Explorative Imitation Learning: A Path Signature Approach for Continuous Environments

Nathan Gavenski [a,*], **Juarez Monteiro, Felipe Meneguzzi**[b,c], **Michael Luck**[d] **and Odinaldo Rodrigues**[a]

[a]King's College London, London, United Kingdom
[b]University of Aberdeen, Aberdeen, United Kingdom
[c]Pontifícia Universidade Católica do RS, Porto Alegre, Brazil
[d]University of Sussex, Sussex, United Kingdom

Abstract. Some imitation learning methods combine behavioural cloning with self-supervision to infer actions from state pairs. However, most rely on a large number of expert trajectories to increase generalisation and human intervention to capture key aspects of the problem, such as domain constraints. In this paper, we propose Continuous Imitation Learning from Observation (CILO), a new method augmenting imitation learning with two important features: (i) exploration, allowing for more diverse state transitions, requiring less expert trajectories and resulting in fewer training iterations; and (ii) path signatures, allowing for automatic encoding of constraints, through the creation of non-parametric representations of agents and expert trajectories. We compared CILO with a baseline and two leading imitation learning methods in five environments. It had the best overall performance of all methods in all environments, outperforming the expert in two of them.

1 Introduction

One of the most common forms of learning is by watching someone else perform a task and, afterwards, trying it ourselves. As humans, we can observe an action being performed and transfer the acquired knowledge into our reality. In this respect, it is less challenging to achieve a goal in an optimal way by observing how an expert behaves; in the field of computer science, this is Imitation Learning (IL). Unlike conventional reinforcement learning, which depends on a reward function, IL learns from expert guidance, and is concerned with an agent's acquisition of skills or behaviours by observing a 'teacher' perform a given task.

Learning from demonstration is the obvious approach for IL, requiring expert demonstrations, which are 'trajectories' including actions performed along the way to goal completion [12]. Such an approach uses the trajectories to learn an approximate policy that behaves like the expert. Learning from demonstration suffers from two significant drawbacks in practice: poor generalisation in environments with multiple alternative trajectories that achieve a goal, which is bound to occur when the dataset size increases, and the unavailability of data about the expert's actions. *Learning from observation* (LfO) overcomes these limitations by learning a task without direct action information via self-supervision, which increases generalisation [8]. This allows a model to learn from sample executions without action information, which would otherwise be unusable. LfO approaches often rely on techniques from classification to improve sample-efficiency [30] and generalisation [18]. Such agents require fewer expert trajectories, yielding more general approaches that are, hence, adaptable to unseen scenarios. However, these methods still fail to leverage some useful learning features, particularly the use of an exploration mechanism.

Some existing work [5, 7] requires manual intervention in different stages of the process, e.g., the hard-coding of environment goals, which is not feasible in complex environments, such as robotic systems with multifaceted goals. Other work [7, 13, 30] is limited in that learning the environment dynamics depends strongly on previously collected samples that usually do not relate to how the environment dynamics operate under expert behaviour, such as random transitions, or prior knowledge of the dynamics of environments. In addition, maintaining self-supervision [7, 13] for an IL method is important since unlabelled data is more readily available, *e.g.*, from sources that are not necessarily meant for agent learning.

In this paper, we propose a novel LfO approach to IL called Continuous Imitation Learning from Observation (CILO) that addresses the above issues. CILO (i) eliminates the need for manual intervention when using different environments by discriminating between policy and expert; (ii) requires fewer samples for learning by leveraging exploration and exploitation; and (iii) does not require expert-labelled data, thus remaining self-supervised. We evaluated CILO in five widely used continuous environments against a baseline and two leading LfO methods (see Section 4). Our results show that CILO outperformed all of the alternatives, surpassing the expert in two of five environments.

CILO's new mechanisms are model-agnostic and applicable to a wider range of environment dynamics than those of the compared LfO alternatives. We argue that the new mechanisms can be readily incorporated into other IL methods, paving the way for more robust and flexible learning techniques.

