Addressing Dimensional Scaling in Reinforcement Learning for Symbolic Locomotion Policies through Leveraging Inductive Priors

Rogier Fransen, Richard Bowden and Simon Hadfield
Centre for Vision Speech and Signal Processing, University of Surrey, Guildford, Surrey, GU2 7XH, UK
{r.fransen, r.bowden, s.hadfield}@surrey.ac.uk

***Abstract*— We explore symbolic policy optimization for various legged locomotion challenges; specifically walker environments ranging from bipedal to highly redundant systems with 128 legs. These represent a broad range of action space dimensionalities. We find that state-of-the-art symbolic policy optimization approaches struggle to scale to these higher dimensional problems, due to the need to iterate over action dimensions, and their reliance on a neural network anchor policy. We thus propose Fast Symbolic Policy (FSP) to accelerate the training of symbolic locomotion policies. This approach avoids the need to iterate over the action dimensions, and does not require a pre-trained neural network anchor. We also propose Dim-X, a method for effectively reducing the action space dimensionality using the inductive priors of legged locomotion. We demonstrate that FSP with Dim-X can learn symbolic policies, with improved scaling performance compared to the baselines, vastly exceeding that possible with previous symbolic techniques. We further show that Dim-X on its own can also be integrated into neural network policies to shorten their training time and improve scaling performance.**

I. INTRODUCTION

Modern deep Reinforcement Learning (RL) is most frequently used for training neural network control policies. However, some more recent works have shown how the same RL frameworks can also be used for learning symbolic control policies [1], [2]. These symbolic policies are constructed from a collection of symbolic mathematical expressions, describing how to map observed input sates to output actions, to achieve a particular goal. In practice, these expressions are produced using sequence generation models with a token library comprising the observation inputs as well as binary and unary mathematical symbols (e.g. $+, \times, \sin$ and $\exp$). These symbolic policies have a number of very attractive properties compared to their neural network counterparts. They are extremely computationally efficient at inference time, and require no specialist hardware to run, making them ideal for power-constrained or edge devices. They are also highly transparent and interpretable, raising the possibility of autonomous systems with provable safety bounds that can be safety-certified.

The challenge with symbolic policies as opposed to neural network policies, is that mathematical expressions are natively scalar constructs with a single output. For multi-dimensional action spaces, numerous independent expressions must be created without any exploitation of redundancies. This means in practice that previous approaches to RL

This work was supported by the Engineering & Physical Sciences Research Council under grant agreement EP/S035761/1

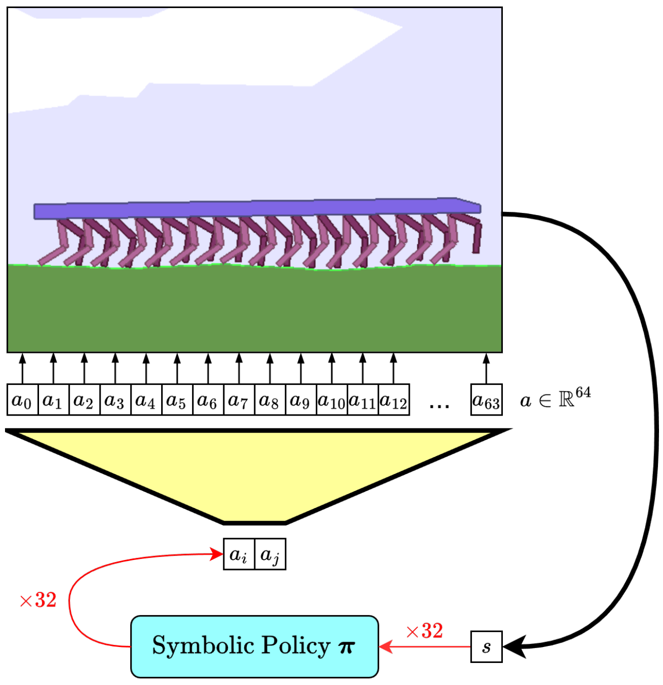

a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9 a_{10} a_{11} a_{12} … a_{63}
$a \in \mathbb{R}^{64}$
a_i a_j
×32
Symbolic Policy π
×32
s

Fig. 1: Effective locomotion in a highly redundant 32 legged walker environment with a small symbolic policy

symbolic policy optimization can scale poorly to complex or higher dimensional problems. This issue is exacerbated by the fact that recent approaches require a neural network anchor policy to be used to support the sequential symbolic training of each action dimension [2]. In practice, this implies that you must first solve the problem using traditional Deep RL techniques (which may itself scale poorly) before you can begin "upgrading" the resulting policy to a symbolic form. Not only can this regime take orders of magnitude longer to train, it also limits the breadth of possible applications, and stifles the potential symbolic policies that may have been generated.

In this paper, we explore RL symbolic policy optimization specifically within the domain of learning legged locomotion control policies. Due to the complexity of learning the control policy, the majority of existing legged systems contain a relatively small number of legs (i.e. bipedal or quadrupedal), keeping the dimensionality of the action space small. However, systems with larger numbers of legs have many practical benefits, including greatly increased tolerance to hardware failure, increased stability, and the ability to surmount more complex obstacles. Unfortunately, the control policies of these more capable multi-legged systems are complex to learn as there are more independent control circuits to derive. Even if these control circuits exhibit significant phase-based correlations, the control circuits have no mechanism to exploit this redundancy during training.

