocessing Luckily, OEIS comes with keywords, and 22% of the sequences are labelled as "easy", meaning that there is a logic to find the next terms (although this logic is by no means easy in most cases). Note that this logic cannot always be formulated as a recurrence relation: for example, the sequence of primes or decimals of π are included in this category, but intractable for our models. We keep the first 10,000 of these sequences as our testbed. Evaluation consists in in showing our models the first $n_{input} \in \{15, 25\}$ terms of each sequence and asking it to predict the $n_{pred} \in \{1, 10\}$ following terms.

Results Results are reported in Table F.7. With only $n_{input} = 15$ terms, the numeric model reaches an impressive accuracy of 53% at next term prediction, and 27% for predicting the next ten terms. The symbolic model achieves lower results, with 33% and 19% respectively; we attribute this to the large number of non-analytic sequences in the testbed. Nonetheless, this means that our model can retrieve a valid recurrence relation for almost a fifth of the sequences, which is rather impressive: we give a few interesting examples in Table F.1. Increasing n_{input} to 25 increases our performances rather marginally.

As a comparison, we ran two built-in Mathematica functions for the task at hand: FindSequenceFunction, which finds non-recurrent expressions, and FindLinearRecurrence, which finds linear recurrence relations. These functions are much more sensitive to the number of terms given as input: they obtain similar accuracies at $n_{input} = 15$, but FindLinearRecurrence performs significantly better at $n_{input} = 25$, while FindSequenceFunction performs pathologically worse. Both these functions perform less well than our symbolic model in all cases.

F.4.2 Float sequences: robustness to out-of-vocabulary tokens

One major difficulty in symbolic mathematics is dealing with out-of-vocabulary constants and operators: the model is forced to approximate them using its own vocabulary. We investigate these two scenarios separately for the float model.

Model	$n_{input} = 15$		$n_{input} = 25$	
	$n_{pred} = 1$	$n_{pred} = 10$	$n_{pred} = 1$	$n_{pred} = 10$
Symbolic (ours)	33.4	19.2	34.5	21.3
Numeric (ours)	53.1	27.4	54.9	29.5
FindSequenceFunction	17.1	12.0	8.1	7.2
FindLinearRecurrence	17.4	14.8	21.2	19.5

Table F.7 – Accuracy of our integer models and Mathematica functions on OEIS sequences. We use as input the first $n_{input} = \{15, 25\}$ first terms of OEIS sequences and ask each model to predict the next $n_{pred} = \{1, 10\}$ terms. We set the tolerance $\tau = 10^{-10}$.

Model	$[\![-10, 10]\!] \cup \{e, \pi, \gamma\}$		$\mathcal{U}(-10, 10)$	
	$n_{op} \leq 5$	$n_{op} \leq 10$	$n_{op} \leq 5$	$n_{op} \leq 10$
Symbolic	**81.9**	**60.7**	60.1	42.1
Numeric	72.4	60.4	**72.2**	**60.2**

Table F.8 – Our symbolic model can approximate out-of-vocabulary prefactors. We report the accuracies achieved when sampling the constants uniformly from $[\![-10, 10]\!] \cup \{e, \pi, \gamma\}$, as during training, versus sampling uniformly in $[-10, 10]$. We set $\tau = 0.01$ (note the higher tolerance threshold as we are considering approximation) and $n_{pred} = 10$.

Out-of-vocabulary constants The first possible source of out-of-vocabulary tokens are prefactors. For example, a formula as simple as $u_n = 0.33n$ is hard to predict perfectly, because our decoder only has access to integers between -10 and 10 and a few mathematical constants, and needs to write 0.33 as `[div,add,mul,3,10,3,mul,10,10]`. To circumvent this issue, Biggio et al. [Big+21] use a separate optimizer to fit the prefactors, once the skeleton of the equation is predicted.

In contrast, our model is end-to-end, and is surprisingly good at approximating out-of-vocabulary prefactors with its own vocabulary. For $u_n = 0.33n$, one could expect the model to predict $u_n = n/3$, which would be a decent approximation. Yet, our model goes much further, and outputs $u_n = -\cos(3)n/3$, which is a better approximation. We give a few other spectacular examples in Table F.2. Our model is remarkably capable of using the values of operators such as $\exp$ and $\arctan$, as if it were able to perform computations internally.

To investigate the approximation capabilities of our model systematically, we evaluate the its performance when sampling the prefactors uniformly in $[-10, 10]$, rather than in $\{-10, -9, ...9, 10\} \cup \{e, \pi, \gamma\}$ as done usually. It is impossible for the symbolic model to perfectly represent the correct formulas, but since we are interested

in its approximation capabilities, we set the tolerance to 0.01. Results are shown in Table F.8. Unsurprisingly, the performance of the numeric model is unaffected as it does not suffer from any out-of-vocabulary issues, and becomes better than the symbolic model. However, the symbolic model maintains very decent performances, with its approximation accuracy dropping from 60% to 42%.

This approximation ability in itself an impressive feat, as the model was not explicitly trained to achieve it. It can potentially have strong applications for mathematics, exploited by the recently proposed Ramanujan Machine [Raa+21]: for example, if a sequence converges to a numerical value such as 1.64493, it can be useful to ask the model to approximate it, yielding $\pi^2/6$. In fact, one could further improve this approximation ability by training the model only on degree-0 sequences with constant leaves ; we leave this for future work.

Out-of-vocabulary functions A similar difficulty arises when dealing with out-of-vocabulary operators, yet again, our model is able to express or approximate them with its own vocabulary. We show this by evaluating our model on various families of functions from `scipy.special`:

- **polynomials**: base + orthogonal polynomials of degree 1 to 5 (Legendre, Chebyshev, Jacobi, Laguerre, Hermite, Gegenbauer)
- **hyperbolic**: base + {sinh, cosh, tanh, arccosh, arcsinh, arctanh}
- **bessel**: base + {Bessel and modified Bessel of first and second kinds}
- **fresnel**: base + {erf, Faddeeva, Dawson and Fresnel integrals}.

The results in Figure F.2b show that both the numeric and symbolic models cope surprisingly well with these functions. The symbolic model has more contrasted results than the numeric model, and excels particularly on functions which it can easily build with its own vocabulary such as polynomials. Surprisingly however, it also outperforms the numeric model on the other groups.

In Table F.3, we show a few examples of the remarkable ability of our model to yield high-quality asymptotic approximations, either using a recurrence relation as for the Catalan numbers and the Dawson function, either with a non-recurrent expression as for the Bessel functions.

F.5 Robustness to noise

One particularity of our model is that it is entirely trained and evaluated on synthetic data which is completely noise-free. Can our model also predict recurrence relations when the inputs are corrupted ? In this section, we show that the answer is yes, provided the model is trained with noisy inputs. For simplicity, we restrict ourselves here to the setup of float sequences, but the setup of integer sequences can be dealt with in a similar manner, trading continuous random variables for discrete ones.

Setup Considering the wide range of values that are observed in recurrent sequences, corruption via additive noise with constant variance, i.e. $u_n = f(n, \{u_i\}_{i<n}) + \xi_n, \xi_n \sim \mathcal{N}(0, \sigma)$ is a poor model of stochasticity. Indeed, the noise will become totally negligible when $u_n \gg 1$, and conversely, totally dominate when $u_n \ll 1$. To circumvent this, we scale the variance of the noise with the magnitude of the sequence, i.e. $\xi_n \sim \mathcal{N}(0, \sigma u_n)$, allowing to define a signal-to-noise ratio $\mathrm{SNR} = 1/\sigma$. This can also be viewed as a multiplicative noise $u_n = f(n, \{u_i\}_{i<n})\xi, \xi \sim \mathcal{N}(1, \sigma)$.

Results To make our models robust to corruption in the sequences, we use stochastic training. This involves picking a maximal noise level σ_{train}, then for each input sequence encountered during training, sample $\sigma \sim \mathcal{U}(0, \sigma_{train})$, and corrupt the terms with a multiplicative noise of variance σ. At test time, we corrupt the input sequences with a noise of fixed variance σ_{test}, but remove the stochasticity for next term prediction, to check whether our model correctly inferred the deterministic part of the formula.

Results are presented in Table F.9. We see that without the stochastic training, the accuracy of our model plummets from 43% to 1% as soon as noise is injected at test time. However, with stochastic training, we are able to keep decent performance even at very strong noise levels: at $\sigma_{test} = 0.5$, we are able to achieve an accuracy of 17%, which is remarkable given that the signal-to-noise ratio is only of two. However, this robustness comes at a cost: performance on the clean dataset is degraded, falling down to 30%.

$\sigma_{train/test}$	$\sigma_{test} = 0$	$\sigma_{test} = 0.1$	$\sigma_{test} = 0.5$
$\sigma_{train} = 0$	**43.3**	0.9	0.0
$\sigma_{train} = 0.1$	38.4	**31.9**	0.2
$\sigma_{train} = 0.5$	35.6	31.8	**11.1**

Table F.9 – Our symbolic model can be made robust to noise in the inputs, with a moderate drop in performance on clean inputs. We report the accuracy on expressions with up to 10 operators, for $n_{pred} = 10$, $\tau = 10^{-10}$, varying the noise level during training σ_{train} and evaluation σ_{test}.

F.6 Robustness to distribution shifts

Note that in all our training examples, the initial terms of the sequences are sampled in the range $[-10, 10]$. This naturally begs the question: can our model handle cases where the first terms lie outside this range?

Results, shown in Tab. F.10, show that our symbolic models displays poor robustness to this distribution shift: their accuracy plummets by more than a factor of two. In comparison, the numeric model is significantly less affected.

Note that an easy fix to this issue would be to train with a larger range for the scale of the initial terms; nonetheless, this confirms the lack of out-of-distribution robustness of numeric-to-symbolic models, previously demonstrated by [Cha21].

Model	Integer		Float	
	$[\![-10, 10]\!]$	$[\![-100, 100]\!]$	$[-10, 10]$	$[-100, 100]$
Symbolic	80.3	30.5	60.7	27.9
Numeric	75.5	67.4	60.4	60.2

Table F.10 – The symbolic models suffer from distribution shifts. We report the in-domain accuracies obtained when sampling the first terms of the sequences uniformly in $[\![-100, 100]\!]$ (integer) and $[-100, 100]$ (float) instead of $[\![-10, 10]\!]$ and $[-10, 10]$ seen during training. We set $\tau = 0.01$ and $n_{pred} = 10$.

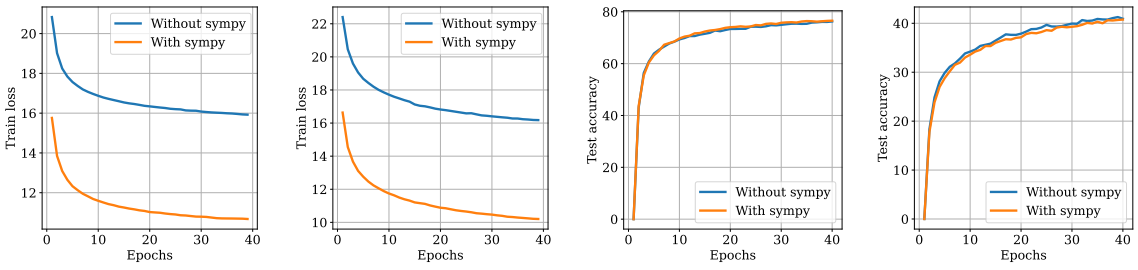
F.7 The effect of expression simplification

One issue with symbolic regression is the fact that a mathematical expression such as `mul, 2, cos, n` can be written in many different ways. Hence, cross-entropy supervision to the tokens of the expression can potentially penalize the model for

generating the same formula written in a different way (e.g. `mul, cos, n, 2` or `mul, cos, n, add, 1, 1`). To circumvent this issue, [Pet+19] first predict the formula, then evaluate it and supervise the evaluations to those of the target function. Yet, since the evaluation step is non-differentiable, they are forced to use a Reinforcement Learning loop to provide reward signals. In our framework, we noticed that such an approach is actually unnecessary.

Instead, one could simply preprocess the mathematical formula to simplify it with SymPy [Meu+17a] before feeding it to the model. This not only simplifies redundant parts such as `add, 1, 1` $\rightarrow$ `2`, but also gets rid of the permutation invariance `mul, x, 2` = `mul, 2, x` by following deterministic rules for the order of the expressions. However, and rather surprisingly, we noticed that this simplification does not bring any benefit to the predictive power of our model : although it lowers the training loss (by getting rid of permutation invariance, it lowers cross-entropy), it does not improve the test accuracy, as shown in Fig. F.4. This suggests that expression syntax is not an issue for our model: the hard part of the problem indeed lies in the mathematics.

Aside from predictive power, SymPy comes with several advantages and drawbacks. On the plus side, it enables the generated expressions to be written in a cleaner way, and improves the diversity of the beam. On the negative side, it slows down training, both because it is slow to parse complex expressions, and because it often lengthens expression since it does not handle division (SymPy rewrites `div,a,b` as `mul,a,pow,b,-1`). Additionally, simplification actually turns out to be detrimental to the out-of-domain generalization of the float model. Indeed, to generate approximations of out-of-vocabulary prefactors, the latter benefits from non-simplified numerical expressions. Hence, we chose to not use SymPy in our experiments.



(**a**) Training loss, integer (**b**) Training loss, float (**c**) Test accuracy, integer (**d**) Test accuracy, float

Figure F.4 – Simplification reduces the training loss, but does not bring any improvement in test accuracy. We displayed the first 40 epochs of training of our symbolic models.

F.8 Does memorization occur?

It is natural to ask the following question: due to the large amount of data seen during training, is our model simply memorizing the training set ? Answering this question involves computing the number of possible inputs sequences N_{seq} which can be generated. To estimate this number, calculating the number of possible mathematical expressions N_{expr} is insufficient, since a given expression can give very different sequences depending on the random sampling of the initial terms. Hence, one can expect that N_{expr} is only a very loose lower bound for N_{seq}.

Nonetheless, we provide the lower bound N_{expr} as a function of the number of nodes in Fig. F.5, using the equations provided in [LC19]. For small expressions (up to four operators), the number of possible expressions is lower or similar to than the number of expressions encountered during training, hence one cannot exclude the possibility that some expressions were seen several times during training, but with different realizations due to the initial conditions. However, for larger expressions, the number of possibilities is much larger, and one can safely assume that the expressions encountered at test time have not been seen during training.

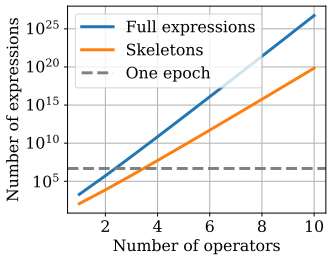


Figure F.5 – Our models only see a small fraction of the possible expressions during training. We report the number of possible expressions for each number of operators (skeleton refers to an expression with the choice of leaves factored out). Even after a hundred epochs, our models have only seen a fraction of the possible expressions with more than 4 operators.

F.9 The issue of finite precision

As explained in the main text, our model tends to ignore subdominant terms in expressions with terms of vastly different magnitudes, partly due to finite precision. For example, when using a float precision of $p = 4$ digits, we obtain a discretization error of 10^5 for numbers of magnitude 10^9. However, there exists two methods to circumvent this issue.

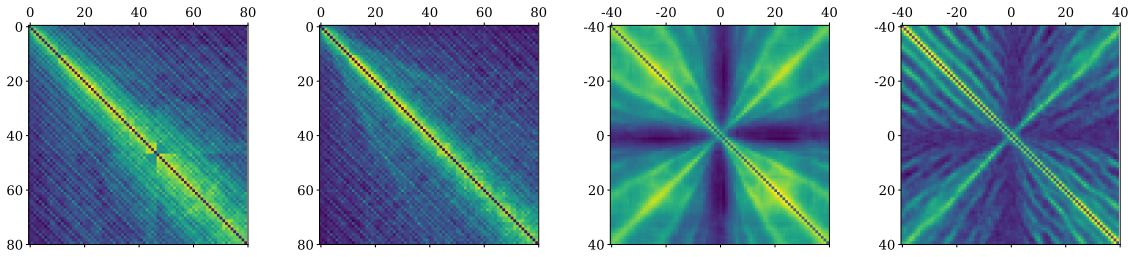
Increasing the precision Naively increase the precision p would cause the vocabulary size of the encoder to rapidly explode, as it scales as 10^p. However, one can instead encode the mantissa on multiple tokens, as performed for integers. For example, using two tokens instead of one, e.g. encoding π as `+ 3141, 5926, E-11`, doubles the precision while increasing the sequence length only by 33%.