2 Problem Formulation

We assume the environment to be an MDP $M = \langle S, A, T, r, \gamma \rangle$, in which S is the state space, A is the action space, T is a transition model, r is the immediate reward function, and γ is the discount factor [25]. Although in general an MDP may carry information regarding the reward and discount factors, we consider that this information is inaccessible to the agent during training, and the learning

* Corresponding Author. Email: nathan.schneider_gavenski@kcl.ac.uk

process does not depend on it. Solving an MDP yields a policy π with a probability distribution over actions, giving the probability of taking an action a in state s. We denote the expert policy by π_ψ.

A common self-supervised approach to solving a task via IL uses an Inverse Dynamic Model $\mathcal{M}$. $\mathcal{M}$ uses a set of state transition samples (s_t, s_{t+1}) to predict the action performed in the transitions. By training $\mathcal{M}$ to infer the actions in the state transitions, these approaches can automatically annotate all expert trajectories $\mathcal{T}^{\pi_\psi}$ with actions without the need for human intervention [27, 18, 7]. The agent policy π_θ then uses these *self-supervised* expert-labelled states $(s^{\pi_\psi}, \hat{a})$ to learn to predict the most likely action given a state $P(a \mid s^{\pi_\psi})$. Torabi et al. [27] show that applying an iterative process in self-supervised IL approaches helps π_θ achieve better performance. Initially, $\mathcal{M}$ uses only single transition samples I^{pre} from π_θ and its randomly initialised weights. At each iteration, these approaches use π_θ to create new samples I^{pos} that are used to fine-tune $\mathcal{M}$. However, using all transitions from π_θ makes this iterative approach susceptible to getting stuck in *local minima* due to class imbalance from the I^{pos} data. Monteiro et al. [18] propose a solution that introduces a goal-aware function to sample from all trajectories at each epoch. This function does not require an aligned goal from the environment, hence it is up to the user to choose a desired goal. If π_θ reaches this goal, the trajectory will be used. Finally, it is sensible to assume that $\mathcal{M}$ is not well-tuned during early iterations and predicts mostly wrong labels. Therefore, Gavenski et al. [7] implement an exploration mechanism that uses the softmax distribution of the output as weights to sample actions proportionally to optimality from the model's prediction. As the confidence in the model increases, it predicts suboptimal actions less than the *maximum a posteriori estimation*. By exploring using the model's confidence, their approach can learn under the exploration and exploitation phases, helping $\mathcal{M}$ to converge faster. Nevertheless, creating a handcrafted goal-aware function and using a softmax distribution as an exploration mechanism requires manual intervention and discrete actions. As a result, these methods become unsuitable for more complex environments where goal achievement is non-trivial to check.

3 Continuous Imitation Learning from Observation

We address the need for manual intervention and for maintaining self-supervision in CILO through two key innovations: an exploration mechanism used when the action predictions are uncertain; and a discriminator to interleave random and current states to improve the prediction of self-supervised actions. CILO achieves this by employing three different models: (i) the inverse dynamic model $\mathcal{M}$ to predict the action responsible for a transition between two states $P(a \mid s_t, s_{t+1})$; (ii) a policy model π_θ that uses the self-supervised labels $\hat{a}$ to imitate the expert π_ψ given a state $P(a \mid s_t)$; and (iii) a discriminator model $\mathcal{D}$ to discriminate between π_ψ and π_θ, creating newer samples for $\mathcal{M}$.

Algorithm 1 provides an overview of CILO's learning process. First, CILO initialises all models with random weights and uses the

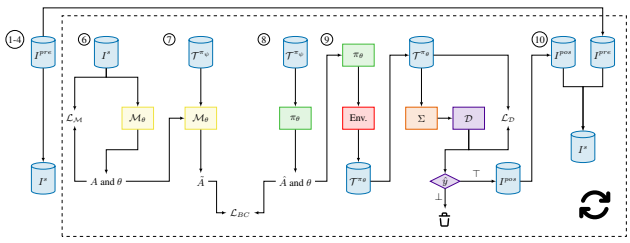


Figure 1. CILO's training cycle.