Therefore, in this paper we propose Fast Symbolic Policy (FSP) and Dim-X, two complementary contributions to address these challenges and enhance the training procedure. FSP is a novel anchor-free RL symbolic policy optimization framework for learning all action dimensions in a symbolic policy simultaneously, and Dim-X is a novel method for reducing the effective dimensionality of a locomotion action space through leveraging the inductive priors of legged locomotion.

In FSP, multiple related but non-redundant action dimensions are learned jointly through a multi-dimensional sequence generator. This addresses the challenges with traditional sequential anchor-based training regimes. Dim-X exploits locomotion-specific inductive biases to drastically reduce the size and redundancy of the control policies needed for systems with large numbers of legs, as can be seen in Figure 1. We do this by sharing the expressions of a symbolic control policy between the different legs, while enforcing different phase offsets and observation input transformations onto the policy based on the positions of the legs. This makes it possible to exploit the cyclical nature of a traditional legged gait, and the self-similarity in the behaviors between different legs, while retaining the flexibility of a deep RL training regime and the efficiency of a symbolic policy.

To fully clarify the separation between the different contributions: If we imagine a hexapod robot with 6 legs, each having 3 joints, a modern RL symbolic policy optimization approach would first train an 18-dimensional neural network anchor policy, followed by 18 loops of symbolic policy learning. In contrast, FSP allows us to jointly learn symbolic policies for the hip, knee, and ankle joints of a leg within a single process. Concurrently, Dim-X provides a mechanism for these 3 jointly-learned policies to be transformed and deployed across all 6 of the robot's legs. Thus only a single 3-dimensional training process must be completed to control the robot.

Within this paper, we exhaustively test the various combinations of our proposed contributions across both symbolic and traditional deep-policy RL regimes. Specifically, we test Dim-X alone for learning both neural network and symbolic control policies, then we evaluate both FSP on its own and FSP with Dim-X for learning symbolic policies. These tests are carried out on 7 different walker environments, each with a different number of legs, ranging from a walker with 2 legs to one with 128 legs.

II. Literature Review

A. Symbolic Regression

Symbolic regression is the process of identifying mathematical expressions that fit the observed output data from a process or system. Early methods utilized evolutionary algorithms [3], such as genetic programming [4], whereas modern techniques leverage deep learning for symbolic regression. Petersen et al. [1] propose such a framework. They use a Recurrent Neural Network (RNN) to emit a distribution over mathematical expressions and employ a novel risk-seeking policy gradient to train the network to generate better-fitting expressions. However, Landajuela et al. [5] demonstrate that this framework can suffer from an early commitment phenomenon and from initialization bias, both of which limit exploration. They present two exploration methods to tackle these issues, building upon ideas of entropy regularization and distribution initialization. Furthermore, Mundhenk et al. [6] build upon this and show that introducing a genetic programming component into this framework can improve the expression recovery rate.

Other frameworks utilize a transformer instead of an RNN to generate the expressions, which shows performance benefits in some metrics [7]–[9]. Additionally, some frameworks leverage pre-trained models to learn solutions more efficiently, such as Holt et al. [10] which use generative models and Silva et al. [11] which use language models. Neuro-symbolic approaches have also been explored, which leverage the advantages of both neural and symbolic methods [12]–[14].

B. Application to Control

RL has been extensively applied to the problem of learning locomotion controllers for legged systems. These controllers are most frequently in the form of a neural network [15]–[17]. More recent works have proposed using symbolic regression for control, including for the control of legged systems [2], [18]. The main challenge with this is that legged systems often have multidimensional action spaces, which means that their symbolic policy will contain multiple expressions. Existing symbolic regression methods can only learn one expression at a time. Landajuela et al. [2] address this by proposing an "anchoring" algorithm that distills pre-trained neural network-based policies into fully symbolic policies, one action dimension at a time. This has the clear limitation that a neural network policy needs to have solved the control problem first before a symbolic solution can be learned. To address this, we propose FSP, a method which enables multiple expressions to be sampled at once, which are then jointly evaluated. This crucially avoids the need for a neural network anchoring policy.

III. Method

A. Fast Symbolic Policy (FSP)

FSP is an RL approach which seeks to produce symbolic policies π that map from observed input states s to output actions a. As outlined in Figure 2, these policies take the form of tokenized sequences, which can be interpreted as mathematical control equations. An RNN policy generator recurrently samples these tokens from a library of tokens and jointly generates these control equations for each action dimension. Then Dim-X transforms observed input states to share the sampled policy between the different limbs.