This approach works well for recurrence prediction, but slightly hampers the ability of the model to approximate prefactors as shown in Tab. F.2, hence we did not use it in the runs presented in this paper.

Iterative refinement Another method to improve the precision of the model is to use an iterative refinement of the predicted expression, akin to perturbation theory in physics.

Consider, for example, the polynomial $f(x) = \sum_{k=0}^{d} a_k x^k$, for which our model generally predicts $\hat{f}(x) = \sum_{k=0}^{d} \hat{a}_k x^k$, with the first coefficient correct ($\hat{a}_d = a_d$) but potentially the next coefficients incorrect ($\hat{a}_k \neq a_k$ for $k < d$). One can correct these subdominant coefficients iteratively, order by order. To obtain the term a_{d-1}, fit the values of $g(x) = f(x) - \hat{f}(x) = \sum_{k=1}^{d-1}(a_k - \hat{a}_k)$. Then fit the values of $h(x) = g(x) - \hat{g}(x)$, etc. By iterating this procedure k times, one can obtain the k highest coefficients a_k.

We checked that this method allows us to approximate any polynomial function. One could in fact use iterative refinement to predict the Taylor approximation of any function, or use a similar approach to catch multiplicative corrections, by fitting $g(x) = f(x)/\hat{f}(x)$; we leave these investigations for future work.



(**a**) Integer symbolic (**b**) Integer numeric (**c**) Float symbolic (**d**) Float numeric

Figure F.6 – The similarity matrices reveal more details on the structure of the embeddings. The element (i, j) is the cosine similarity between embeddings i and j.

F.10 Structure of the embeddings

To gain better understanding on the number embeddings of our models, we depict similarity matrices whose element (i, j) is the cosine similarity between embeddings i and j in Fig. F.6.

Integer sequences The numeric and symbolic similarity matrices look rather similar, with a bright region appearing around the line $y = x$, reflecting the sequential nature of the embeddings. In both cases, we see diagonal lines appear : these correspond to lines of common divisors between integers. Strikingly, these lines appear most clearly along multiples of 6 and 12, especially in the symbolic model, suggesting that 6 is a natural base for reasoning. These results are reminiscent of the much earlier explorations of Paccanaro and Hinton [PH01].

Float sequences Both for the numeric and symbolic setups, the brightest regions appear along the diagonal lines $y = x$ and $y = -x$, reflecting respectively the sequential nature of the embeddings and their symmetry around 0. The darkest regions appears around the vertical line $x = 0$ and the horizontal line $y = 0$, corresponding to exponents close to zero: these exponents strongly overlap with each other, but weakly overlap with the rest of the exponents. Interestingly, dimmer lines appear in both setups, but follow a very different structure. In the symbolic setup, the lines appear along lines $y = \pm x/k$, reminiscent of the effect of polynomials of degree k. In the numeric setup, the lines are more numerous, all diagonal but offset vertically.

F.11 Visualizations

Success and failure modes In Fig. F.7, we show a few examples of success and failure modes of our symbolic models. The failure modes are particularly interesting, as they reflect the strong behavioral difference between our symbolic models and models usually used for regression tasks.

The latter generally try to interpolate the values of the function they are given, whereas our symbolic model tries to predict the expression of the function. Hence, our model cannot simply "overfit" the inputs. A striking consequence of this is that in case of failure, the predicted expression is wrong both on the input points (green area) and the extrapolation points (blue area).

In some cases, the incorrectly predicted formula provides a decent approximation of the true function (e.g. when the model gets a prefactor wrong). In others, the predicted formula is catastrophically wrong (e.g. when the model makes a mistake on an operator or a leaf).

Training curves In Fig. F.8, we show the training curves of our models, presenting an ablation over the tolerance τ, the number of predictions n_{pred}, the number of operators o, the recurrence degree d and the number of input points l, as explained in the main text.

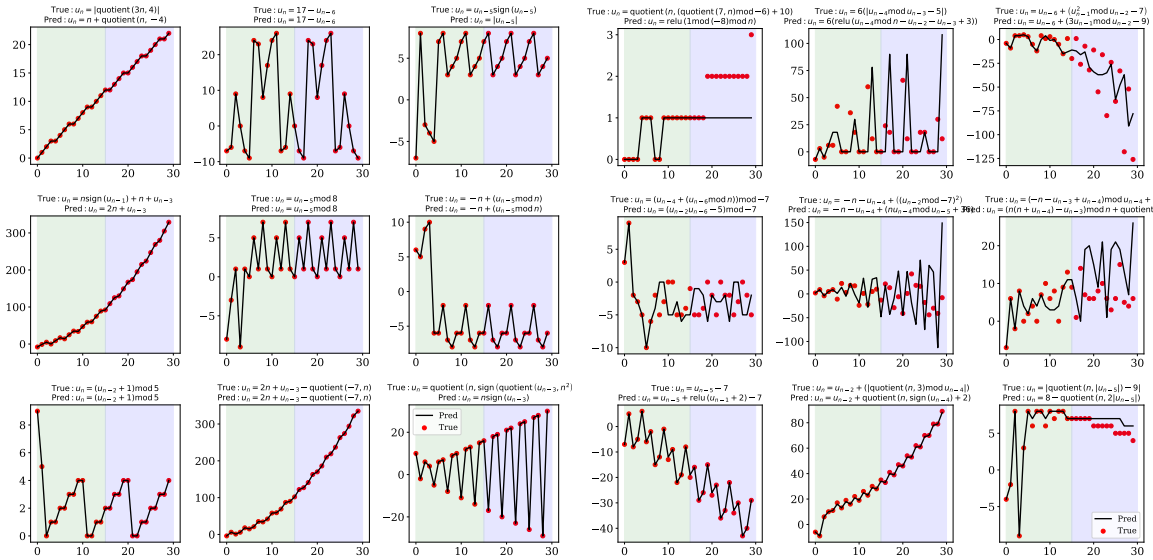
Attention maps In Fig. F.9, we provide attention maps for the 8 attention heads and 4 layers of our Transformer encoders. Clearly, different heads play very different roles, some focusing on local interactions and others on long-range interactions. However, the role of different layers is hard to interpret.

F.12 Conclusion

In this work, we have shown that Transformer models can successfully infer recurrence relations from observations. We applied our model to challenging out-of-distribution tasks, showing that it outperforms Mathematica functions on integer sequences and yields informative approximations of complex functions as well as numerical constants.

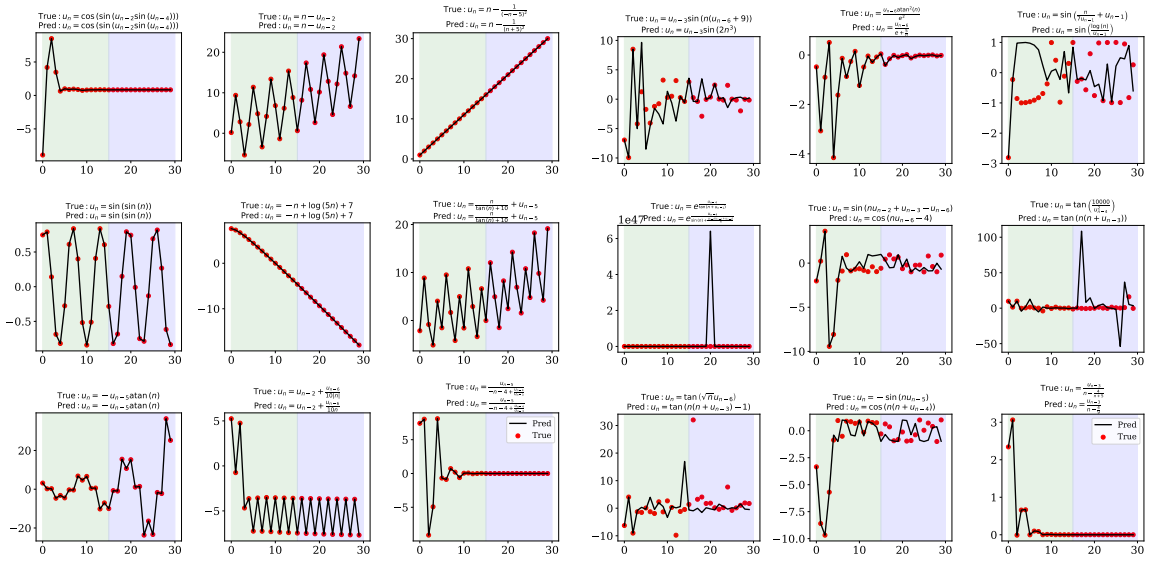
Scope of our approach One may ask to what extent our model can be used for real-world applications, such as time-series forecasting. Although robustness to noise is an encouraging step in this direction, we believe our model is not directly adapted to such applications, for two reasons.

First, real-world time-series often cannot be described by a simple mathematical formula, in which case numeric approaches will generally outperform symbolic approaches. Second, even when they can be expressed by a formula, the latter will contain complex prefactors and non-deterministic terms which will make the task extremely challenging.



(**a**) Integer, success

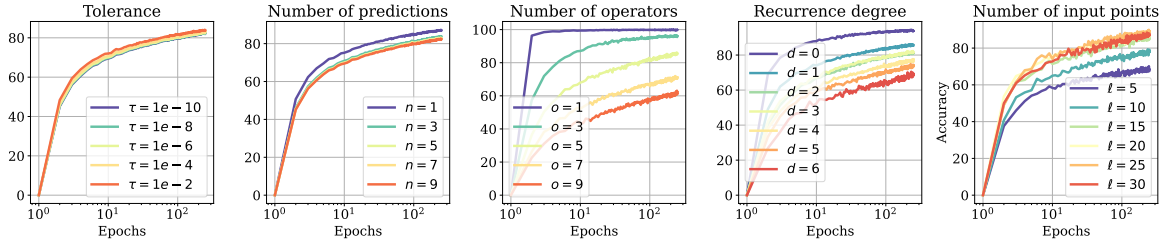
(**b**) Integer, failure



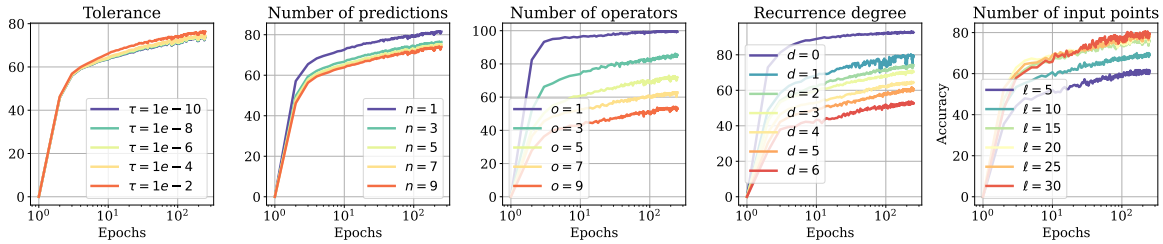
(**c**) Float, success

(**d**) Float, failure

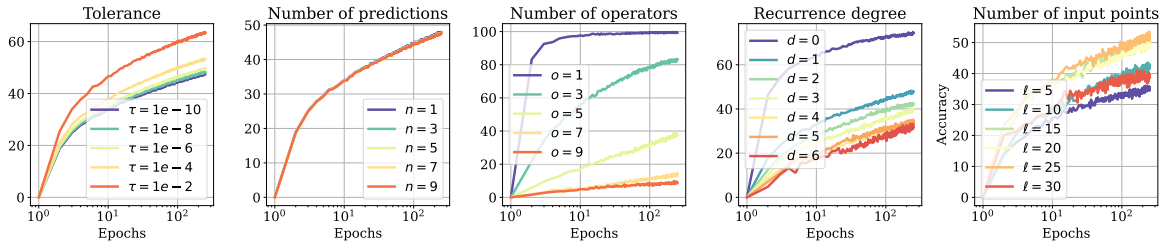
Figure F.7 – Success and failure modes of our models. The models are fed the first 15 terms of the sequence (green area) and predict the next 15 terms (blue area). We randomly selected expressions with 4 operators from our generator, and picked the first successes and failures.



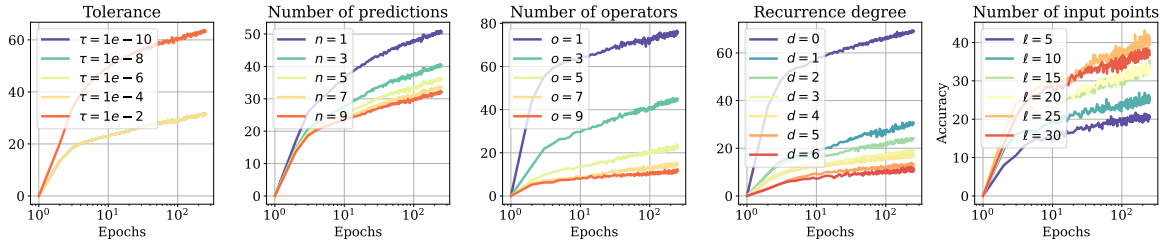
(**a**) Integer symbolic



(**b**) Integer numeric

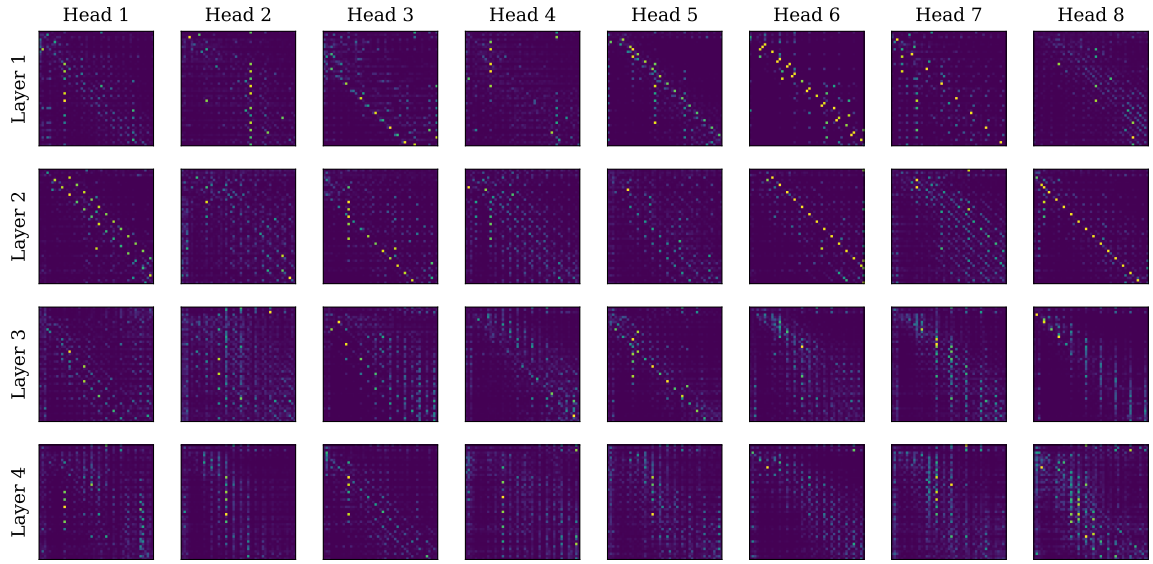


(**c**) Float symbolic

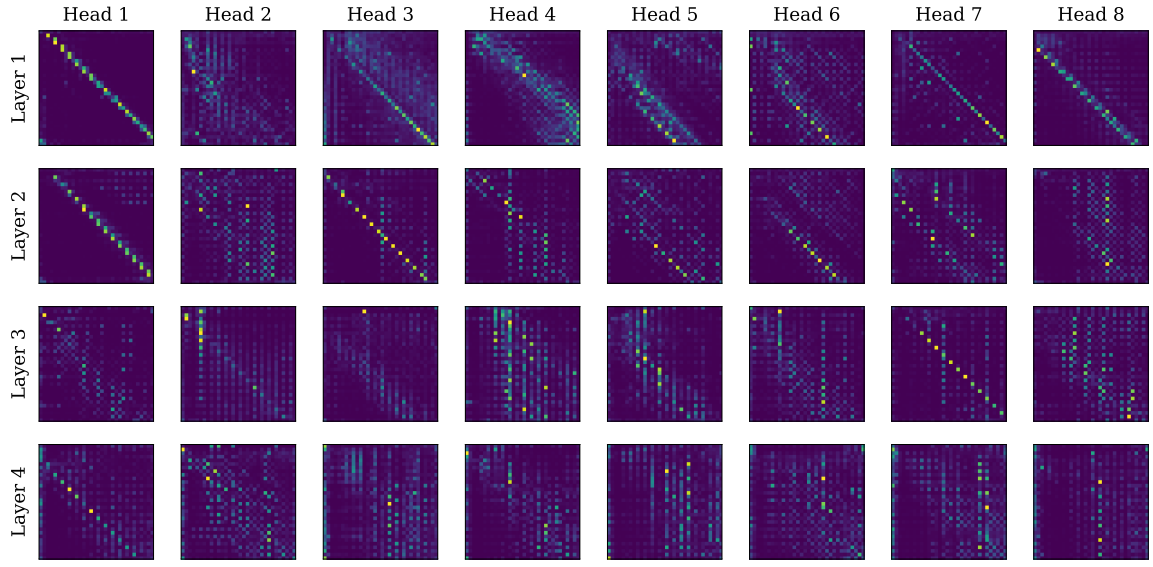


(**d**) Float numeric

Figure F.8 – Training curves of our models. We plot the accuracy of our models at every epoch, evaluated of 10,000 sequences generated from the same distribution as during trianing. From left to right, we vary the tolerance τ, the number of predictions n_{pred}, the number of operators o, the recurrence degree d and the number of input terms l. In each plot, we use the following defaults for quantities which are not varied: $\tau = 10^{-10}$, $n_{pred} = 10$, $o \in [\![1, 10]\!]$, $d \in [\![1, 6]\!]$, $l \in [\![5, 30]\!]$.