Algorithm 1 CILO

1: Initialize $\mathcal{M}_\theta$, π_θ, and $\mathcal{D}$ with random weights
2: $I^s \leftarrow I^{pre}$ s.t. $I^{pre} \leftarrow$ samples from π_θ
3: **for** $i \leftarrow 1$ to epochs **do**
4: Improve $\mathcal{M}_\theta$ by TRAINM(I^s)
5: Use $\mathcal{M}_\theta$ with $\mathcal{T}^{\pi_\psi}$ to predict $\hat{A}$
6: Improve π_θ by $\text{error}_{\pi_\theta} \leftarrow$ BEHAVIOURCLONING$(\mathcal{T}^{\pi_\psi}, \hat{A})$
7: Use π_θ to solve environments E
8: $\mathcal{T}^{\pi_\theta} \leftarrow \mathcal{T}^{\pi_\theta} \oplus \{(s_0, \hat{a}_0, s_1), \cdots, (s_{t-1}, \hat{a}_{t-1}, s_t)\}$
9: $I^{pos} \leftarrow I^{pos} \oplus \left\{\forall_{i \in \mathcal{T}^{\pi_\theta}} \mid \mathcal{D}(\beta(\mathcal{T}_i^{\pi_\theta})) \text{ is } \top\right\}$
10: $I^s \leftarrow I^{pre} \oplus I^{pos}$
11: **if** $\text{error}_{\pi_\theta} \leqslant threshold$ **then**
12: Finish training

random initialised policy to collect random samples I^{pre} from the environment (Lines 1-2). The dynamics model uses these random samples to train in a supervised manner (Function TRAINM, Line 4). These samples are vital since they help $\mathcal{M}$ learn how actions cause environmental transitions without expert behaviour-specific knowledge. TRAINM uses the loss from Eq. 1, where θ are the model's current parameters, S is the vector for state representations, A is the action vector representation, and t is the timestep.

$$\mathcal{L}_\mathcal{M}(I^s, \theta) = \sum_{t=1}^{I^s} |\mathcal{M}_\theta(S_t, S_{t+1}) - A_t| \tag{1}$$

With the updated parameters θ, $\mathcal{M}$ predicts the self-supervised labels $\hat{A}$ to all expert transitions ($\mathcal{T}^{\pi_\psi}$ in Line 5). CILO then uses these expert labelled transitions to train π_θ using behaviour cloning (Function BEHAVIOURALCLONING with Eq. 2) coupled with an exploration mechanism (Line 6).

$$\mathcal{L}_{BC}(I^s) = \sum_{(s_t, s_{t+1}) \in I^s} |\mathcal{M}_\theta(s_t, s_{t+1}) - \pi_\theta(s_t)| \tag{2}$$

The policy then generates new samples ($\mathcal{T}^{\pi_\theta}$) that might help $\mathcal{M}$ approximate the unknown ground-truth actions from the expert (Lines 7-8). Given all new samples, CILO generates path signatures β [2] and uses $\mathcal{D}$ to classify signatures as from the expert or the agent. Line 9 updates the discriminator weights with the classification loss in Eq. 3, where $\mathcal{T}_\beta$ are path signatures for all trajectories from expert and agent, C is for the source of the observation (*expert* and *agent*), y is the ground-truth label, and $\hat{y}$ is the source predicted by $\mathcal{D}$.

$$\mathcal{L}_\mathcal{D}(\mathcal{T}_\beta^{\pi_\psi}, \mathcal{T}_\beta^{\pi_\theta}) = -\sum_{i=1}^{|\mathcal{T}_\beta|} \sum_{j=1}^{|C|} y_{ij} log(\hat{y}_{ij}), \tag{3}$$

Samples classified as expert by $\mathcal{D}$ are added to I^{pos} (Line 9), which is then combined with the original I^{pre} to form I^s (Line 10). CILO uses this updated I^s in each iteration for a specified number of epochs (Line 4) or until it no longer improves (Lines 11-12), with an optional hyperparameter $threshold$.

The exploration mechanism allows CILO to deviate from its original action distribution according to the model certainty. This behaviour is helpful during early iterations when $\mathcal{M}$ is unsure about which action might be responsible for a specific transition. Since random samples can be very different from the expert's transitions, we can assume that the model does not learn to recognise these transitions and generalises poorly. Here, we assume that the environment is stochastic, in that multiple actions might occur with a non-zero probability of transition between any pair of states. $\mathcal{D}$'s key objective is to discard trajectories that could result in $\mathcal{M}$ getting stuck in bad local minima, and for instance, stop predicting specific actions

(underfitting). Without a discriminator, it would be difficult to ignore signatures that differ considerably from the ground-truth without an environment-specific hyperparameter, hence reducing the method's generalisability. Finally, combining these mechanisms makes CILO more sample efficient, allowing for oversampling without misrepresenting the action distributions and overfitting. Figure 1 shows the CILO learning cycle in more detail with the different loss functions.