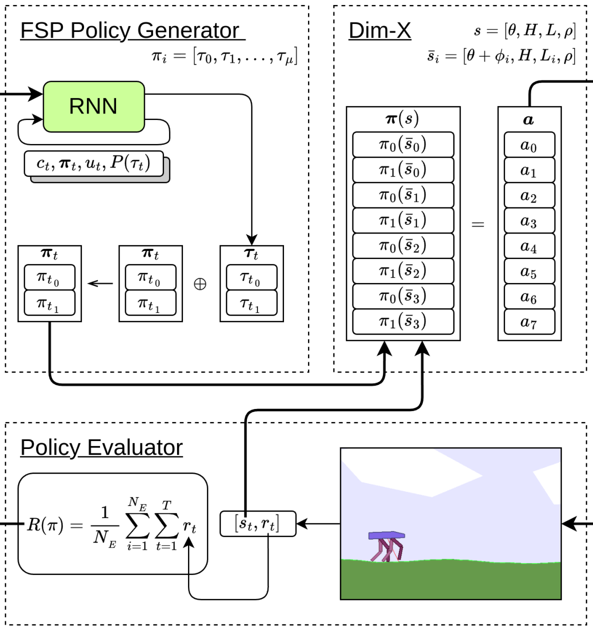


Fig. 2: Flow diagram of the FSP with Dim-X system. The environment has 4 legs, with 2 joints each, therefore $a \in \mathbb{R}^8$. Dim-X is used to reduce the dim of the action space to 2 (i.e. one for each joint in a leg) meaning that the symbolic policy contains 2 expressions, which are used 4 times to get the actions for all the legs

The training of the policy generator proceeds by first sampling a batch of such symbolic policies. Each policy then interacts with an environment in parallel to gather their respective rewards. The policy generator is then updated to favor the generation of policies which succeeded in the environment, and to avoid policies with low rewards. The process is then repeated.

Policy Definition Within our FSP framework, a multi-dimensional symbolic control policy is defined as a collection of scalar expressions all using the same state input

$$\pi(s) = [\pi_0(s), \pi_1(s), ..., \pi_n(s)], \tag{1}$$

where n is the dimensionality of the action space. Each scalar sub-policy $\pi_i(s)$ is itself a symbolic expression formed out of a sequence of tokens

$$\pi_i = [\tau_0, \tau_1, ..., \tau_\mu] \quad \text{where} \quad \tau_i \in \mathcal{L}, \tag{2}$$

μ is the number of tokens in the sequence and $\mathcal{L}$ is the library of available tokens, which in our experiments consists of $\{s_0, s_1, ..., s_n, +, -, \times, \div, \sin, \cos, \exp, \log, 0.1, 1.0, 5.0\}$.

Policy Generator Formulation As shown in Figure 2, the policy generator is an RNN which takes cell state c_t, sampled policy π_t, input u_t and prior $P(\tau_t)$ as input. The cell state, input and prior are defined using the standard Deep Symbolic Policy (DSP) formulation [2], extended with multiple dimensions to allow for the sampling of multiple expressions simultaneously. The input $u_t = [\tau_{t-1}, \tau_{\text{parent}}, \tau_{\text{sibling}}, N_{\text{dangling}}]$ where τ_{t-1} is the previously sampled stack of tokens, τ_{parent} and τ_{sibling} are the stacks of parent and sibling tokens of τ_{t-1} in the expression tree, and N_{dangling} is the vector of cumulative arities of the expression so far (i.e. the number of nested and unclosed subexpressions). The prior $P(\tau_t)$ encourages higher arity tokens at start of the sequence and encourages terminal tokens later in the sequence.

Let u_0 be the initial input, $\bar{u}_t$ be the one hot encoding of u_t, c_0 be the initial RNN cell state and $P(\tau_0)$ be the initial prior. Then the policy generator is defined as a recurrent process. At each iteration, a softmax is applied to the cell state in order to extract the token likelihoods

$$c_{t+1} = \text{RNN}(\bar{u}_t, c_t), \tag{3}$$

$$P(\bar{u}_t|\tau_t) = \text{SMax}(c_{t+1}). \tag{4}$$

These likelihoods are combined with the token prior following Bayes' rule, noting that both arguments are in log form

$$P(\tau_t|\bar{u}_t) = P(\bar{u}_t|\tau_t) + P(\tau_t), \tag{5}$$

and the policy's next stack of tokens are sampled and concatenated

$$\tau_t \sim P(\tau_t|\bar{u}_t), \quad \pi_i \leftarrow \pi_i \oplus \tau_t. \tag{6}$$

After each newly sampled stack of tokens, the input state is updated. The process then repeats until either t exceeds the maximum allowed length, or all the symbolic expressions are fully complete (with the same number of terminal tokens as branching tokens).

B. Joint Policy Evaluator

On each loop of FSP, an entire batch of possible policies are sampled from the policy generator following the above approach. The sampled batch of symbolic policy expressions is then collectively evaluated by a joint policy evaluator. This runs each complete policy in parallel for N_E episodes and obtains its return as the sum of the rewards at each step. The average return R for a single policy across episodes is thus defined as

$$R(\pi) = \frac{1}{N_E}\sum_{i=1}^{N_E}\sum_{t=1}^{T} r_t, \tag{7}$$

where π is the set of token sequences, N_E is the number of episodes, T is the length of the episode and r_t is the reward at timestep t. The return is then passed to the FSP loss function to update the policy generator.