(**a**) Integer



(**b**) Float

Figure F.9 – Attention maps of our integer model and float models. We evaluated the integer model on the first 25 terms of the sequence $u_n = -(6 + u_{n-2}) \bmod n$ and the float model on the first 25 terms of the sequence $u_n = \exp(\cos(u_{n-2}))$.

Future directions To bring our model closer to real-world problems, we need our model to be able to handle arbitrary prefactors. We introduce a method to solve this problem in our follow-up work by introducing numeric tokens in the decoder [Kam+22].

Another important extension of our work is the setting of multi-dimensional sequences. With two dimensions, one could study complex sequences, which are a well-studied branch of mathematics given their relationship with fractals.

Finally, recurrence relations being a discrete version of differential equations, the most natural extension to this work is to infer differential equations from trajectories; this will be an important direction for future work.

Appendix G

Leveraging prior knowledge in DGSR

In SR, the goal is to find an analytical expression that accurately fits experimental data with the minimal use of mathematical symbols such as operators, variables, and constants. However, the combinatorial space $\mathcal{E}$ of possible expressions can make it challenging for traditional evolutionary algorithms to find the correct expression in a reasonable amount of time. To address this issue, Neural Symbolic Regression (NSR) algorithms have been developed that can quickly identify patterns in the data and generate analytical expressions. However, these methods, in their current form, lack the capability to incorporate user-defined prior knowledge, which is often required in natural sciences and engineering fields. To overcome this limitation, we propose a novel neural symbolic regression method, named Neural Symbolic Regression with Hypothesis (NSRwH) that enables the explicit incorporation of assumptions about the expected structure of the ground-truth expression into the prediction process. Our experiments demonstrate that the proposed conditioned deep learning model outperforms its unconditioned counterparts in terms of accuracy while also providing control over the predicted expression structure.

G.1 Introduction

Researchers in natural sciences frequently rely on prior knowledge and analogies to comprehend novel systems and predict their behavior. When studying specific physical phenomena, scientists might anticipate particular constants or symmetries to appear in the mathematical laws describing the data. For instance, in astrophysics, the gravitational constant has a significant impact on determining the scale of interactions

between celestial bodies, while in fluid dynamics, the Reynolds number denotes the relative significance of inertial and viscous forces. Thus, it is crucial to prioritize expressions that contain such constants while employing symbolic regression techniques, as they conform better to the physics laws governing the data. Access to a part of the underlying ground-truth system equation is also a common assumption made in the system identification literature where the physical laws are known up to a few parameters [BPK16; KKB20]. In our work, we will refer to the assumptions made by the SR practitioner about the underlying symbolic expression as *hypotheses*. These hypotheses may be incomplete or partially incorrect and can be used in any form to restrict the search space. If a hypothesis is true, we will name it *privileged information*.

Related work and background

Genetic Programming. Up to our knowledge, injection of prior information in GP methods can only be accomplished by filtering during selection, e.g. using properties like function positivity or convexity [Kro+22; Hai+22]. This strategy is inherently greedy and can result in the selection of suboptimal expressions due to early convergence to local minima. Other forms of high-level prior information available to the user, e.g. complexity of the expected expression, can hardly be incorporated into GP algorithms. Recently [Mun+21], a combination of neural networks and genetic programming (GP) has been proposed to improve the performance of symbolic regression. The neural network is used to generate the initial population for a GP algorithm, resulting in a hybrid approach that combines the strengths of both methods. This combination allows for the ability to learn patterns and explore a large solution space, resulting in remarkable performances. However, these systems are not easily controllable, meaning that it can be difficult for the user to constrain the predictions to conform to high-level properties that are known from prior knowledge of the problem.

AI-Feynman. Recent studies [UT20; Udr+20] have investigated the idea of constraining the search to expressions that exhibit particular properties, such as compositionality, additivity, and generalized symmetry. By utilizing these properties, the task of SR becomes significantly less complex as it leverages the modular nature of the resulting expression trees. However, these approaches necessitate fitting a new neural network for every new input dataset and then examining the trained network to identify the desired properties, leading to an inevitably time-consuming process.

DGSR Inspired by recent advances in language models, a line of work named *Neural Symbolic Regression* (NSR), tackles SR as a natural language processing task [Big+20; Big+21; Val+21; dAs+22; Kam+22; Vas+22a; LYS22; Bec+22]. NSR consists of two primary steps: firstly, large synthetic datasets are generated by i) sampling expressions from a prior distribution $p_\theta(\mathcal{E})$ where θ is a parametrization induced by an off-the-shelf expression generator [LC19], ii) evaluating these expressions on a set of points $\mathbf{x} \in \mathbb{R}^d$ where d is the feature dimension, e.g. sampled from a uniform distribution. Secondly, a generative model $g_\phi(\mathcal{E}|\mathcal{D})$, practically a Transformer [Vas+17] parametrized by weights ϕ, that is conditioned on input points $\mathcal{D} = (\mathbf{x}, \mathbf{y})$, is trained on the task of next-token prediction with target the Polish notation of the expression. NSR predicts expressions that share properties of their implicitly biased synthetic generator $p_\theta(\mathcal{E})$. Control over the shape of the predicted expressions, e.g complexity or sub-expression terms, boils down to a sound design of the generator and the pipeline introduced in [LC19] allows only limited degrees of freedom such as operators, variables, constants probability, and tree depth.
Similarly to querying a text-to-image generative model [Ram+22; Sah+22] with a prompt, the SR practitioner might want to restrict the class of predicted expressions to be in a subclass $h(\mathcal{E}) \subset \mathcal{E}$ by using privileged information. Examples of $h(\mathcal{E})$ can be the class of expressions with low complexity, or that include a specific sub-expression like $e^{-\sqrt{x_1^2+x_2^2}}$. However, a trained NSR model $g_\phi(\mathcal{E}|\mathcal{D})$ can only be adapted to $h(\mathcal{E})$ in one of two ways: i) by using rejection sampling, which is time-inefficient and does not guarantee to find candidate expressions with the expected inductive biases, or ii) by designing a new generator with the desired properties and fine-tuning the model on the new dataset, which is a tedious and time-consuming task.

Contributions

In this work, we propose a new method called *Neural Symbolic Regression with Hypotheses* (NSRwH) to address the aforementioned limitations of NSR algorithms. NSRwH efficiently restricts the class of predicted expressions of NSR models *during inference*, if provided privileged information $\mathcal{D}_{\text{PI}}$, with a simple modification to both the model architecture and the training data generation: with the training set of expressions from $p_\theta(\mathcal{E})$, we produce descriptions $\mathcal{D}_{\text{PI}}$, e.g. appearing operators or complexity, and feed this meta-data into the Transformer model as an extra input, i.e. $g_\phi(S|\mathcal{D}, \mathcal{D}_{\text{PI}})$. During training, we use a masking strategy to avoid our model considering sub-classes of

expressions when no privileged information is provided. We show that our model exhibits the following desirable characteristics:

1. In a similar vein to the recent literature on expression derivation and integration [LC19] and mathematical understanding capabilities of Transformers [Cha22], our results demonstrate that Transformer models can succeed in capturing complex, high-level symbolic expression properties, such as complexity and symmetry.

2. The proposed model is able to output expressions that closely align with user-determined privileged information and/or hypotheses on the sought-for expression when it is conditioned on such information. This makes the model effectively *controllable* as its output reflects the user's expectations of specific high-level properties. This stands in contrast to previous work in the NSR and GP literature, where steering symbolic regressors toward specific properties required either retraining from scratch or using inefficient post hoc greedy search routines.

3. The injection of privileged information provides significant improvements in terms of recovery rate. Such an improvement is, as expected, proportional to the amount of conditioning signal provided to the model. This effect is even more apparent in the case where numerical data are corrupted by noise and in the small data regime, where standard NSR approaches witness a more marked performance deterioration.

4. We empirically demonstrate that incorporating conditioning hypotheses not only enhances the controllability of NSRwH but also improves its exploration capabilities, in contrast to standard NSR approaches that rely solely on increasing the beam size. In particular, we show that injecting a large number of hypotheses randomly chosen from a large pool of candidates results in better exploration performance compared to a standard NSR approach operating with a large beam size.

In essence, our approach provides an additional degree of freedom to standard NSR algorithms by allowing the user to quickly steer the prediction of the model in the direction of their prior knowledge at inference time. This is accomplished by leveraging established techniques from language modeling and prompt engineering. The paper is structured as follows: in Section 7.2, we describe our data generation pipeline and the model architecture; in Section G.3 we detail our experimental setup and report our empirical results and in Section G.4 we discuss promising future directions and the current limitations of our approach. Code is available at https://github.com/SymposiumOrganization/ControllableNeuralSymbolicRegression.

G.2 Method

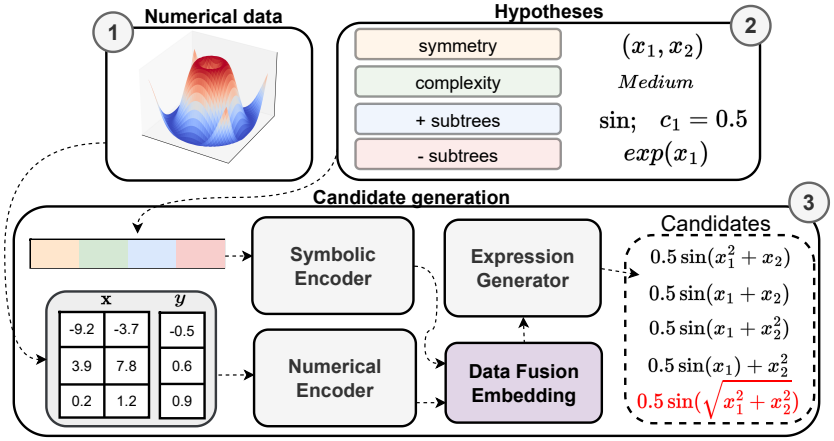


Figure G.1 – Neural Symbolic Regression with Hypotheses. 1) A dataset of numerical observations is obtained; 2) the user formulates a set of hypotheses based on some properties they believe the final expression should possess. After being tokenized independently, the properties tensors are concatenated to form a unique conditioning tensor; 3) numerical data as well as the formulated hypotheses are given as input to two different encoders. Their outputs are then summed and the resulting tensor is processed by a decoder which outputs a set of candidate equations. For NSRwH to be effective and controllable, the candidate expressions should respect the input hypotheses.

G.2.1 Notation and framework

A symbolic regressor is an algorithm that takes as input a dataset $\mathcal{D}$ of n features-value pairs $(\mathbf{x_i}, y_i) \sim \mathbb{R}^d \times \mathbb{R}$, where d is the feature dimension, and returns a symbolic expression $e \sim \mathcal{E}$ such that $\forall (\mathbf{x}_i, y_i) \in \mathcal{D},\ e(\mathbf{x}_i) = \tilde{y_i} \approx y_i$. NSR is a class of SR algorithms that learns a distribution model $g_\phi(\mathcal{E} \mid \mathcal{D})$, parametrized by a neural network with weights ϕ, over symbolic expressions conditioned on an input dataset $\mathcal{D}$. In this work, we introduce NSRwH, a new subclass of neural symbolic regressors, that allows for conditioning their predictions with user-specified prior knowledge about the output expression. More concretely, given a set of privileged information $\mathcal{D}_{\text{PI}}$, NSRwH approaches are trained to model the conditional distribution $g_\phi(\mathcal{E} \mid \mathcal{D}, \mathcal{D}_{\text{PI}})$. An illustration of the proposed approach is shown in Fig. G.1.

G.2.2 Dataset generation

In our framework, a synthetic training sample is defined as a tuple $(e, \mathcal{D}, \mathcal{D}_{\text{PI}})$ where each element is produced as explained below.

Generating e and $\mathcal{D}$. As in other NSR works [Big+21; Kam+22; Val+21], we sample analytical expressions e from $\mathcal{E}$ using the strategy introduced by Lample and Charton [LC19]: random unary-binary trees with depth between 1 and 6 are generated, then internal nodes are assigned either unary or binary operators as described in Table G.2 in Appendix G.5.1 according to their arity, and leaves are assigned variables $\{x_d\}_{d\leq 5}$ and constants. In order to generate $\mathcal{D}$, for each expression e, we sample a support of n points $\mathbf{x}_i \in \mathbb{R}^d$. The values for each coordinate are drawn independently from one another using a uniform distribution $\mathcal{U}$, with the bounds randomly selected from the interval $[-10, 10]$. Next, the expression value y_i is obtained via the evaluation of the expression e on the previously sampled support. More details on the generation of numerical data can be found in Appendix G.5.1

Generating $\mathcal{D}_{\text{PI}}$. Privileged information $\mathcal{D}_{\text{PI}}$ is composed of *hypotheses*. From an expression e, we extract the following properties:

- **Complexity.** We use the definition of complexity provided by [La +21b], i.e. the number of mathematical operators, features, and constants in the output prediction.
- **Symmetry.** We use the definition of generalized symmetry proposed in [Udr+20]: f has generalized symmetry if the d components of the vector $\mathbf{x} \in \mathbb{R}^d$ can be split into groups of k and $d-k$ components (which we denote by the vectors $\mathbf{x}' \in \mathbb{R}^k$ and $\mathbf{x}'' \in \mathbb{R}^{d-k}$) such that $f(\mathbf{x}) = f(\mathbf{x}', \mathbf{x}'') = g[h(\mathbf{x}'), \mathbf{x}'']$ for some unknown function g.
- **Appearing branches.** We consider the set of all the branches that appear in the prefix tree of the generating expression. For instance, for $x_1 + \sin x_2$ this set would be

  $$[+, x_1, +x_1, \sin, \sin(x_2), x_2, +\sin, +\sin(x_2)]$$

  For each expression, in the training set, we sample a subset of this list, ensuring that each element of the subset is sampled with a probability inversely proportional to its length squared and that the full expression tree is never given to the model.

- **Appearing constants.** We also enable the inclusion of a-priori-known constants at test time. We implement this conditioning by drawing inspiration from the concept of pointers in computer programming: we give as input to the model the numerical constant and a pointer, and the model has to place the input pointer in the correct location in the output prediction. This approach does not require representing each constant with a different token, hence preventing the explosion of the output vocabulary size.
- **Absent branches.** We condition our model with the information about subtrees not appearing in true expression. The procedure for extracting this property follows the same logic as the extraction of appearing subtrees.

In the rest of the paper, we refer to these properties as `Complexity`, `Symmetry`, `Positive`, `Constants`, and `Negative`. It is important to note that the set of properties used in this work is not exhaustive and can easily be expanded based on the user's prior knowledge. We provide more details on their exact computation along with a practical example of their extraction in Appendix G.5.2.