3.1 Exploration

Exploration is vital for IL methods that use dynamics models to learn how the expert behaves. It enables policy divergence when the dynamics model is uncertain and increases state diversity, which helps the model approximate labelled transitions from unlabelled ones (expert). CILO borrows an exploration mechanism from reinforcement learning in continuous domains, in which each action in a policy consists of two outputs: the mean and standard deviation to sample from a Gaussian distribution. However, unlike traditional reinforcement learning, where a policy receives feedback in the form of the reward function, IL lacks this information. Thus, for a model $\mathbb{M}$ and parameters θ ($\mathbb{M}_\theta$), we employ the sampling mechanism in Eq. 4, where $\boldsymbol{\pi}$ is the usual mathematical constant $3.14\ldots$ and ε, as defined in Eq. 5, is used as standard deviation, where a is the ground-truth action (or pseudo-labels from $\mathcal{M}$) and $\hat{a}$ is the action predicted by the model:

$$\tilde{a}_{\mathbb{M}_\theta} = \frac{1}{\varepsilon\sqrt{2\boldsymbol{\pi}}} e^{-\frac{(s_t^e - \mathbb{M}_\theta(S))}{2\varepsilon^2}} \tag{4}$$

$$\varepsilon = \|a - \hat{a}\|^p \tag{5}$$

In Eq. 4, $\mathbb{M}$ is either $\mathcal{M}$ or π, and θ are the parameters of the model updated for the epoch. Notice that when $p = 1$, the model $\mathbb{M}$ uses the absolute value between the predicted and ground-truth labels $\|a - \hat{a}\|$ and this allows for higher exploration.

Observation 1. *If $\mathcal{L}$ is a loss function that monotonically decreases a model's $\mathbb{M}$ error as it approximates the ground-truth function, eventually $\|a - \hat{a}\| < 1$. If we then use $p > 1$ in Eq. 5, ε will exponentially decrease.*

Given all of the above, Eq. 5 offers a trade-off between exploration and exploitation. Since ε is the standard deviation for the exploration function, as the model's predictions get closer to the ground-truth and pseudo-labels, the clusters will have lower variance because the exploration ratio is directly correlated to the model's error.

In Alg. 1, functions TRAINM (ln. 4) and BEHAVIOURCLONING (ln. 6) use this adaptation to adjust the exploration ratio depending on how close the model's predictions are to the ground-truth (or pseudo-labels), in accordance with the standard deviation of the Gaussian distribution. This mechanism also has the benefit of not having to predict information beyond the agent's actions, such as standard deviation, instead obtaining this directly from the model's error. For deterministic behaviour, we can assume that the standard deviation for the model is 0 and use the model's output since sampling from a Gaussian distribution with average x and deviation 0 equals x.

3.2 Goal-aware function

Developing a goal-aware function may not be a trivial task. For environments with a well-defined goal, such as CartPole [1], which defines the goal to be balancing the pole for 195 steps, a goal-identification function could simply classify all trajectories that reach 195 steps as optimal. In this work, we formally define trajectories as:

Definition 1. *A trajectory τ is a finite sequence of states $(s_1, \ldots, s_n)$ where for each $1 \leq i < n$, s_{i+1} is obtained from s_i via the execution of some action. We use the term $(\tau_t^1, \tau_t^2, \ldots, \tau_t^d) \in \mathbb{R}^d$ to denote the particular state s_t ($1 \leqslant t \leqslant n$) within the trajectory τ.*

However, recall that in the context of IL, the agent has no access to the reward signal, and as environments grow in complexity, such a function becomes even harder to encode. By contrast, some environments have no prescribed goal. For example, the Ant environment requires the agent to walk as far as possible without falling, but with no defined cap on the number of time steps [23]. Thus, existing IL approaches [18, 7, 10] often rely on manually defined goal-aware functions, which have the benefit of dispensing with the alignment of the environment's goal. For example, we might define a specific trajectory as required in the Ant environment. Unless the agent reaches all points in this trajectory, our goal-aware function does not classify the episode as successful. However, this creates a degree of unwanted complexity in a learning algorithm and a cumbersome process as the number of environments grows. Yet, trajectories may carry relevant information for CILO since they approximate I^s's samples from $\mathcal{T}^{\pi_\psi}$ [7]. Therefore, CILO tries to classify trajectories that are close to $\mathcal{T}^{\pi_\psi}$ instead of successful ones.