C. Fast Symbolic Policy Loss

Our FSP policy generator is trained using an extension of the risk-seeking policy gradient algorithm [1], with entropy regularization [2]. More specifically, we optimize the expectation of the returns across the generated policy batch.

The policy generator loss takes the sampled policy π, the input u and the return of the policy in the environment R, from the filtered batch $\mathcal{B} = \{(\pi, u, R) \mid R \geq R_\epsilon\}$. This contains all the experiences whose return is greater than or equal to the $(1-\epsilon)$ quantile of rewards R_ϵ, where ϵ is a hyperparameter controlling the degree of risk-seeking. The loss is defined as

$$\mathcal{L}(\psi) = -\mathop{\mathbb{E}}_{(\pi,u,R)\sim\mathcal{B}}\left[\sum_{\pi}\log\pi_\psi(\pi|u)\cdot(R - R_\epsilon)\right] - \beta\cdot\mathop{\mathbb{E}}_{(\pi,u)\sim\mathcal{B}}\left[\mathcal{H}(\pi_\psi(\cdot|u))\right], \tag{8}$$

where $\pi_\psi(\pi|u)$ is the probability of that policy according to the policy generator and β is a hyperparameter controlling the weight of the entropy term. The entropy term itself is computed as

$$\mathcal{H}(\pi_\psi(\cdot|u)) = -\sum_{\pi} \pi_\psi(\pi|u) \log \pi_\psi(\pi|u). \tag{9}$$

This represents the entropy of the policy generator itself rather than that of the generated policies.

D. *Dim-X*

The proposed Dim-X approach attempts to effectively reduce the dimensionality of the action space. To achieve this, it leverages several inductive biases related to multi-legged locomotion. The first is that in the absence of obstacles, the optimal gait for an individual leg is cyclical in nature. The second is that the different legs should behave similarly to each other, under the same circumstances and at the same point in their walking cycle (noting that the walking cycles of different legs need not be in phase with each other).

Thus, the Dim-X regime drastically reduces the number of control equations needed from the policy generator, by sharing a smaller set of symbolic expressions across all the legs. The only requirement is that the input observations must be transformed into a different "leg-centric" frame and the walking cycle must be offset by an appropriate leg specific phase (which can be a learnable parameter).

More formally, the global observation space without Dim-X is defined as

$$s = [\theta, H, L, \rho], \tag{10}$$

where θ is the sine of the timestep, H is the hull pose information, L is the concatenated pose information for all the legs and ρ are the environmental readings (e.g. lidar).

We then define the leg-specific transformed observation space as

$$\bar{s}_i = [\theta + \phi_i, H, L_i, \rho], \tag{11}$$

where ϕ_i is the phase shift for leg i and L_i is the subset of the leg pose information for leg i.

We then define the result of the overall control policy for the robot $\boldsymbol{\pi}$ as the concatenation of the results from the set of expressions for a single leg, applied to each leg-specific state observation

$$\boldsymbol{\pi}(s) = [\pi(\bar{s}_0), \pi(\bar{s}_1), ..., \pi(\bar{s}_N)], \tag{12}$$

where N is the number of legs. We note that this approach can be applied to both our FSP scheme as well as traditional deep-policy RL approaches utilizing neural networks. We contrast both options in our experiments.

E. *Fast Symbolic Policy with Dim-X*

Our combined FSP with Dim-X framework introduces the FSP policy definition in equation 1 into the Dim-X approach defined in equation 12. This joint framework is thus defined as

$$\begin{aligned}\boldsymbol{\pi}(s) = [&\pi_0(\bar{s}_0), \pi_1(\bar{s}_0), ...\pi_n(\bar{s}_0),\\ &\pi_0(\bar{s}_1), \pi_1(\bar{s}_1), ...\pi_n(\bar{s}_1),\\ &...,\\ &\pi_0(\bar{s}_N), \pi_1(\bar{s}_N), ...\pi_n(\bar{s}_N)].\end{aligned} \tag{13}$$

This yields an efficient framework which benefits from the advantages of both FSP and Dim-X. The joint framework thus avoids the need to iterate over action dimensions, therefore not requiring a neural network anchor policy, and reduces the complexity of the training challenge, through leveraging the inductive priors of legged locomotion, further speeding up the training time.

IV. RESULTS

A. *Experimental Setup*

We evaluate our technique using multiple custom variations of the Box2D [19] OpenAI Gym bipedal walker environment [20]. These new environments include walkers with 4, 8, 16, 32, 64 and 128 legs and are shown in Figure 3. This represents a broad range of action space dimensionalities allowing us to evaluate the scaling performance of our technique.

The main technique we present in this paper is FSP with Dim-X (FSP w/ Dim-X), however we also evaluate FSP on its own as an ablation. We compare these to a neural network (NN) and a DSP baseline. Then as an independent evaluation of our Dim-X approach, we also evaluate it on its own applied to both a neural network (NN w/ Dim-X) and a DSP policy (DSP w/ Dim-X). We compare the scaling performance of these different frameworks against 3 metrics, the distance covered (in 1000 timesteps), the train time (per training iteration) and the inference time (per iteration).