G.2.3 Model

Architecture. We use NeSymReS [Big+21] as our base neural symbolic regressor for its simplicity and in the following, we explain how to incorporate the description $\mathcal{D}_{\text{PI}}$ as an input to the model $g_\phi(e|\mathcal{D}, \mathcal{D}_{\text{PI}})$. Note that the very same conditioning strategy can easily be applied to alternative more advanced NSR architectures, such as those introduced in [Val+21; Kam+22]. NSRwH consists of three architectural components: a numerical encoder enc_{num}, a symbolic encoder enc_{sym}, and a decoder dec (see Fig. G.1). Numerical data $\mathcal{D}$, represented by a tensor of size (B, n, D), where B is the batch size, n is the number of points and D is the sum of dependent/independent variables $(D = 5 + 1)$, is converted into a higher dimensional tensor $\mathcal{D}'$ of size (B, n, H) using a multi-hot bit representation according to the half-precision IEEE-754 standard and an embedding layer, where H is the hidden dimension (512 for our experiments). $\mathcal{D}'$ is then processed by a set-transformer encoder [Lee+19a], a variation of [Vas+17] with better inference time and less memory requirement, to produce a new tensor $\mathbf{z}_{num} = enc_{num}(\mathcal{D}')$ of size (B, S, H), where S (50 for our experiments) is the sequence length after the encoder processing. $\mathcal{D}_{\text{PI}}$, represented by a tensor of size (B, M) where M is the number of tokens composing the conditioning hypotheses string, is converted into a higher dimensional tensor $\mathcal{D}'_{PI}$ of size (B, M, H) via an embedding layer. This new tensor is then input into an additional set-transformer to produce a

tensor $\mathbf{z}_{sym} = enc_{sym}(\mathcal{D}'_{PI})$ of size (B, S, H). $\mathbf{z}_{num}$ and $\mathbf{z}_{sym}$ are summed together to produce a new tensor $\mathbf{z}_{fused} = \mathbf{z}_{num} + \mathbf{z}_{sym}$ of size (B, S, H). Finally, $\mathbf{z}_{fused}$ is fed into a standard transformer decoder network, dec, that autoregressively predicts token by token the corresponding expressions using beam search for the best candidates. We resorted to the element-wise summation of $\mathbf{z}_{num}$ and $\mathbf{z}_{sym}$ instead of concatenation in order to reduce memory usage in the decoder, which increases quadratically with the sequence length due to cross attention.

Training and testing. As done in all NSR approaches, we use the cross-entropy loss on next-token prediction using teacher-forcing [SVL14], i.e. conditioning $g_\phi(\tilde{e}_{t+1}|e_{1:t}, \mathcal{D}, \mathcal{D}_{\text{PI}})$ on the first t tokens of the ground-truth e. As for NeSymReS, we "skeletonize" target expressions by replacing constants by a constant token $\diamond$ or, in the case the position of the constant is known a priori, a pointer symbol is used. To prevent our model from being dependent on privileged information at test time, we include training examples with partial privileged information. This means we only provide the model with a subset of all the possible conditionings. For example, only `Positive` and `Symmetry` are given, while `Negative`, `Complexity` and `Constants` are masked out. This is a useful feature of our model as, depending on the use case, some information might not be available and we want the model to still be usable in those cases. At test time, as for NeSymReS, we use beam search to produce a set of predicted expressions, then we apply Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS) [Fle87] to recover the values of the constants by minimizing the squared loss between the original outputs and the output from the predicted expressions. More details on the model and training hyperparameters can be found in Appendix G.5.3.

G.3 Experiments

In this section, we first introduce the datasets and metrics used to evaluate the model and then we present our experiments aimed to assess different properties of NSRwH, including its controllability, and its performance when $\mathcal{D}_{\text{PI}}$ is available, and when it is not. Over the experimental section, we use the standard NeSymReS as a reference baseline, which is referred to as `standard_nesy` in the plots. While our approach could be used with other NSR methods, we have chosen to solely focus on NeSymReS as a baseline model. This allows us to better comprehend the advantages that come from conditioning, instead of assessing various NSR models with distinct numerical input architectures and expression generators. As mentioned in Section 8.1, GP methods

can be hardly conditioned on our set of properties, and as such a comparison with them would be unfair.

G.3.1 Experimental setup

To generate training data, we follow the pipeline introduced in Section G.2.2 resulting in a training set comprising 200 million symbolic expressions with up to 5 variables. The datasets and metrics used to test NSRwH are described below.

Datasets. We use five different databases in our experiments, each characterized by different degrees of complexity: 1) `train_nc`: this dataset comprises 300 symbolic expressions, not including numerical constants. The number of independent variables varies from 1 to 5. The equations are sampled from the same distribution of the training set; 2) `train_wc`: it comprises the same equations of `train_nc` but with numerical constants randomly included in each expression. As such, it represents a more challenging framework than the previous one as the model has the output constant placeholders in the correct positions and BFGS has to find their numerical value; 3) `only_five_variables_nc`: it consists of 300 expressions without constants, strictly selected to have 5 independent variables each. The dataset has been chosen to assess the performance of our algorithm in a higher-dimensional scenario; 4) `AIF`: it comprises all the equations with up to 5 independent variables extracted from the publicly available AIFeynman database [UT20]. It includes equations from the *Feynman Lectures on Physics series* and serves to test the performance of NSRwH on mathematical expressions stemming from several physics domains; 5) `black_box`: it is extracted from the ODE-Strogatz [Str18] databases and serves to evaluate NSRwH in the case where no prior information is available. As also noted by Kamienny et al. [Kam+22], these datasets are particularly challenging as they include non-uniformly distributed points and have different input support bounds than those used by our dataset generation pipeline.

Metrics. We use three different metrics to evaluate our models: 1) `is_satisfied`: this metric measures the percentage of output predictions that agree with a certain property. For all the properties this metric is calculated as follows: given a known equation, we calculate the mean over the total number of times the predictions of the model across the beam size matches the property under consideration. The final metric value is given by the average of the above quantity across all the equations in

the test set; 2) `is_correct`: given a test equation, for each point (x, y) and prediction $\hat{y}$, we calculate `numpy.is_close`$(y, \hat{y})$. Then, we take the mean over all the support points and obtain a real number. If this number is larger than 0.99, we deem our prediction to match the true one and we assign a score of 1, otherwise 0. The final metric value is obtained by calculating the percentage of correctly predicted equations over the entire test set. Importantly, the support points are chosen to be different from those fed into the model at test time; 3) $R^2_{\texttt{mean}}$: given a test equation, and n points $\{x_i, y_i\}_{i=1}^n$, and the corresponding predictions $\{\hat{y}_i\}_{i=1}^n$, we calculate the coefficient of determination, also known as R^2 score, as defined below:

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y}_i)^2} \quad \text{where} \quad \bar{y} = \frac{1}{n}\sum_{i=1}^n y_i$$

The final metric is calculated by taking the mean of the R^2 scores obtained for each equation in the test set. More details on the test datasets and metrics can be found in Appendix G.5.4.

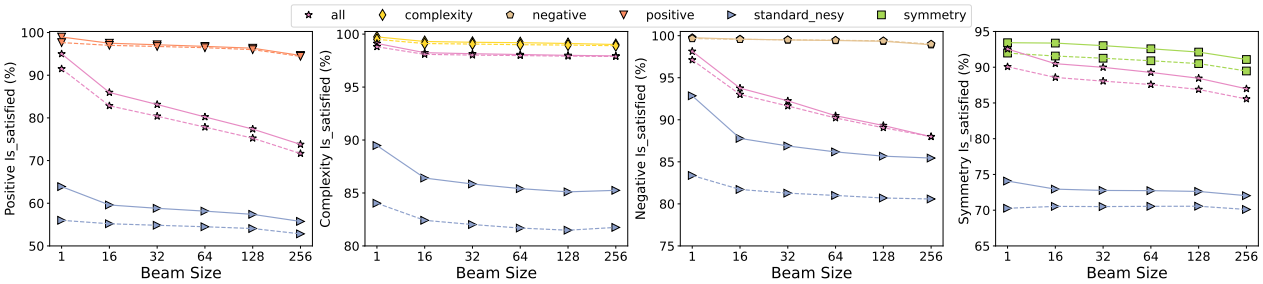


Figure G.2 – Controllability and property matching: The panels show the level of agreement with various types of input conditioning signals – in terms of the `is_satisfied` metric – of our model and the unconditioned baseline (`standard_nesy`), both in the noiseless case (full line) and when noise is injected in the input data (dashed line), as a function of the beam size. The reported results are averaged across all datasets apart from `black_box`.

G.3.2 Can transformers efficiently restrict the inference space using descriptions?

Arguably, the main challenge in symbolic regression is represented by the extremely large search space over mathematical expressions. Methods based on brute force search techniques are doomed to fail or to fall into spurious local minima. The goal of this section is to show that neural symbolic regression algorithms can be *controlled* in such a way that their output adheres with a set of pre-specified inductive biases – meant to narrow the search space – on the nature of the sought expression.

Each panel in Fig. G.2 shows the evolution of the `is_satisfied` metric for various types of conditioning properties as the beam size increases, with and without noise injected in the input data. Noise perturbations are injected in the output of the input data, y, according to the following formula:

$$\tilde{y} = y + \rho\varepsilon \quad \text{where} \quad \varepsilon \sim \mathcal{N}(0, |y|) \;\; and \;\; \rho = 0.01. \tag{G.1}$$

The goal of the experiment is twofold: first, we want to assess whether NSRwH is able to capture the meaning of the input conditing, and second, we want to verify how consistent such an agreement is as we increase the beam size and inject noise. From the results in Fig. G.2, we can observe that the predictions of NSRwH attain a very high `is_satisfied` score for all the evaluated properties. This is in contrast with the unconditioned model which does not consistently capture the underlying properties. This is particularly evident when noise is added to the data, as our model shows robustness to such perturbations, while the standard NSR method experiences greater variations. This is explained by the fact that the standard method grounds its predictions solely on numerical data. As such, when these are severely corrupted, results deteriorate accordingly. We also note that when all possible conditioning properties are given to the model (see `all`), NSRwH tends to underperform with respect to the case when a single property is provided, in particular as the beam size increases. This is likely due to interference effects between different hypotheses, which causes the model, at large beam sizes, to select the subset of them that is more consistent with the numerical data.

G.3.3 Can NSRwH leverage privileged information?

In this section, we investigate whether the ability of NSRwH to capture the meaning of the input properties can be leveraged to improve performance.

To perform these experiments we make use of `is_correct` metric introduced above and we study how performance changes under the effect of noise, number of input data, and amount of conditioning. The beam size for both NSRwH and NeSymReS is set to 5. We start our evaluation from the noiseless case, i.e. no noise is injected in the expressions' evaluation at test time. As such, the mapping between input covariates and the output value is exactly represented by the ground truth symbolic expression. Fig. G.3, shows the performance of NSRwH and the unconditioned model in terms of the `is_correct` metric described above and the different properties provided at test

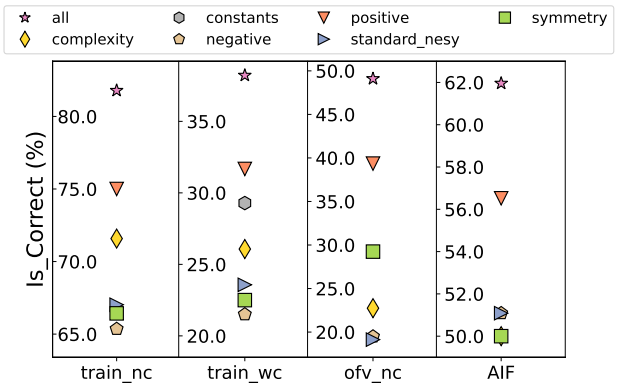


Figure G.3 – Conditioning improves performance. Comparison between NSRwH conditioned with different types of hypotheses and the unconditioned baseline (`standard_nesy`) in terms of the `is_correct` metric. Each column corresponds to a different test dataset.

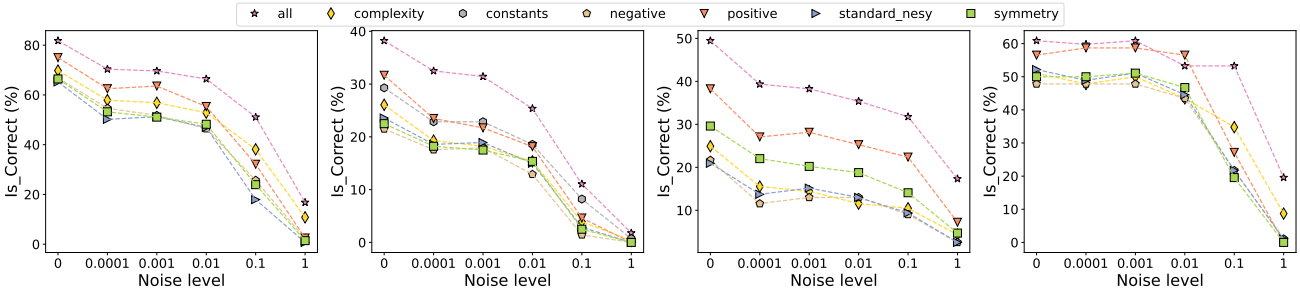


Figure G.4 – Dependence on the input noise. Comparison between NSRwH conditioned with different types of hypotheses and the unconditioned baseline (`standard_nesy`) in terms of the `is_correct` metric, as a function of the noise level in the input data, for the `train_nc`, `train_wc`, `only_five_variables_nc` and `AIF` datasets from left to right.

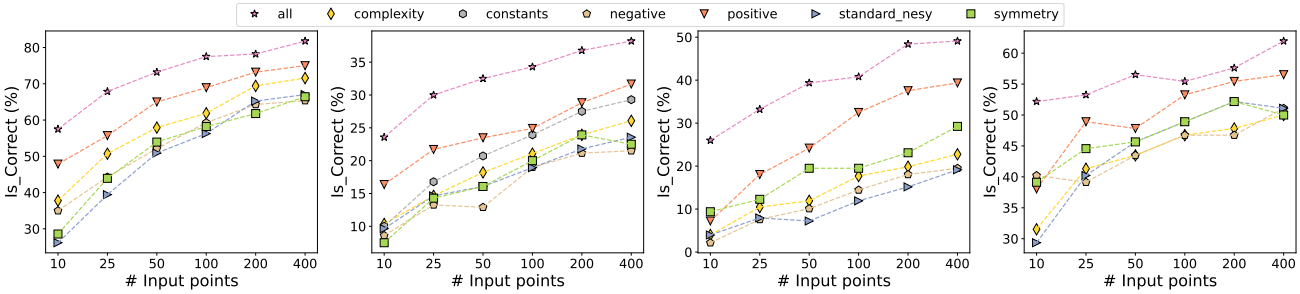


Figure G.5 – Dependence on the number of input points. Comparison between NSRwH conditioned with different types of hypotheses and the unconditioned baseline (`standard_nesy`) in terms of the `is_correct` metric, as a function of the number of input points, for the `train_nc`, `train_wc`, `only_five_variables_nc` and `AIF` datasets from left to right.

time.
Generally, NSRwH efficiently leverages the prompted information to improve its performance. Among the considered individual properties, `Positive` is the most effective one. However, it is interesting to note that `Symmetry` is particularly effective on the `only_five_variables_nc` (ofv_nc) dataset. This is due to the high-dimensional nature of the dataset and the fact the symmetry information is more useful in such cases. Providing information about the ground-truth constants leads to significant performance improvements on the `train_wc` dataset, showcasing the effectiveness of our strategy of providing numerical constants to the model. Finally, `all`, the combination of all the considered properties, is by far the most impacting conditioning. It is noteworthy that, while in some cases the performance of individual properties may not be significantly better than the baseline, their combination (`all`) proves to be highly successful, indicating that the model is able to combine them together effectively.

Case with noise

In this paragraph, we explore the more challenging scenario where noise is injected into the output value y at test time. In particular, we use Eq. G.1 with six different noise levels $\rho \in \{0, 0.0001, 0.001, 0.01, 0.1, 1\}$. The beam size for both NSRwH and NeSymReS is set to 5 in this experiment. As shown in Fig. G.4, the performance improvements are even more pronounced than in the noiseless case shown in Fig. G.3. This illustrates that the incorporation of meaningful inductive biases in our model enables it to effectively manage the impact of noise and, as a result, improves generalization.

Dependence on the number of input points.