B. *Neural Network Baselines*

For the neural network baseline experiments, and the experiments applying Dim-X to neural network policies, we trained using the StableBaselines3 [21] Twin Delayed Deep Deterministic Policy Gradient (TD3) RL algorithm [22]. TD3 consists of two loss functions, a critic loss and an actor loss. To mitigate overestimation bias, it employs two critic networks. The critic loss takes the observations s, actions a, rewards r, and next observations s' from the replay buffer $\mathcal{D}$. The loss for each critic is defined as

$$\mathcal{L}(\theta_i) = \mathop{\mathbb{E}}_{(s,a,r,s')\sim\mathcal{D}} \left[\left(Q_{\theta_i}(s,a) - y \right)^2 \right], \tag{14}$$

where Q_{θ_i} are the critic networks and y is the expected Q value at the next state. This is defined as

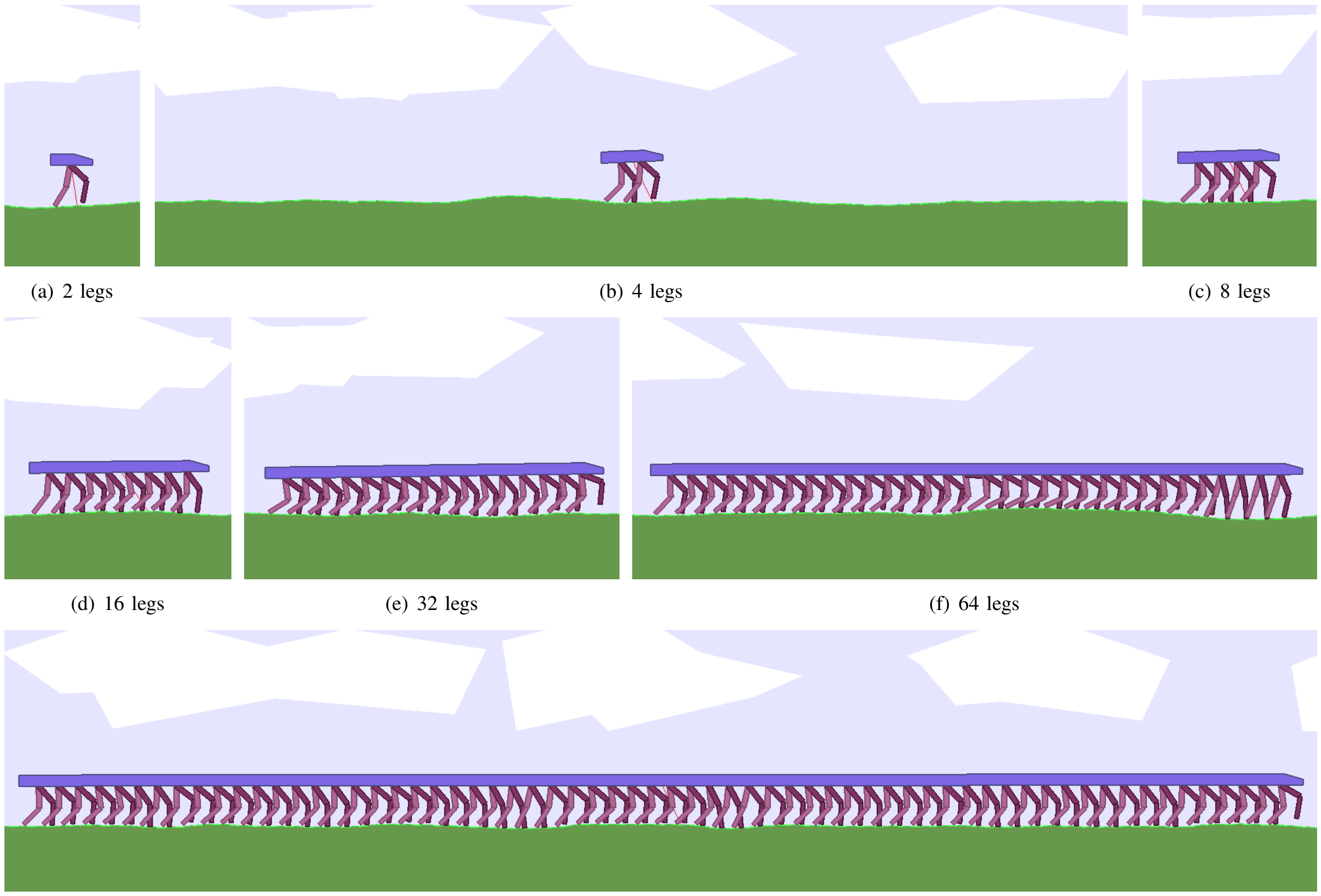
$$y = r + \gamma \min_{j=1,2} Q_{\theta'_j}(s', a'), \tag{15}$$

where γ is the discount factor and a' is the action at the next state selected as $a' = \pi_{\phi'}(s') + \epsilon$, with $\epsilon \sim \mathcal{N}(0, \sigma)$ being clipped noise scaled by the target actor noise hyperparameter σ.