In a similar manner as the previous paragraph, this investigation examines whether NSRwH can utilize input conditioning to enhance its performance in the challenging, yet common scenario where small datasets are used as input. As before, the beam size for both NSRwH and NeSymReS is set to 5. As illustrated in Fig. G.5, as the number of input points decreases, the performance of both the conditioned and unconditioned models also declines. However, in NSRwH this effect is significantly reduced, keeping relatively high levels of accuracy even when working in the small data regime.

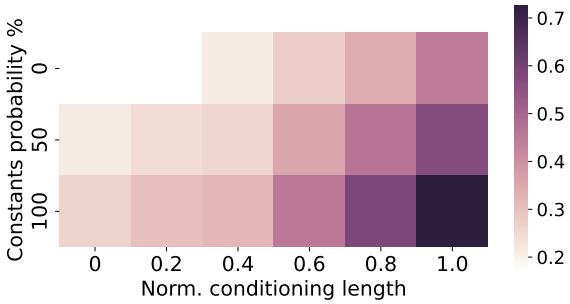


Figure G.6 – Dependence on the amount of conditioning. The heatmap shows how changing the probability of appearing subtrees and constants affects NSRwH's performance on the `train_wc` dataset, measured by the `is_correct` metric. The y-axis shows the probability of constants appearing, with 100% meaning all constants are inputted. The x-axis shows the normalized conditioning length, with 1.0 meaning the model sees positive sub-branches whose length adds up to the prefix ground truth.

Dependence on the amount of conditioning

In this section, we investigate how the performance changes as we increase the amount of conditioning. We conduct this experiment using both `Positive` and `Constants` as we can easily control the degree of conditioning by adjusting the probability of the number of subtrees and constants that appear, respectively. As before, the beam size for both NSRwH and NeSymReS is set to 5. Fig G.6 shows how the value of the `is_correct` metric changes as we vary the amount of `Positive` and `Constants` information. As expected, a monotonic trend can be observed for both properties as the amount of conditioning is increased. The peak in performance is reached when the two properties are provided in the largest amounts, suggesting that the model can combine the two prompts to maximize its prediction accuracy.

G.3.4 What if no assumptions can be made?

This section investigates the scenario where no prior knowledge is available to condition the model. The objective of the experiment is to determine if using NSRwH with randomly sampled hypotheses can outperform a standard NSR model, which can only improve its predictions by increasing the beam size. According to prior work, conventional search techniques of NSR, such as beam search and random sampling, quickly reach a saturation point in exploring the search space, making larger beam sizes ineffective for exploration (see Fig. 16 in [Kam+22]). The experiment is conducted on the `black_box` dataset. The standard model uses a large beam size

Model Type	`is_correct`	$R^2_{\texttt{mean}}$
Random Positive Conditions	**0.35 ± 0.06**	**0.86 ± 0.05**
Standard Model [5K]	0.23 ± 0.00	0.74 ± 0.05
Standard Model [10]	0.12 ± 0.04	0.32 ± 0.03

Table G.1 – No privileged information available. Comparison between NSRwH with randomly sampled hypotheses, a standard NSR approach (NeSymReS) with beam size 5000, and a standard NSR approach with beam size 10 (same as NSRwH). Results are averaged over 5 runs.

of $M = 5000$, which is within the saturation regime, and NSRwH uses $N = 500$ diverse, randomly sampled `Positive` conditionings with a beam size of $M/N = 10$ for each. As such, both methods utilize the same computational budget. Table G.1 shows that NSRwH outperforms the standard NSR model on the `black_box` dataset. We highlight that the policy used to randomly sample positive operators is very sparse and highly suboptimal. As such, the design of more effective search routines over the space of properties represents an interesting avenue for future research.

G.4 Discussion

Conclusive remarks. This work presents a novel approach for symbolic regression that enables the explicit incorporation of inductive biases reflecting prior knowledge of the problem at hand. In contrast to previous works, this can be effectively done at test time, drastically reducing the computational overhead. Thanks to this property, our model better lends itself to online and interactive applications of symbolic regression, thus enabling fast hypothesis testing, a highly desirable feature for scientific discovery applications. We demonstrate the value of this approach with a number of examples and ablation studies where numerical data is scarce or affected by noise.

Limitations and future work The main limitation of the proposed approach is realized in the scenario where no prior knowledge is available. In this case, the performance gains obtained in Section G.3.3 are not guaranteed. However, in Section G.3.4, our final experiment suggests an intriguing opportunity for future research - leveraging NSRwH's extra degree of freedom to explore the equation space more efficiently. In addition, the properties investigated in this work are not exhaustive and it is conceivable to include additional forms of prior knowledge, such as alternative definitions of the complexity of mathematical expressions based on syntax or

semantics [Kom+15a; VSH09]. Finally, we remark that thanks to its simplicity, the same idea at the basis of NSRwH can be applied to more advanced NSR algorithms, like the one recently proposed by [Kam+22], likely resulting in further performance improvements. We intend to investigate the above questions in future work.

G.5 Appendix

G.5.1 Generating $\mathcal{D}$

We build our training dataset first by generating symbolic expressions skeletons (i.e. mathematical expressions where the values of the constants are replaced by placeholder tokens) using the framework introduced by [LC19]. Our vocabulary consists of the unary and binary operators shown in Table G.2. We considered scalar (output dimension equal to 1) expressions with up to 5 independent variables with a maximum prefix length and depth of 20 and 6 respectively.

To obtain the mathematical expression and corresponding numerical evaluation during training for each equation we adopt the following procedure:

- An equation skeleton is randomly sampled from the pool of symbolic expression skeletons
- The sampled expression is simplified using the `simplify` function from `Sympy` [Meu+17b] in order to remove any redundant term.
- Constants of the skeleton are sampled from $\mathcal{U}(-10, -10)$ if they are additive, and logarithmically from $\mathcal{U}(0.05, 10)$ if multiplicative.
- The extrema of the support for each independent variable is sampled independently from a uniform distribution $\mathcal{U}(-10, 10)$ with the distance between the left and right extrema of at least 1.
- For each independent variable, n input points are sampled from the previously sampled support, where n is sampled between $\mathcal{U}(1, 1000)$. Support points that lead to absolute values bigger than 65504 or NANs are discarded and re-sampled.
- We evaluate the sampled expression on the previously obtained support points by using the `lambdify` function from `Sympy` [Meu+17b].

As the input evaluations can lead to large values, we follow [Big+21] and we convert them from float to a multi-hot bit representation according to the half-precision IEEE-754 standard before feeding them into the model.

Arity	Operators

Unary	`sqrt, pow2, pow3, pow4`
	`inv, log, exp`
	`sin, cos, asin`

Binary	`add, sub, mul, div`

Table G.2 – Operators used in our data generation pipeline.

G.5.2 Generating $\mathcal{D}_{\text{PI}}$

Complexity

The complexity of a sentence is determined by the sum of the number of nodes and leaves in the expression, as outlined in [La +21b]. Each complexity value is represented by a unique token, ranging from 1 (i.e. x_1) to 20.

Symmetry

We use the definition of generalized symmetry proposed by [Udr+20]: f has generalized symmetry if the d components of the vector $\mathbf{x} \in \mathbb{R}^d$ can be split into groups of k and $d-k$ components (which we denote by the vectors $\mathbf{x}' \in \mathbb{R}^k$ and $\mathbf{x}'' \in \mathbb{R}^{d-k}$) such that $f(\mathbf{x}) = f(\mathbf{x}', \mathbf{x}'') = g[h(\mathbf{x}'), \mathbf{x}'']$ for some unknown function g. As explained in [Udr+20], in order to check the presence of generalized symmetry in the set of variables $\mathbf{x}'$, it is sufficient to check whether the normalized gradient of f with respect to $\mathbf{x}'$ is independent on $\mathbf{x}''$, i.e. $\frac{\nabla_{\mathbf{x}'} f(\mathbf{x}', \mathbf{x}'')}{|\nabla_{\mathbf{x}'} f(\mathbf{x}', \mathbf{x}'')|}$ is $\mathbf{x}''$-independent. We have created two tokens for each symmetry combination, one to represent the presence of symmetry and one to represent its absence. The total number of tokens is 50, as there are 32 possible symmetry combinations when there are five variables, but some of them are not informative and are excluded, leaving 25 useful combinations. When the number of variables is less than five, only the tokens related to the actual variables are passed to the model (see example in Section G.5.2).

Appearing / absent branches

To sample both appearing and absent branches for an expression, we create two candidate pools: a positive and a negative one. The positive pool is created by using the Depth-First-Search (DFS) algorithm to list all the subtrees within the current prefix

expression and then by removing the subtree corresponding to the entire expression and other non-informative subtrees like x_i. The negative pool is created by filtering out the branches that are already present in the current prefix tree from a pre-computed set of branches, obtained from a large pool of expression trees within the training distribution. We sample subtrees from these pools with a probability proportional to the inverse of their length squared, both during training and evaluation. To regulate the total information given to the model, two parameters, p_p for the positive subtrees and p_n for the negative subtrees, are used. The product of p_p (p_n) and the ground truth length determines the total number of tokens provided to the model, denoted as s_p (s_n). Positive (Negative) subtrees are sampled until the aggregate sum of their lengths, $\sum_{i=1}^{N} \texttt{len}(\text{sampled subtrees}_i)$, reaches the s_p (s_n) value. Sub-branches are separated by special separator tokens.

Constants

Each a-priori-known constant is assigned to a specific symbol, such as `pointer_0` for the first constant, `pointer_1` for the second, and so on. We then give the symbolic encoder the corresponding pointer and a numerical embedding obtained by first converting the known constant in its equivalent 16-bit representation and then passing it through a learnable linear layer that makes its dimension match that of the symbolic embedding. In the target expression, we replace the standard constant placeholder with the `pointer_`i token in the expression. At training and evaluation stages, we regulate the probability of a constant being a-priori-known with a parameter p_c.

Example of extraction and processing of conditioning information

This section provides a concrete example of how different conditionings are extracted and processed to be fed into our model. Consider the expression $x_3 \sin(x_1 + x_2)$.
To determine the `Positive` conditioning, we must first convert it into prefix notation. This is achieved by first rewriting it as ['mul', 'x_3', 'sin', 'add', 'x_1', 'x_2'] and then enumerating all the possible subtrees of the expression, excluding trivial subtrees such as 'x_2' alone. These are: [['add'], ['mul'], ['sin'], ['add', 'x_1'], ['add', 'x_2'], ['mul', 'x_3'], ['add', 'x_1', 'x_2'], ['sin', 'add', 'x_1', 'x_2'], ['mul', 'sin', 'add', 'x_1', 'x_2']]. Positive conditionings are then sampled from this pool with a probability inversely proportional to the length squared of the subtree. So a positive conditioning such as ['mul', 'x_3'] is less likely to be sampled than ['sin'] but more likely than ['mul', 'sin', 'add', 'x_1', 'x_2'].

To obtain the `Negative` conditionings, we generate subtrees at random that are absent from the positive pool. This is achieved by randomly selecting an expression, enumerating the subtrees within it, and then randomly choosing subtrees from the expression that are not present in the positive pool. For example, for the expression above, a negative conditioning could be ['mul', 'x_1'], or ['exp'] since none of these are present in the positive pool. The number of sub-trees supplied to the model is determined by the values of p_p and p_n, and the total length of the expression. For example, if p_p is 0.5, then the total length of the sampled sub-trees will be 3, since the overall length of the ground truth is 6.
`Constant` conditioning would be empty since no constants can be obtained.
`Complexity` conditioning is simply the sum of total nodes and leaves of the prefix expression tree, so in this case, it is equal to 6.
For the `Symmetry` conditioning, we followed the definition given provided by [Udr+20]. For our example expression, we will have symmetry between x_1 and x_2 but not between x_1, x_3 or x_2, x_3.
Once computed, the conditionings are wrapped into a string, tokenized, and then fed into the model. The string will have the following form:

[<Positive>, 'sin', </Positive>, <Positive>, 'mul', 'x_3', </Positive>, <Negative>, 'exp', </Negative>, <Negative>, 'mul', 'x_1', </Negative>, 'Complexity=6', 'TrueSymmetryX1X2', 'FalseSymmetryX2X3', 'FalseSymmetryX1X3'].

If some conditionings should be masked, they are simply excluded from the list; for instance, if we only want to provide symmetry conditioning, the string would have the following form:

['TrueSymmetryX1X2', 'FalseSymmetryX2X3', 'FalseSymmetryX1X3']

G.5.3 Training and testing details

We trained the model with 200 million equations using three NVIDIA GeForce RTX 3090 for a total of five days with a batch size of 400. As in [Big+21], we used a 5-layer set encoder as our numerical encoder and a five-layer standard Transformer decoder as our expression generator. The conditioning and numerical embedding are summed before the expression generator.
In the training process, the Adam optimizer is employed to optimize the cross-entropy loss, utilizing an initial learning rate of 10^{-4}, which is subsequently adjusted in proportion to the inverse square root of the number of steps taken.

To ensure a fair comparison, the standard model, `standard_nesy`, was trained using the same number of equations and the same numerical encoder and expression generator architecture. In addition, both models have been trained for an equal number of iterations.

Amount of conditioning during training

During training, we give the model a varying amount of conditioning signals to avoid excessive dependence on them. We adopt the following approach:

- `Positive`: p_p, as defined in the sub-section G.5.2, is 0 with probability 0.7. Otherwise, it is sampled from $\mathcal{U}(0, 1)$
- `Negative`: p_n, as defined in the sub-section G.5.2 is 0 with probability 0.7. Otherwise, it is sampled from $\mathcal{U}(0, 1)$
- `Complexity`: We provide the complexity token to the network with a probability of 0.3
- `Symmetry`: We provide the symmetry tokens to the network with a probability of 0.2.
- `Constants`: p_c as defined in the sub-section G.5.2 is equal to 0.15.

Amount of conditioning during testing

We use a variety of conditioning signals, with each combination of signals referred to by a specific term.

- `Positive`: p_p as defined in the sub-section G.5.2 is equal to 0.5. The other conditioning signals are disabled.
- `Negative`: p_n as defined in the sub-section G.5.2 is equal to 0.5. The other conditioning signals are disabled.
- `Complexity`: We provide the complexity token to the network. The other conditioning signals are disabled.
- `Symmetry`: We provide the symmetry token to the network. The other conditioning signals are disabled.

- `Constants`: We provide the value of each constant with a probability of 0.8. The other conditioning signals are disabled.
- `Vanilla`: No conditioning is given (all conditionings are masked). The model sees only the numerical inputs. This is equivalent to the standard model.
- `All`: combines `Positive`, `Negative`, `Complexity`, `Symmetry` and `Constants` conditioning. Each conditioning signal is enabled, with parameters equal to the values mentioned for each individual setting with the sole exception of constants where the probability of providing a constant is set to 0.3.

G.5.4 Test datasets and metrics

Evaluation datasets

We created three datasets, `train_nc`, `train_wc` and `only_five_variables_nc` using the same generator configuration as the training set, but with different initial seeds. For `train_nc` and `train_wc` datasets, we selected 300 equations, removing all constants from the first and selecting random constants for the second. These equations have different levels of complexity. In contrast, for `only_five_variables_nc`, we restricted our dataset to equations with five variables, discarding the others. We also removed any constants from these equations. In addition, we evaluate our model on two open-source datasets, namely `AIF`, consisting of the equation in the AI Feynman database [UT20] and `black_box` comprising of 14 datasets from the ODE-Strogatz database [Str18]. For all experiments except G.3.4, the training points were used to both fit constants with BFGS and to select the predicted expression among the beam candidates. Specifically, once the constants were fitted, the expression with the lowest BFGS loss was chosen as the predicted expression. However, since in Section G.3.4 a much larger beam size (5000 compared to 5 of the other experiments) was used, we followed a different approach: 60% of the points were used for fitting constants, and the remaining 40% to select the best expression. The expression with the highest R^2 scores on this 40% support was chosen as the predicted expression.

Evaluation Procedure

For `train_nc`, `train_wc` and `only_five_variables_nc` we sample the support points following the same procedure as in the training pipeline. For AI Feynman equations,

we use the support defined in the dataset. For the ODE-Strogatz dataset we followed the approach from [La +21b] and used 75% of the points from the function call `fetch_data` from the PMLB repository for training [Ols+17] and the remaining for testing.
For the other datasets, to test the quality of our prediction, we sampled 500 points from the OOD support $\mathcal{U}(-25, 25)$. Our criterion for identifying equations as symbolically equivalent to the ground truth was a 99% or higher average output of the `numpy.is_close`$(y, \hat{y})$ function across the support points. This threshold accounted for numerical inaccuracies, such as those caused by numerical instability near support points close to zero, so that equations symbolically equivalent were not misclassified due to these errors.