The actor loss only utilizes one of the critic networks, Q_{θ_1}, and is defined as

$$\mathcal{L}(\phi) = -\mathop{\mathbb{E}}_{s\sim\mathcal{D}} \left[Q_{\theta_1}(s, \pi_\phi(s)) \right], \tag{16}$$

where π_ϕ is the current actor network.



(a) 2 legs (b) 4 legs (c) 8 legs

(d) 16 legs (e) 32 legs (f) 64 legs

(g) 128 legs

Fig. 3: The different walker environments

C. Deep Symbolic Policy (DSP) Baselines

For the DSP baseline experiments, and the experiments applying Dim-X to DSP, we train using the standard DSP framework [2]. As described in their paper, this approach can only train the symbolic control expression for one action dimension at a time. It is obvious that in this case the training time and complexity scale poorly to environments with higher dimensional action spaces. Adding to this is the requirement for a neural network anchor policy, that has already solved the control problem. This is needed to populate the actions for dimensions that are not currently being learned. This therefore has to be trained beforehand adding an extra step to the training procedure.

The neural network baseline policies were used as the anchor policies for the DSP baseline experiments, and the neural network with Dim-X policies were used as the anchor policies for the DSP with Dim-X experiments. DSP with Dim-X already presents a massive improvement over default DSP, for environments with higher dimensional action spaces, as it drastically reduces the effective size of the action space for the policy. This therefore reduces the number of action dimensions that have to be iterated over and the number of independent symbolic expressions that have to be learned.

D. Simulation Environment

The simulation environment utilized is a custom variation of the common bipedal walker environment. This custom environment takes the desired number of legs as an input argument and recursively builds the walker. It also scales the action and observation spaces accordingly. These different legged environments can be seen in Figure 3. The reward function for the environment is the standard bipedal walker reward formulation, taking into consideration the additional action dimensions. The reward r_t is thus defined as

$$r_t = x - x_{\text{prev}} - (|\theta| - |\theta_{\text{prev}}|) - \sum \text{clip}(|a|, 0, 1) \quad (17)$$

where x and x_{prev} are the current and previous horizontal positions of the hull respectively, θ and θ_{prev} are the current and previous orientations of the hull respectively, and a are the actions.

The $x - x_{\text{prev}}$ portion of the reward aims to maximize the progress in the positive horizontal direction. The $|\theta| - |\theta_{\text{prev}}|$ cost aims to minimize changes in the hull orientation between timesteps, thereby keeping the walker upright and penalizing tipping over. Finally, the $\sum \text{clip}(|a|, 0, 1)$ cost aims to minimize the amount of energy used at each timestep. The actions are torques so their sum is the energy used. Minimizing the energy used penalizes impractical gaits with unnecessary leg movements that do not contribute to the goal of moving forward.

E. Experimental Results

Framework vs Distance Traversed Each of the frameworks were trained for 1000 iterations, with a batch size of 1000. However, note that for the frameworks which must iterate over the action dimensions (DSP, DSP w/ Dim-X), 1000 iterations were carried out on each action dimension. For the DSP baseline, the total number of iterations completed is dependent on the number of legs (e.g. with 16 legs it would train for 16,000 iterations). For the DSP w/ Dim-X experiments, this was reduced to just 2000 iterations for each of the walker configurations, as Dim-X reduced the effective dimensionality of the action space to 2 in each case from the policy's perspective.

The resultant policies obtained from the different frameworks were then run on the different walker configurations for 1000 timesteps, and the final distances in each case were recorded. Each evaluation was repeated 100 times and the results are presented in Table I and plotted in Figure 4.

TABLE I: Distance traversed in 1000 timesteps

Framework	Number of legs						
	2	4	8	16	32	64	128
NN (baseline)	**21.5**	**22.5**	**20.5**	12.6	6.7	4.1	2.0
DSP (baseline)	21.3	21.1	19.7	9.4	-	-	-
NN w/ Dim-X	20.2	21.4	18.2	16.3	14.9	14.0	13.8
DSP w/ Dim-X	19.9	20.2	17.1	14.4	12.3	10.6	9.9
FSP (ours)	20.7	21.5	9.4	2.8	1.5	0.2	0.2
FSP w/ Dim-X (ours)	21.4	21.8	19.9	**17.9**	**16.4**	**15.4**	**14.3**

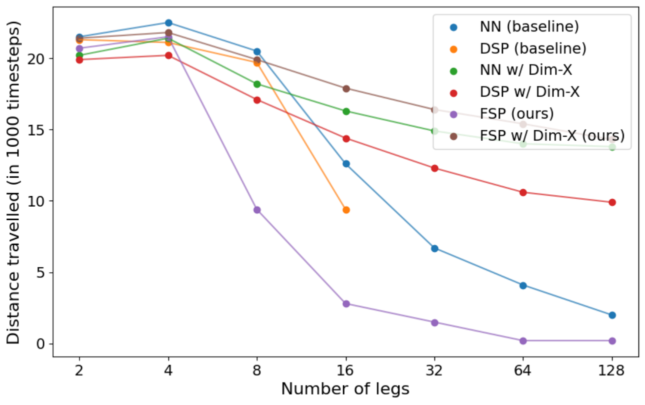


Fig. 4: Chart showing the distance traveled in 1000 timesteps vs the different numbers of legs for the different frameworks

It can be seen from Figure 4 that the 3 frameworks utilizing Dim-X (NN w/ Dim-X, DSP w/ Dim-X, FSP w/ Dim-X) have the greatest scaling performance to higher dimensional action spaces (number of legs). This shows that Dim-X effectively reduces the complexity of the control challenges in each case it is applied. However, both the DSP baseline and our independent FSP framework struggle to scale effectively. This is likely because of the added complexity of the control problems, thus requiring more training iterations to derive an optimal policy. Furthermore, the DSP baseline could not feasibly be trained beyond 16 legs within a reasonable amount of time (24 hours), so these experiments were omitted.

The independent FSP framework struggled to train beyond 8 legs due to the exponentially increasing size of the search space, and the relatively few iterations it was allowed to carry out compared to the number of legs in the environment. However, it can be seen that the application of Dim-X to FSP resolves this issue.

Framework vs Training Time The average time taken for 1 full training iteration was recorded for each of the frameworks and is presented in Table II and plotted in Figure 5. Note that for those frameworks which require a neural network anchor policy (DSP, DSP w/Dim-X), the time taken to train these anchors was not taken into account.

TABLE II: Time taken in seconds for 1 training iteration

Framework	Number of legs						
	2	4	8	16	32	64	128
NN (baseline)	23.8	26.2	28.8	30.9	34.1	42.7	64.6
DSP (baseline)	337.0	1708.0	8476.6	27349.9	-	-	-
NN w/ Dim-X	26.4	32.9	38.7	45.7	52.5	68.6	89.3
DSP w/ Dim-X	337.4	354.0	1919.1	3418.7	5868.5	9454.6	15873.7
FSP (ours)	168.5	277.0	859.5	1409.3	2134.2	3285.3	7593.1
FSP w/ Dim-X (ours)	193.9	304.8	959.3	1679.0	2422.5	3568.7	8135.7

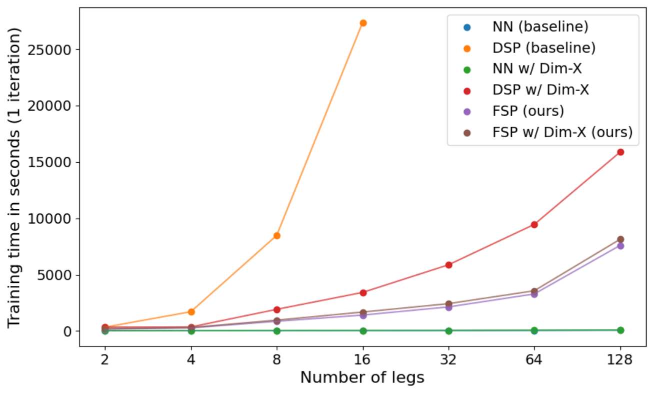


Fig. 5: Chart showing the training time for 1 iteration vs the different numbers of legs for the different frameworks

Figure 5 shows that the training time scaling performance of the DSP baseline is by far the worst. This makes sense as the framework has to iterate over all of the action dimensions one by one. DSP w/ Dim-X has far better scaling performance as Dim-X reduces the number of action dimensions that have to be iterated over. Then both our FSP methods improve on this and have similar performance to each other. This is because they sample the expressions for all of the action dimensions at once, therefore not needing to iterate over them. Then finally both NN methods have the best scaling performance, as they are also able to train in parallel and do not have a recursive policy generation process.

Framework vs Inference Time The average time taken for one inference iteration was measured for each trained policy obtained from the different frameworks and is presented in Table III and plotted in Figure 6.

TABLE III: Time taken in nanoseconds for 1 iteration at inference

Framework	Number of legs						
	2	4	8	16	32	64	128
NN (baseline)	486.7	495.1	509.2	520.2	539.9	553.0	574.1
DSP (baseline)	16.2	34.8	60.6	92.0	-	-	-
NN w/ Dim-X	780.7	799.1	824.4	855.7	879.3	899.4	921.2
DSP w/ Dim-X	19.5	41.4	69.9	111.4	178.6	281.8	402.8
FSP (ours)	16.5	31.4	65.2	97.3	150.9	251.3	392.5
FSP w/ Dim-X (ours)	19.1	40.7	63.3	93.8	162.7	237.2	399.8

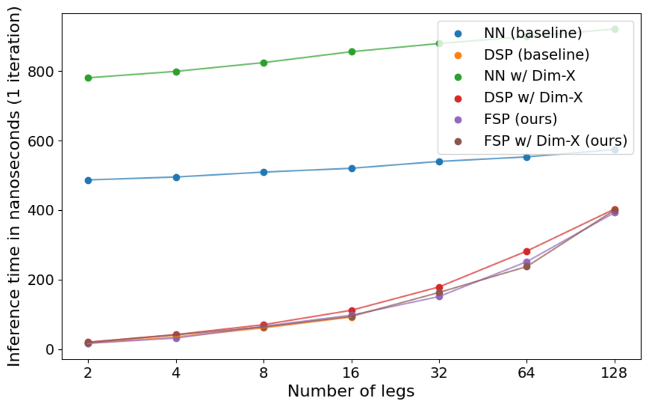


Fig. 6: Chart showing the inference time for 1 iteration vs the different numbers of legs for the different frameworks

Figure 6 shows that the symbolic policy based frameworks are faster at inference time than the neural network based frameworks. This is because symbolic policies are made up of mathematical expressions, which are very fast to compute. In contrast to this, the neural network based policies have to perform many linear operations, which all together take a longer time. For the symbolic policy frameworks, it can also be seen that the inference time scales fairly linearly with the number of legs, which makes sense as the number of expressions the policy has the compute increases linearly with the number of legs.