G.5.5 Additional results

In this section, we report some additional results obtained by evaluating the model on the R^2 metric. We conclude with a subsection comparing the model obtained by completely masking the symbolic encoder of NSRwH (`vanilla` model) and a standard NSR model without any symbolic encoder (`standard_nesy`).

R^2 metric

Figure G.7, Figure G.8 and Figure G.9 repeat the analysis performed in the main body but with the R^2 metric instead of the `is_correct` metric. The scores in this section are calculated by extracting the R^2 value for each expression. If such a value is above 0.99, a score of 1 is assigned, otherwise zero. Finally, the so obtained boolean scores are averaged across the entire test set. We refer to this metric as $R^2_{0.99}$.

Comparison between masked NSRwH and standard model

In this section, we compare the fully masked model – referred to as `vanilla` – to the standard NSR method (without a symbolic encoder) – referred to as `standard_nesy`. The goal is to show that their performance is aligned, indicating that NSRwH represents an enhanced version of standard NSR.

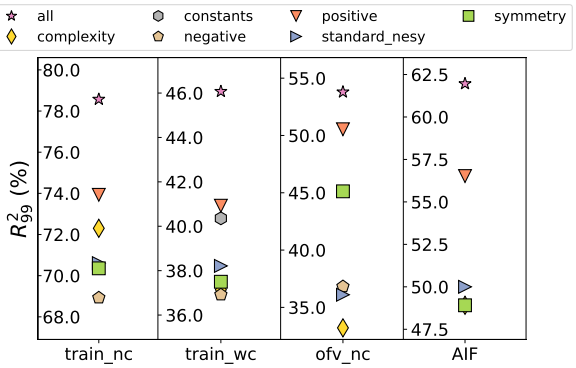


Figure G.7 – Conditioning improves performance. Comparison between NSRwH conditioned with different types of hypotheses and the standard NeSymReS for the different datasets in terms of $R^2_{0.99}$ score

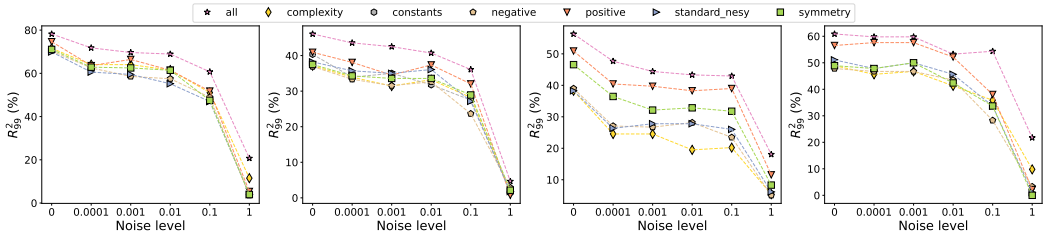


Figure G.8 – Dependence on the input noise. Comparison between NSRwH conditioned with different types of hypotheses and the unconditioned baseline (`standard_nesy`) in terms of the $R^2_{0.99}$ metric, as a function of the noise level in the input data, for the `train_nc`, `train_wc`, `only_five_variables_nc` and `AIF` datasets from left to right.

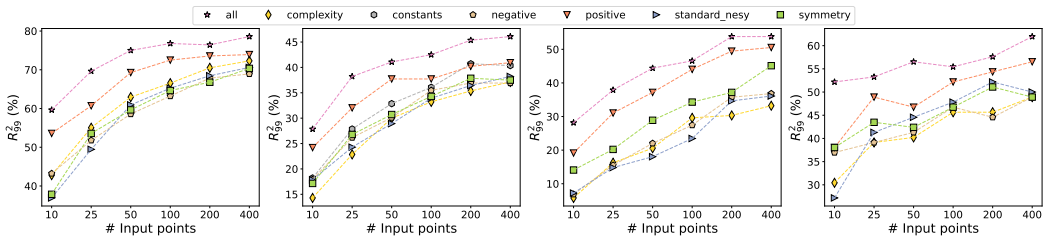


Figure G.9 – Dependence on the number of input points. Comparison between NSRwH conditioned with different types of hypotheses and the unconditioned baseline (`standard_nesy`) in terms of the $R^2_{0.99}$ metric, as a function of the number of input points, for the `train_nc`, `train_wc`, `only_five_variables_nc` and `AIF` datasets from left to right.

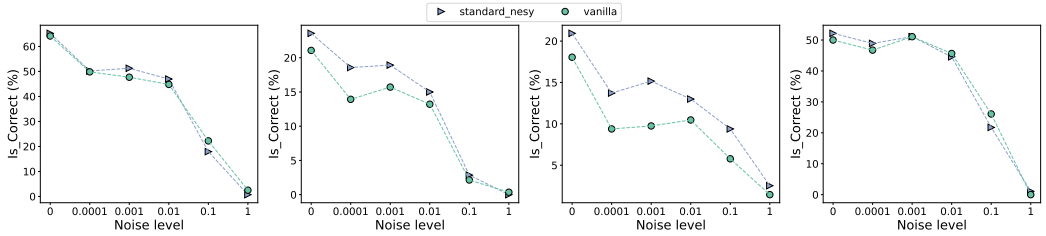


Figure G.10 – Masked NSRwH vs. NeSymReS. Comparison between fully masked NSRwH (`vanilla`) and standard NeSymReS (`standard_nesy`) for different noise levels for the `train_nc`, `train_wc`, `only_five_variables_nc` and `AIF` datasets from left to right.

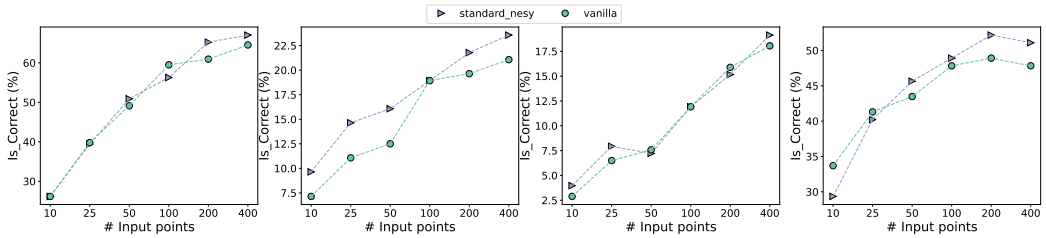


Figure G.11 – Masked NSRwH vs. NeSymReS. Comparison between fully masked NSRwH (`vanilla`) and standard NeSymReS (`standard_nesy`) for a different number of input points for the `train_nc`, `train_wc`, `only_five_variables_nc` and `AIF` datasets from left to right.

List of Figures

List of Algorithms

List of Tables

Bibliography

[Ach18] Joshua Achiam. Spinning Up in Deep Reinforcement Learning (2018).

[Ach+18] Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. Variational option discovery algorithms. *arXiv preprint arXiv:1807.10299* (2018).

[AH18] Hammad Ahmad and Thomas Helmuth. A comparison of semantic-based initialization methods for genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 2018, pp. 1878–1881.

[Ain+23] Joshua Ainslie, Tao Lei, Michiel de Jong, Santiago Ontañón, Siddhartha Brahma, Yury Zemlyanskiy, et al. Colt5: Faster long-range transformers with conditional computation. *arXiv preprint arXiv:2303.09752* (2023).

[Akk+19] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113* (2019).

[All+17] Miltiadis Allamanis, Pankajan Chanthirasegaran, Pushmeet Kohli, and Charles Sutton. Learning continuous semantic representations of symbolic expressions. In *International Conference on Machine Learning*. PMLR. 2017, pp. 80–88.

[ATB17] Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. *Advances in neural information processing systems* 30 (2017).

[ASA18] Forough Arabshahi, Sameer Singh, and Animashree Anandkumar. Towards solving differential equations through neural programming. In *ICML Workshop on Neural Abstract Machines and Program Induction (NAMPI)*. 2018.

[AKO14] Ignacio Arnaldo, Krzysztof Krawiec, and Una-May O'Reilly. Multiple regression genetic programming. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. 2014, pp. 879–886.

[AMH20] Arthur Aubret, Laëtitia Matignon, and Salima Hassas. ELSIM: End-to-end learning of reusable skills through intrinsic motivation. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*. 2020.

[Aue02] Peter Auer. Using confidence bounds for exploitation-exploration tradeoffs. *Journal of Machine Learning Research* 3.Nov (2002), pp. 397–422.

[AB00] Douglas Adriano Augusto and Helio JC Barbosa. Symbolic regression via genetic programming. In *Proceedings. Vol. 1. Sixth Brazilian Symposium on Neural Networks*. IEEE. 2000, pp. 173–178.

[BK20] Akhil Bagaria and George Konidaris. Option discovery using deep skill chaining. In *International Conference on Learning Representations*. 2020.

[BSK21] Akhil Bagaria, Jason K Senthil, and George Konidaris. Skill Discovery for Exploration and Planning using Deep Skill Graphs. In *International Conference on Machine Learning*. PMLR. 2021, pp. 521–531.

[Bak01] Bram Bakker. Reinforcement Learning with Long Short-Term Memory. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*. NIPS'01. Vancouver, British Columbia, Canada: MIT Press, 2001, pp. 1475–1482.

[BA04] David Barber and Felix Agakov. The im algorithm: a variational approach to information maximization. *Advances in neural information processing systems* 16.320 (2004), p. 201.

[BM02] Peter L Bartlett and Shahar Mendelson. Rademacher and Gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research* 3.Nov (2002), pp. 463–482.

[BSA83] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics* 5 (1983), pp. 834–846.

[Bau+21] Kate Baumli, David Warde-Farley, Steven Hansen, and Volodymyr Mnih. Relative Variational Intrinsic Control. In *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 8. 2021, pp. 6732–6740.

[Bec+23] Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, et al. A survey of meta-reinforcement learning. *arXiv preprint arXiv:2301.08028* (2023).

[Bec+22] Sören Becker, Michal Klein, Alexander Neitz, Giambattista Parascandolo, and Niki Kilbertus. Discovering ordinary differential equations that govern time-series. In *NeurIPS 2022 AI for Science: Progress and Promises*. 2022.

[Bel+20] Marc G Bellemare, Salvatore Candido, Pablo Samuel Castro, Jun Gong, Marlos C Machado, Subhodeep Moitra, et al. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature* 588.7836 (2020), pp. 77–82.

[Bel+13] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47 (2013), pp. 253–279.

[BPC20] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150* (2020).

[BBK23] Tommaso Bendinelli, Luca Biggio, and Pierre-Alexandre Kamienny. Controllable Neural Symbolic Regression. *arXiv preprint arXiv:2304.10336* (2023).

[BYM19] Homanga Bharadhwaj, Shoichiro Yamaguchi, and Shin-ichi Maeda. *MANGA: Method Agnostic Neural-policy Generalization and Adaptation*. 2019.

[Big+20] Luca Biggio, Tommaso Bendinelli, Aurelien Lucchi, and Giambattista Parascandolo. A seq2seq approach to symbolic regression. In *Learning Meets Combinatorial Algorithms at NeurIPS2020*. 2020.

[Big+21] Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. *Neural Symbolic Regression that Scales*. 2021.

[Blu+15] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International conference on machine learning*. PMLR. 2015, pp. 1613–1622.

[Boj+16] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).

[Bot+13] Zdravko I Botev, Dirk P Kroese, Reuven Y Rubinstein, and Pierre L'Ecuyer. The cross-entropy method for optimization. In *Handbook of statistics*. Vol. 31. Elsevier, 2013, pp. 35–59.

[Bro+16a] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, et al. *OpenAI Gym*. 2016.

[Bro+16b] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, et al. Openai gym (2016). *arXiv preprint arXiv:1606.01540* 476 (2016).

[Bro+20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, et al. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[Bro+12] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, et al. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games* 4.1 (2012), pp. 1–43.

[BPK16] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences* 113.15 (2016), pp. 3932–3937.

[BL82] Bruno Buchberger and Rüdiger Loos. Algebraic simplification. In *Computer Algebra*. Springer, 1982, pp. 11–43.

[BKK20a] Bogdan Burlacu, Gabriel Kronberger, and Michael Kommenda. Operon C++ an efficient genetic programming framework for symbolic regression. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. 2020, pp. 1562–1570.

[BKK20b] Bogdan Burlacu, Gabriel Kronberger, and Michael Kommenda. Operon C++: An Efficient Genetic Programming Framework for Symbolic Regression. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. GECCO '20. Cancún, Mexico: Association for Computing Machinery, 2020, pp. 1562–1570.

[Cal+16] Roberto Calandra, Jan Peters, Carl Edward Rasmussen, and Marc Peter Deisenroth. Manifold Gaussian processes for regression. In *2016 International joint conference on neural networks (IJCNN)*. IEEE. 2016, pp. 3338–3345.

[Cam+20] Víctor Campos, Alexander Trott, Caiming Xiong, Richard Socher, Xavier Giro-i-Nieto, and Jordi Torres. Explore, Discover and Learn: Unsupervised Discovery of State-Covering Skills. In *International Conference on Machine Learning*. 2020.

[CLS22] Nicolas Castanet, Sylvain Lamprier, and Olivier Sigaud. Stein Variational Goal Generation For Reinforcement Learning in Hard Exploration Problems. *arXiv preprint arXiv:2206.06719* (2022).

[CL11] Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. In *Advances in neural information processing systems*. 2011, pp. 2249–2257.

[Cha21] François Charton. Linear algebra with transformers. *arXiv preprint arXiv:2112.01898* (2021).

[Cha22] François Charton. What is my math transformer doing?–Three results on interpretability and generalization. *arXiv preprint arXiv:2211.00170* (2022).

[CHL20] François Charton, Amaury Hayat, and Guillaume Lample. Learning advanced mathematical computations from examples. *arXiv preprint arXiv:2006.06462* (2020).

[CG16] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, pp. 785–794.

[Che18] Maxime Chevalier-Boisvert. *gym-miniworld environment for OpenAI Gym*. https://github.com/maximecb/gym-miniworld. MiniWorld licensed under Apache License 2.0. 2018.

[CYZ01] Samuel Choi, Dit-Yan Yeung, and Nevin Zhang. Hidden-Mode Markov Decision Processes for Nonstationary Sequential Decision Making. In Jan. 2001, pp. 264–287.

[Chu+18] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems* 31 (2018).

[Cob+21] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168* (2021).

[Cra20] Miles Cranmer. *PySR: Fast & Parallelized Symbolic Regression in Python/Julia*. Sept. 2020.

[Cra23] Miles Cranmer. Interpretable machine learning for science with PySR and SymbolicRegression. jl. *arXiv preprint arXiv:2305.01582* (2023).

[Cra+20] Miles Cranmer, Alvaro Sanchez Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, et al. Discovering symbolic models from deep learning with inductive biases. *Advances in Neural Information Processing Systems* 33 (2020), pp. 17429–17442.

[CBK20] Sebastian Curi, Felix Berkenkamp, and Andreas Krause. Efficient model-based reinforcement learning through optimistic policy search and planning. *Advances in Neural Information Processing Systems* 33 (2020), pp. 14156–14170.

[dAs+22] Stéphane d'Ascoli, Pierre-Alexandre Kamienny, Guillaume Lample, and François Charton. Deep Symbolic Regression for Recurrent Sequences. *arXiv preprint arXiv:2201.04600* (2022).

[Dav+21] A Davies, P Velickovic, L Buesing, S Blackwell, D Zheng, N Tomasev, et al. Advancing mathematics by guiding human intuition with AI. *Nature* (2021).

[DFR13] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE transactions on pattern analysis and machine intelligence* 37.2 (2013), pp. 408–423.

[Den+21] Ludovic Denoyer, Alfredo de la Fuente, Song Duong, Jean-Baptiste Gaya, Pierre-Alexandre Kamienny, and Daniel H Thompson. SaLinA: Sequential Learning of Agents. *arXiv preprint arXiv:2110.07910* (2021).

[Der+19] Erik Derner, Jirí Kubalík, Nicola Ancona, and Robert Babuška. Symbolic regression for constructing analytic models in reinforcement learning. *ArXiv, abs/1903.11483* (2019).