V. CONCLUSIONS

The work undertaken in this paper has led to Fast Symbolic Policy (FSP) with Dim-X, two distinct contributions which each have clear benefits in scaling performance over existing state-of-the-art RL symbolic policy optimization methods. We show that the technique scales effectively in terms of policy success, training time, and inference time across a broad range of action space dimensionalities. Additionally, we demonstrate that FSP can effectively train symbolic policies without needing a neural network anchor policy, which is a major advantage of the technique. Dim-X significantly reduces the training complexity of higher dimensional environments, by leveraging the inductive priors of legged locomotion, to reduce the number of action dimensions that have to be iterated over. We further show how Dim-X can be used in other frameworks, including those using non-symbolic policies. Overall, both proposed contributions independently have clear advantages, and they effectively complement each other when combined, resulting in a framework which inherits the benefits of both.

REFERENCES

[1] B. K. Petersen, M. L. Larma, T. N. Mundhenk, C. P. Santiago, S. K. Kim, and J. T. Kim, "Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients," in *International Conference on Learning Representations (ICLR)*, 2021.

[2] M. Landajuela, B. K. Petersen, S. Kim, C. P. Santiago, R. Glatt, N. Mundhenk, J. F. Pettit, and D. Faissol, "Discovering symbolic policies with deep reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2021.

[3] N. L. Cramer, "A representation for the adaptive generation of simple sequential programs," in *International Conference on Genetic Algorithms (ICGA)*, 1985.

[4] J. R. Koza, "Hierarchical genetic algorithms operating on populations of computer programs," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 1989.

[5] M. Landajuela, B. K. Petersen, S. K. Kim, C. P. Santiago, R. Glatt, T. N. Mundhenk, J. F. Pettit, and D. M. Faissol, "Improving exploration in policy gradient search: Application to symbolic optimization," in *Mathematical Reasoning in General Artificial Intelligence Workshop, International Conference on Machine Learning (ICML)*, 2021.

[6] T. Mundhenk, M. Landajuela, R. Glatt, C. P. Santiago, D. faissol, and B. K. Petersen, "Symbolic regression via deep reinforcement learning enhanced genetic programming seeding," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[7] P.-A. Kamienny, S. d'Ascoli, G. Lample, and F. Charton, "End-to-end symbolic regression with transformers," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[8] P. Shojaee, K. Meidani, A. B. Farimani, and C. K. Reddy, "Transformer-based planning for symbolic regression," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

[9] S. Becker, M. Klein, A. Neitz, G. Parascandolo, and N. Kilbertus, "Predicting ordinary differential equations with transformers," in *International Conference on Machine Learning (ICML)*, 2023.

[10] S. Holt, Z. Qian, and M. van der Schaar, "Deep generative symbolic regression," in *International Conference on Learning Representations (ICLR)*, 2023.

[11] F. Silva, A. Goncalves, S. Nguyen, D. Vashchenko, R. Glatt, T. Desautels, M. Landajuela, D. Faissol, and B. Petersen, "Language model-accelerated deep symbolic optimization," *Neural Computing and Applications*, 2023.

[12] D. Yu, B. Yang, D. Liu, H. Wang, and S. Pan, "A survey on neural-symbolic learning systems," *Neural Networks*, 2023.

[13] Q. Delfosse, H. Shindo, D. S. Dhami, and K. Kersting, "Interpretable and explainable logical policies via neurally guided symbolic abstraction," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

[14] K. Acharya, W. Raza, C. Dourado, A. Velasquez, and H. H. Song, "Neurosymbolic reinforcement learning and planning: A survey," *IEEE Transactions on Artificial Intelligence*, 2024.

[15] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "Rma: Rapid motor adaptation for legged robots," *Robotics: Science and Systems*, 2021.

[16] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, 2020.

[17] C. Yang, K. Yuan, Q. Zhu, W. Yu, and Z. Li, "Multi-expert learning of adaptive legged locomotion," *Science Robotics*, 2020.

[18] W. Zheng, S. P. Sharan, Z. Fan, K. Wang, Y. Xi, and Z. Wang, "Symbolic visual reinforcement learning: A scalable framework with object-level abstraction and differentiable expression search," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 2025.

[19] E. Catto, "Box2d physics engine," *http://box2d.org*, 2006.

[20] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv:1606.01540*, 2016.

[21] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," in *Journal of Machine Learning Research*, 2021.

[22] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning (ICML)*, 2018.