[DOW20] Grant Dick, Caitlin A Owen, and Peter A Whigham. Feature standardisation and coefficient optimisation for effective symbolic regression. In

Proceedings of the 2020 Genetic and Evolutionary Computation Conference. 2020, pp. 306–314.

[Dua+16] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779* (2016).

[ET93] Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Monographs on Statistics and Applied Probability 57. Boca Raton, Florida, USA: Chapman & Hall/CRC, 1993.

[Efr+19] Yonathan Efroni, Nadav Merlis, Mohammad Ghavamzadeh, and Shie Mannor. Tight regret bounds for model-based reinforcement learning with greedy policies. *Advances in Neural Information Processing Systems* 32 (2019).

[Eys+19] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is All You Need: Learning Skills without a Reward Function. In *International Conference on Learning Representations*. 2019.

[Faw+22] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature* 610.7930 (2022), pp. 47–53.

[FAL17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*. PMLR. 2017, pp. 1126–1135.

[Fle87] Roger Fletcher. *Practical Methods of Optimization*. Second. New York, NY, USA: John Wiley & Sons, 1987.

[FDA17] Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1704.03012* (2017).

[Flo07] Razvan V Florian. Correct equations for the dynamics of the cart-pole system. *Center for Cognitive and Neural Studies (Coneural), Romania* (2007).

[FA20] F. O. de Franca and G. S. I. Aldeia. Interaction-Transformation Evolutionary Algorithm for Symbolic Regression. *Evolutionary Computation* (Dec. 2020), pp. 1–25.

[Fra22] Fabricio Olivetti de Franca. Transformation-Interaction-Rational Representation for Symbolic Regression. *arXiv preprint arXiv:2205.06807* (2022).

[FA21] Fabricio Olivetti de França and Guilherme Seidyo Imai Aldeia. Interaction–Transformation Evolutionary Algorithm for Symbolic Regression. *Evolutionary computation* 29.3 (2021), pp. 367–390.

[Fre+21] C Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax–A Differentiable Physics Engine for Large Scale Rigid Body Simulation. *arXiv preprint arXiv:2106.13281* (2021).

[Fri01] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), pp. 1189–1232.

[FHM18] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*. PMLR. 2018, pp. 1587–1596.

[GHK17] Yarin Gal, Jiri Hron, and Alex Kendall. Concrete dropout. *Advances in neural information processing systems* 30 (2017).

[GLK16] Aurélien Garivier, Tor Lattimore, and Emilie Kaufmann. On Explore-Then-Commit strategies. In *NIPS*. 2016, pp. 784–792.

[GRW16] Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control. *arXiv preprint arXiv:1611.07507* (2016).

[Gui+20] Roger Guimerà, Ignasi Reichardt, Antoni Aguilar-Mogas, Francesco A Massucci, Manuel Miranda, Jordi Pallarès, et al. A Bayesian machine scientist to aid in the solution of challenging scientific problems. *Science advances* 6.5 (2020), eaav6971.

[Gup+18] Abhishek Gupta, Russell Mendonca, Yuxuan Liu, Pieter Abbeel, and Sergey Levine. Meta-Reinforcement Learning of Structured Exploration Strategies. In *NeurIPS*. 2018, pp. 5307–5316.

[Haa+18] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*. PMLR. 2018, pp. 1861–1870.

[Hai+22] Christian Haider, Fabrício Olivetti de França, Gabriel Kronberger, and Bogdan Burlacu. Comparing optimistic and pessimistic constraint evaluation in shape-constrained symbolic regression. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 2022, pp. 938–945.

[Han+19] Steven Hansen, Will Dabney, Andre Barreto, David Warde-Farley, Tom Van de Wiele, and Volodymyr Mnih. Fast Task Inference with Variational Intrinsic Successor Features. In *International Conference on Learning Representations*. 2019.

[Har+99] Georges Harik et al. Linkage learning via probabilistic modeling in the ECGA. *IlliGAL report* 99010 (1999).

[Har+20] Kristian Hartikainen, Xinyang Geng, Tuomas Haarnoja, and Sergey Levine. Dynamical Distance Learning for Semi-Supervised and Unsupervised Skill Discovery. In *International Conference on Learning Representations*. 2020.

[Has10] Hado Hasselt. Double Q-learning. *Advances in neural information processing systems* 23 (2010).

[Hau+18] Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin A. Riedmiller. Learning an Embedding Space for Transferable Robot Skills. In *ICLR (Poster)*. OpenReview.net, 2018.

[Hay94] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[Hee+15] Nicolas Heess, Jonathan J. Hunt, Timothy P. Lillicrap, and David Silver. Memory-based control with recurrent neural networks. *CoRR* abs/1512.04455 (2015).

[Hem+12] Erik Hemberg, Kalyan Veeramachaneni, James McDermott, Constantin Berzan, and Una-May O'Reilly. An investigation of local patterns for estimation of distribution genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 2012, pp. 767–774.

[Her+19] Alberto Hernandez, Adarsh Balasubramanian, Fenglin Yuan, Simon AM Mason, and Tim Mueller. Fast, accurate, and transferable many-body interatomic potentials by symbolic regression. *npj Computational Materials* 5.1 (2019), pp. 1–11.

[Hes+17] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, et al. *Rainbow: Combining Improvements in Deep Reinforcement Learning*. 2017.

[Hor21] Richard Zou Horace He. *functorch: JAX-like composable function transforms for PyTorch*. https://github.com/pytorch/functorch. 2021.

[Hos+20] T Hospedales, A Antoniou, P Micaelli, and A Storkey. Meta-learning in neural networks: a survey. arXiv preprint arXiv: 200405439 (2020).

[Hou+16] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. VIME: variational information maximizing exploration. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*. 2016, pp. 1117–1125.

[HVP21] Mike Huisman, Jan N Van Rijn, and Aske Plaat. A survey of deep meta-learning. *Artificial Intelligence Review* 54.6 (2021), pp. 4483–4541.

[Hum+19] Jan Humplik, Alexandre Galashov, Leonard Hasenclever, Pedro A Ortega, Yee Whye Teh, and Nicolas Heess. Meta reinforcement learning as task inference. *arXiv preprint arXiv:1905.06424* (2019).

[JRG23] Minqi Jiang, Tim Rocktäschel, and Edward Grefenstette. General intelligence requires rethinking exploration. *Royal Society Open Science* 10.6 (2023), p. 230539.

[Jin+20a] Ying Jin, Weilin Fu, Jian Kang, Jiadong Guo, and Jian Guo. *Bayesian Symbolic Regression*. 2020.

[Jin+19] Yuu Jinnai, Jee Won Park, David Abel, and George Konidaris. Discovering options for exploration by minimizing cover time. In *International Conference on Machine Learning*. PMLR. 2019, pp. 3130–3139.

[Jin+20b] Yuu Jinnai, Jee Won Park, Marlos C Machado, and George Konidaris. Exploration in reinforcement learning with deep covering options. In *International Conference on Learning Representations*. 2020.

[Kab+21] Evgeniya Kabliman, Ana Helena Kolody, Johannes Kronsteiner, Michael Kommenda, and Gabriel Kronberger. Application of symbolic regression for constitutive modeling of plastic deformation. *Applications in Engineering Science* 6 (2021), p. 100052.

[KLC98] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and Acting in Partially Observable Stochastic Domains. *Artif. Intell.* 101.1-2 (1998), pp. 99–134.

[KKB20] Kadierdan Kaheman, J Nathan Kutz, and Steven L Brunton. SINDy-PI: a robust algorithm for parallel implicit sparse identification of nonlinear dynamics. *Proceedings of the Royal Society A* 476.2242 (2020), p. 20200279.

[KS15] Łukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228* (2015).

[Kam+22] Pierre-Alexandre Kamienny, Stéphane d'Ascoli, Guillaume Lample, and François Charton. End-to-end symbolic regression with transformers. *arXiv preprint arXiv:2204.10532* (2022).

[Kam+23] Pierre-Alexandre Kamienny, Guillaume Lample, Sylvain Lamprier, and Marco Virgolin. Deep Generative Symbolic Regression with Monte-Carlo-Tree-Search. *arXiv preprint arXiv:2302.11223* (2023).

[Kam+20] Pierre-Alexandre Kamienny, Matteo Pirotta, Alessandro Lazaric, Thibault Lavril, Nicolas Usunier, and Ludovic Denoyer. Learning adaptive exploration strategies in dynamic environments through informed policy regularization. *arXiv preprint arXiv:2005.02934* (2020).

[Kam+21] Pierre-Alexandre Kamienny, Jean Tarbouriech, Sylvain Lamprier, Alessandro Lazaric, and Ludovic Denoyer. Direct then diffuse: Incremental unsupervised skill discovery for state covering and goal reaching. *arXiv preprint arXiv:2110.14457* (2021).

[Kid21] Patrick Kidger. *SympyTorch*. https://github.com/patrick-kidger/sympytorch. 2021.

[Kim+14] Kangil Kim, Yin Shan, Xuan Hoai Nguyen, and Robert I McKay. Probabilistic model building in genetic programming: A critical review. *Genetic Programming and Evolvable Machines* 15.2 (2014), pp. 115–167.

[Kim+20] Samuel Kim, Peter Y Lu, Srijon Mukherjee, Michael Gilbert, Li Jing, Vladimir Čeperić, et al. Integration of neural network-based symbolic regression in deep learning for scientific discovery. *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[Koh+21] Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, et al. Wilds: A benchmark of in-the-wild distribution shifts. In *International Conference on Machine Learning*. PMLR. 2021, pp. 5637–5664.

[Kom+15a] Michael Kommenda, Andreas Beham, Michael Affenzeller, and Gabriel Kronberger. Complexity Measures for Multi-objective Symbolic Regression. In *Computer Aided Systems Theory – EUROCAST 2015*. Springer International Publishing, 2015, pp. 409–416.

[Kom+15b] Michael Kommenda, Andreas Beham, Michael Affenzeller, and Gabriel Kronberger. Complexity measures for multi-objective symbolic regression. In *Computer Aided Systems Theory–EUROCAST 2015: 15th International Conference, Las Palmas de Gran Canaria, Spain, February 8-13, 2015, Revised Selected Papers 15*. Springer. 2015, pp. 409–416.

[Kom+20] Michael Kommenda, Bogdan Burlacu, Gabriel Kronberger, and Michael Affenzeller. Parameter identification for symbolic regression using nonlinear least squares. *Genetic Programming and Evolvable Machines* 21.3 (2020), pp. 471–501.

[Koz94] John R Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing* 4.2 (1994), pp. 87–112.

[Kro+22] Gabriel Kronberger, Fabricio Olivetti de França, Bogdan Burlacu, Christian Haider, and Michael Kommenda. Shape-Constrained Symbolic Regression—Improving Extrapolation with Prior Knowledge. *Evolutionary Computation* 30.1 (2022), pp. 75–98.

[Kub+21] Jiří Kubalík, Erik Derner, Jan Žegklitz, and Robert Babuška. Symbolic regression methods for reinforcement learning. *IEEE Access* 9 (2021), pp. 139697–139711.

[Kur+18] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592* (2018).

[La +19] William La Cava, Thomas Helmuth, Lee Spector, and Jason H Moore. A probabilistic and multi-objective analysis of lexicase selection and ε-lexicase selection. *Evolutionary Computation* 27.3 (2019), pp. 377–402.

[La +21a] William La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabricio Olivetti de Franca, Marco Virgolin, Ying Jin, et al. Contemporary symbolic regression methods and their relative performance. *arXiv preprint arXiv:2107.14351* (2021).

[La +21b] William La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabrício Olivetti de França, Marco Virgolin, Ying Jin, et al. *Contemporary Symbolic Regression Methods and their Relative Performance*. 2021.

[La +18] William La Cava, Tilak Raj Singh, James Taggart, Srinivas Suri, and Jason H Moore. Learning concise representations for regression by evolving networks of trees. *arXiv preprint arXiv:1807.00981* (2018).

[LC19] Guillaume Lample and François Charton. Deep learning for symbolic mathematics. *arXiv preprint arXiv:1912.01412* (2019).

[Lam+22] Guillaume Lample, Marie-Anne Lachaux, Thibaut Lavril, Xavier Martinet, Amaury Hayat, Gabriel Ebner, et al. HyperTree Proof Search for Neural Theorem Proving. *arXiv preprint arXiv:2205.11491* (2022).

[Lan+22] Mikel Landajuela, Chak Lee, Jiachen Yang, Ruben Glatt, Claudio P Santiago, Ignacio Aravena, et al. A Unified Framework for Deep Symbolic Regression. In *Advances in Neural Information Processing Systems*. 2022.

[Lan+21] Mikel Landajuela, Brenden K Petersen, Sookyung Kim, Claudio P Santiago, Ruben Glatt, Nathan Mundhenk, et al. Discovering symbolic policies with deep reinforcement learning. In *International Conference on Machine Learning*. PMLR. 2021, pp. 5979–5989.

[Laz12] Alessandro Lazaric. Transfer in reinforcement learning: a framework and a survey. In *Reinforcement Learning*. Springer, 2012, pp. 143–173.

[Laz+18] Nevena Lazic, Craig Boutilier, Tyler Lu, Eehern Wong, Binz Roy, MK Ryu, et al. Data center cooling using model-predictive control. *Advances in Neural Information Processing Systems* 31 (2018).

[Lee+19a] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*. PMLR. 2019, pp. 3744–3753.

[Lee+19b] Lisa Lee, Benjamin Eysenbach, Emilio Parisotto, Eric Xing, Sergey Levine, and Ruslan Salakhutdinov. Efficient exploration via state marginal matching. *arXiv preprint arXiv:1906.05274* (2019).

[LA14] Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. *Advances in neural information processing systems* 27 (2014).

[LYS22] Jiachen Li, Ye Yuan, and Hongze Shen. Symbolic Expression Transformer: A Computer Vision Approach for Symbolic Regression. *ArXiv* abs/2205.11798 (2022).

[Li+19] Li Li, Minjie Fan, Rishabh Singh, and Patrick Riley. Neural-guided symbolic regression with asymptotic constraints. *arXiv preprint arXiv:1901.07714* (2019).

[Lil+15] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, et al. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[Liu+21] Evan Z Liu, Aditi Raghunathan, Percy Liang, and Chelsea Finn. Decoupling exploration and exploitation for meta-reinforcement learning without sacrifices. In *International conference on machine learning*. PMLR. 2021, pp. 6925–6935.

[Liu+20] Evan Zheran Liu, Aditi Raghunathan, Percy Liang, and Chelsea Finn. *Explore then Execute: Adapting without Rewards via Factorized Meta-Reinforcement Learning*. 2020.

[Loo07] Moshe Looks. On the behavioral diversity of random programs. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. 2007, pp. 1636–1642.

[Lu+20] Kevin Lu, Aditya Grover, Pieter Abbeel, and Igor Mordatch. Reset-Free Lifelong Learning with Skill-Space Planning. *arXiv preprint arXiv:2012.03548* (2020).

[Lu+21] Qiang Lu, Fan Tao, Shuo Zhou, and Zhiguang Wang. Incorporating Actor-Critic in Monte Carlo tree search for symbolic regression. *Neural Computing and Applications* 33.14 (2021), pp. 8495–8511.

[Ma+22] He Ma, Arunachalam Narayanaswamy, Patrick Riley, and Li Li. Evolving symbolic density functionals. *arXiv preprint arXiv:2203.02540* (2022).

[MH08a] Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605.

[MH08b] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605.

[MBB17] Marlos C Machado, Marc G Bellemare, and Michael Bowling. A laplacian framework for option discovery in reinforcement learning. In *International Conference on Machine Learning*. PMLR. 2017, pp. 2295–2304.

[Mac+18] Marlos C Machado, Clemens Rosenbaum, Xiaoxiao Guo, Miao Liu, Gerald Tesauro, and Murray Campbell. Eigenoption Discovery through the Deep Successor Representation. In *International Conference on Learning Representations*. 2018.

[McC11] Trent McConaghy. FFX: Fast, scalable, deterministic symbolic regression technology. In *Genetic Programming Theory and Practice IX*. Springer, 2011, pp. 235–260.

[MWB95] Ben McKay, Mark J Willis, and Geoffrey W Barton. Using a tree structured genetic algorithm to perform symbolic regression. In *First international conference on genetic algorithms in engineering systems: innovations and applications*. IET. 1995, pp. 487–492.

[Meh+19] Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J. Pal, and Liam Paull. *Active Domain Randomization*. 2019.

[Meu+17a] Aaron Meurer, Christopher P Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B Kirpichev, Matthew Rocklin, et al. SymPy: symbolic computing in Python. *PeerJ Computer Science* 3 (2017), e103.

[Meu+17b] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, et al. SymPy: symbolic computing in Python. *PeerJ Computer Science* 3 (Jan. 2017), e103.

[Mni+16] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, et al. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. PMLR. 2016, pp. 1928–1937.

[Mni+13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, et al. *Playing Atari with Deep Reinforcement Learning*. 2013.

[Mod+20] Nirbhay Modhe, Prithvijit Chattopadhyay, Mohit Sharma, Abhishek Das, Devi Parikh, Dhruv Batra, et al. IR-VIC: Unsupervised Discovery of Sub-goals for Transfer in RL. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. 2020.

[MBJ20] Thomas M Moerland, Joost Broekens, and Catholijn M Jonker. Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712* (2020).

[MR15] Shakir Mohamed and Danilo Jimenez Rezende. Variational information maximisation for intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*. 2015, pp. 2125–2133.

[Mun+21] T Nathan Mundhenk, Mikel Landajuela, Ruben Glatt, Claudio P Santiago, Daniel M Faissol, and Brenden K Petersen. Symbolic regression via neural-guided genetic programming population seeding. *arXiv preprint arXiv:2111.00053* (2021).

[Mur+14] A Murari, E Peluso, M Gelfusa, I Lupelli, M Lungaroni, and P Gaudio. Symbolic regression via genetic programming for data driven derivation of confinement scaling laws without any assumption on their mathematical form. *Plasma Physics and Controlled Fusion* 57.1 (2014), p. 014008.

[MPR21] Mirco Mutti, Lorenzo Pratissoli, and Marcello Restelli. Task-Agnostic Exploration via Policy Gradient of a Non-Parametric State Entropy Estimate. In *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 10. 2021, pp. 9028–9036.

[Nag+18] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S. Fearing, Pieter Abbeel, Sergey Levine, et al. Learning to Adapt in Dynamic, Real-World Environments Through Meta-Reinforcement Learning. *arXiv e-prints*, arXiv:1803.11347 (Mar. 2018), arXiv:1803.11347.

[Nai+18] Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. *Advances in neural information processing systems* 31 (2018).

[NKJ18] Hyoungwook Nam, Segwang Kim, and Kyomin Jung. *Number Sequence Prediction Problems for Evaluating Computational Powers of Neural Networks*. 2018.

[Ols+17] Randal S. Olson, William La Cava, Patryk Orzechowski, Ryan J. Urbanowicz, and Jason H. Moore. PMLB: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining* 10.1 (Dec. 2017), p. 36.

[Ope+18] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, et al. *Learning Dexterous In-Hand Manipulation*. 2018.

[Ope+21] OpenAI OpenAI, Matthias Plappert, Raul Sampedro, Tao Xu, Ilge Akkaya, Vineet Kosaraju, et al. Asymmetric self-play for automatic goal discovery in robotic manipulation. *arXiv preprint arXiv:2101.04882* (2021).

[Osb16] Ian Osband. Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout. In *NIPS workshop on bayesian deep learning*. Vol. 192. 2016.

[OR17] Ian Osband and Benjamin Van Roy. Why is Posterior Sampling Better than Optimism for Reinforcement Learning? In *ICML*. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 2701–2710.

[PH01] Alberto Paccanaro and Geoffrey E. Hinton. Learning distributed representations of concepts using linear relational embedding. *IEEE Transactions on Knowledge and Data Engineering* 13.2 (2001), pp. 232–244.

[Par+22] Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, et al. Evolving Curricula with Regret-Based Environment Design. *arXiv preprint arXiv:2203.01302* (2022).

[Pat+17] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2017, pp. 16–17.

[PK16] Tomasz P Pawlak and Krzysztof Krawiec. Semantic geometric initialization. In *Genetic Programming: 19th European Conference, EuroGP 2016, Porto, Portugal, March 30-April 1, 2016, Proceedings 19*. Springer. 2016, pp. 261–277.

[Pen+18] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 3803–3810.

[Pet+19] Brenden K Petersen, Mikel Landajuela Larma, T Nathan Mundhenk, Claudio P Santiago, Soo K Kim, and Joanne T Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *arXiv preprint arXiv:1912.04871* (2019).

[Pet+21] Jacob F Pettit, Brenden K Petersen, FL Silva, Dale B Larie, RC Cockrell, Gary An, et al. *Learning sparse symbolic policies for sepsis treatment*. Tech. rep. Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2021.

[Pin+21] Luis Pineda, Brandon Amos, Amy Zhang, Nathan O. Lambert, and Roberto Calandra. MBRL-Lib: A Modular Library for Model-based Reinforcement Learning. *Arxiv* (2021).

[PLM08] Ricardo Poli, William B Langdon, and Nicholas F McPhee. *A Field Guide to Genetic Programming*. 2008.

[PS20] Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393* (2020).

[Pon+20] Vitchyr H Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. In *International Conference on Machine Learning*. 2020.

[Raa+21] Gal Raayoni, Shahar Gottlieb, Yahel Manor, George Pisha, Yoav Harris, Uri Mendlovic, et al. Generating conjectures on fundamental constants with the Ramanujan Machine. *Nature* 590.7844 (2021), pp. 67–73.

[Rag+19] Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. Rapid Learning or Feature Reuse? Towards Understanding the Effectiveness of MAML. *arXiv e-prints*, arXiv:1909.09157 (Sept. 2019), arXiv:1909.09157.

[RK11] Marco Ragni and Andreas Klein. Predicting Numbers: An AI Approach to Solving Number Series. In *KI 2011: Advances in Artificial Intelligence*. Ed. by Joscha Bach and Stefan Edelkamp. Springer Berlin Heidelberg, 2011, pp. 255–259.

[Rak+19] Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables. In *ICML*. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 5331–5340.

[Ram+22] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125* (2022).

[RK21] Maria Ryskina and Kevin Knight. Learning Mathematical Properties of Integers. *arXiv preprint arXiv:2109.07230* (2021).

[Sah+22] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, et al. Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding. *arXiv preprint arXiv:2205.11487* (2022).

[SLM18] Subham Sahoo, Christoph Lampert, and Georg Martius. Learning equations for extrapolation and control. In *International Conference on Machine Learning*. PMLR. 2018, pp. 4442–4450.

[SS97] Rafal Salustowicz and Jürgen Schmidhuber. Probabilistic incremental program evolution. *Evolutionary computation* 5.2 (1997), pp. 123–141.

[Sax+19] David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv:1904.01557* (2019).

[Sch87] Jurgen Schmidhuber. Evolutionary Principles in Self-Referential Learning. On Learning now to Learn: The Meta-Meta-Meta...-Hook. Diploma Thesis. Technische Universitat Munchen, Germany, 14 5 1987.

[SL09] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *science* 324.5923 (2009), pp. 81–85.

[SL11] Michael Schmidt and Hod Lipson. Age-fitness pareto optimization. In *Genetic programming theory and practice VIII*. Springer, 2011, pp. 129–146.

[Sch+15] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438* (2015).

[Sch+17a] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. *Proximal Policy Optimization Algorithms*. 2017.

[Sch+17b] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[Sek+20] Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *International Conference on Machine Learning*. PMLR. 2020, pp. 8583–8592.

[Sha+20] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-Aware Unsupervised Discovery of Skills. In *International Conference on Learning Representations*. 2020.

[Al-+17] Maruan Al-Shedivat, Trapit Bansal, Yuri Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. Continuous Adaptation via Meta-Learning in Nonstationary and Competitive Environments. *arXiv e-prints*, arXiv:1710.03641 (Oct. 2017), arXiv:1710.03641.

[Sil+16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, et al. Mastering the game of Go with deep neural networks and tree search. *nature* 529.7587 (2016), pp. 484–489.

[Sil+17] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, et al. Mastering chess and shogi by self-

play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815* (2017).

[Sil+18] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362.6419 (2018), pp. 1140–1144.

[Slo07] Neil JA Sloane. The on-line encyclopedia of integer sequences. In *Towards mechanized mathematical assistants*. Springer, 2007, pp. 130–130.

[SK05] Guido F Smits and Mark Kotanchek. Pareto-front exploitation in symbolic regression. In *Genetic programming theory and practice II*. Springer, 2005, pp. 283–299.

[Ste16] Trevor Stephens. *gplearn*. https://github.com/trevorstephens/gplearn. 2016.

[Str18] Steven H Strogatz. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. CRC press, 2018.

[Str+21] DJ Strouse, Kate Baumli, David Warde-Farley, Vlad Mnih, and Steven Hansen. Learning more skills through optimistic exploration. *arXiv preprint arXiv:2107.14226* (2021).

[Suk+17] Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407* (2017).

[Sun+22] Fangzheng Sun, Yang Liu, Jian-Xun Wang, and Hao Sun. Symbolic Physics Learner: Discovering governing equations via Monte Carlo tree search. *arXiv preprint arXiv:2205.13134* (2022).

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems* 27 (2014).

[Sut+99] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems* 12 (1999).

[Sut84] Richard Stuart Sutton. *Temporal credit assignment in reinforcement learning*. University of Massachusetts Amherst, 1984.

[Tas+18] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690* (2018).

[TS11] Matthew E. Taylor and Peter Stone. An Introduction to Inter-task Transfer for Reinforcement Learning. *AI Magazine* 32.1 (2011), pp. 15–34.

[Teh+17] Yee Whye Teh, Victor Bapst, Wojciech Marian Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, et al. Distral: Robust Multitask Reinforcement Learning. *CoRR* abs/1707.04175 (2017).

[TID23] Wassim Tenachi, Rodrigo Ibata, and Foivos I Diakogiannis. Deep symbolic regression for physics guided by units constraints: toward the automated discovery of physical laws. *arXiv preprint arXiv:2303.03192* (2023).

[Tho33] William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* 25.3/4 (1933), pp. 285–294.

[TP12] Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

[Tob+17] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. *Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World*. 2017.

[TET12] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2012, pp. 5026–5033.

[Tra+18] Andrew Trask, Felix Hill, Scott Reed, Jack Rae, Chris Dyer, and Phil Blunsom. Neural arithmetic logic units. *arXiv preprint arXiv:1808.00508* (2018).

[Udr+20] Silviu-Marian Udrescu, Andrew Tan, Jiahai Feng, Orisvaldo Neto, Tailin Wu, and Max Tegmark. AI Feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity (2020).

[UT20] Silviu-Marian Udrescu and Max Tegmark. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances* 6.16 (2020), eaay2631.

[Vad+20] Harsha Vaddireddy, Adil Rasheed, Anne E Staples, and Omer San. Feature engineering and symbolic regression methods for detecting hidden physics from sparse sensor observation data. *Physics of Fluids* 32.1 (2020), p. 015113.

[Val+21] Mojtaba Valipour, Bowen You, Maysum Panju, and Ali Ghodsi. SymbolicGPT: A Generative Transformer Model for Symbolic Regression. *arXiv preprint arXiv:2106.14131* (2021).

[Vap06] Vladimir Vapnik. *Estimation of dependences based on empirical data*. Springer Science & Business Media, 2006.

[Vas+22a] Martin Vastl, Jonás Kulhánek, Jirí Kubalík, Erik Derner, and Robert Babu*vs*.ka. SymFormer: End-to-end symbolic regression using transformer-based architecture. *ArXiv* abs/2205.15764 (2022).

[Vas+22b] Martin Vastl, Jonáš Kulhánek, Jirí Kubalík, Erik Derner, and Robert Babuška. SymFormer: End-to-end symbolic regression using transformer-based architecture. *arXiv preprint arXiv:2205.15764* (2022).

[Vas+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, et al. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[Vid+22] Mathurin Videau, Alessandro Leite, Olivier Teytaud, and Marc Schoenauer. Multi-objective Genetic Programming for Explainable Reinforcement Learning. In *European Conference on Genetic Programming* (*Part of EvoStar*). Springer. 2022, pp. 278–293.

[Vin+19] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575.7782 (2019), pp. 350–354.

[VAB19] Marco Virgolin, Tanja Alderliesten, and Peter A. N. Bosman. Linear Scaling with and within Semantic Backpropagation-Based Genetic Programming for Symbolic Regression. In *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '19. Prague, Czech Republic: Association for Computing Machinery, 2019, pp. 1084–1092.

[Vir+21] Marco Virgolin, Tanja Alderliesten, Cees Witteveen, and Peter AN Bosman. Improving model-based genetic programming for symbolic regression of small expressions. *Evolutionary computation* 29.2 (2021), pp. 211–237.

[VP22] Marco Virgolin and Solon P Pissis. Symbolic Regression is NP-hard. *Transactions on Machine Learning Research* (2022).

[VSH09] Ekaterina Vladislavleva, Guido Smits, and Dick den Hertog. Order of Nonlinearity as a Complexity Measure for Models Generated by Symbolic Regression via Pareto Genetic Programming. *IEEE Transactions on Evolutionary Computation* 13 (2009), pp. 333–349.

[WWR19] Yiqun Wang, Nicholas Wagner, and James M Rondinelli. Symbolic regression in materials science. *MRS Communications* 9.3 (2019), pp. 793–805.

[War+19] David Warde-Farley, Tom Van de Wiele, Tejas Kulkarni, Catalin Ionescu, Steven Hansen, and Volodymyr Mnih. Unsupervised Control Through Non-Parametric Discriminative Rewards. In *International Conference on Learning Representations*. 2019.

[Wel+22] Sean Welleck, Peter West, Jize Cao, and Yejin Choi. Symbolic brittleness in sequence models: on systematic generalization in symbolic mathematics. In *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 8. 2022, pp. 8629–8637.

[WYS15] David R White, Shin Yoo, and Jeremy Singer. The programming game: evaluating MCTS as an alternative to GP for symbolic regression. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. 2015, pp. 1521–1522.

[Wil92] Ronald J. Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning* 8 (1992), pp. 229–256.

[Wil+07] Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-Task Reinforcement Learning: A Hierarchical Bayesian Approach. In *Proceedings of the 24th International Conference on Machine Learning*. ICML '07. Corvalis, Oregon, USA: Association for Computing Machinery, 2007, pp. 1015–1022.

[WR16] Sam Wiseman and Alexander M. Rush. *Sequence-to-Sequence Learning as Beam-Search Optimization*. 2016.

[WM97] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation* 1.1 (1997), pp. 67–82.

[Wu18] Chai Wah Wu. Can machine learning identify interesting mathematics? An exploration using empirically observed laws. *arXiv preprint arXiv:1805.07431* (2018).

[Xie+21] Kevin Xie, Homanga Bharadhwaj, Danijar Hafner, Animesh Garg, and Florian Shkurti. Skill Transfer via Partially Amortized Hierarchical Planning. In *International Conference on Learning Representations*. 2021.

[Xu+20] Kelvin Xu, Siddharth Verma, Chelsea Finn, and Sergey Levine. Continual Learning of Control Primitives: Skill Discovery via Reset-Games. *Advances in Neural Information Processing Systems* 33 (2020).

[Yar+21] Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Reinforcement Learning with Prototypical Representations. In *Proceedings of the 38th International Conference on Machine Learning*. PMLR, 2021, pp. 11920–11931.

[YLT18] Wenhao Yu, C. Karen Liu, and Greg Turk. *Policy Transfer with Strategy Optimization*. 2018.

[Yu+17] Wenhao Yu, Jie Tan, C. Karen Liu, and Greg Turk. *Preparing for the Unknown: Learning a Universal Policy with Online System Identification*. 2017.

[Zha+22] Hengzhe Zhang, Aimin Zhou, Hong Qian, and Hu Zhang. PS-Tree: A piecewise symbolic regression tree. *Swarm and Evolutionary Computation* 71 (2022), p. 101061.

[ZYX21] Jesse Zhang, Haonan Yu, and Wei Xu. Hierarchical Reinforcement Learning by Discovering Intrinsic Options. In *International Conference on Learning Representations*. 2021.

[ZPG19] Wenxuan Zhou, Lerrel Pinto, and Abhinav Gupta. Environment Probing Interaction Policies (2019).

[Zin+19] Luisa Zintgraf, Maximilian Igl, Kyriacos Shiarlis, Anuj Mahajan, Katja Hofmann, and Shimon Whiteson. Variational Task Embeddings for Fast Adaptation in Deep Reinforcement Learning. *arXiv e-prints* (2019).