Nevertheless, identifying whether samples are near $\mathcal{T}^{\pi_\psi}$ is also difficult. If we consider a stationary agent, we might discard samples that allow $\mathcal{M}$ to better predict transitions due to their distance to the π_ψ states alone. But, if we consider whole trajectories, it might be difficult to identify middling trajectories needed to close the gap between $\mathcal{T}^{\pi_\theta}$ and $\mathcal{T}^{\pi_\psi}$, and better generalise [8]. Therefore, CILO needs a function that (i) simplifies comparisons between trajectories and (ii) allows $\mathcal{M}$ to receive suboptimal samples.

For the first problem, previous work [19] dealt with the issue of trajectory length by using the average of all states up to a point in time to account for the trajectory changes. Conversely, we use path signatures [2], which are fixed-length feature vectors that are used to represent multi-dimensional time series (i.e., trajectories). A path signature is computed by the function β comprehensively defined in Section 3 of Yang et al. [29], succinctly summarised in the definition below (see Supplementary Material for more detail).[1]

Definition 2. *Let a trajectory τ of a countable length between $[1, n]$ ($n \in \mathbb{N}$), where each state is a vector in $\mathbb{R}^d$ with dimensions indexed by a collection of indices $i_1, \cdots, i_k \in \{1, \cdots, d\}$. Let the recursively computed path signature β for a trajectory τ for any $k \geqslant 1$ and time t ($1 \leqslant t \leqslant n$) be:*

$$\beta(\tau)_{1,t}^{i_1,\cdots,i_k} = \int_{1<s\leqslant t} \beta(\tau)_{1,s}^{i_1,\cdots,i_{k-1}} d\tau_s^{i_k}. \tag{6}$$

Then, the signature of a trajectory $\tau : [1, n] \to \mathbb{R}^d$ is the collection of all the iterated integrals of τ:

$$\begin{aligned}\beta(\tau)_{1,n}^{1,\ldots,i_k} = \Big(1, \beta(\tau)_{1,n}^{1}, \cdots, \beta(\tau)_{1,n}^{d}, \beta(\tau)_{1,n}^{1,1}, \cdots, \\ \beta(\tau)_{1,n}^{1,d}, \beta(\tau)_{1,n}^{2,1}, \ldots, \beta(\tau)_{1,n}^{i_1,i_2,\cdots,i_k}\Big),\end{aligned} \tag{7}$$

where the zero-th term is conventionally equal to 1, and k is defined as the k-th level of the signature, which defines the finite collection of all terms $\beta(\tau)_{1,n}^{i_1,\cdots,i_k}$ for the multi-index of length k. For example, when $k = d$, the last term would be $\beta(\tau)_{1,n}^{d,d,\cdots,d}$.

[1] In our experiments, β (Line 9, Algorithm 1) was computed using the implementation provided by [14].

Path signatures allow CILO to solve the issue of comparing two trajectories and encoding different characteristics that may be relevant when classifying how close a new trajectory is from $\mathcal{T}^{\pi_\psi}$. By using path signatures generated from trajectories in $\mathcal{T}^{\pi_\theta}$ and $\mathcal{T}^{\pi_\psi}$, CILO benefits from: (i) a common signature size, regardless of the original length of trajectories, helping the discriminator not to discriminate against longer trajectories; (ii) independence of environment characteristics embedded in the data (avoiding the need for re–parametrisation for each environment); and (iii) the preservation of the uniqueness of trajectories via the non-linearity of the signatures.

The use of signatures still requires some manual intervention in CILO to define how close a trajectory needs to be before adding it to I^s (i.e., an appropriate similarity threshold). To prevent the need for manually defining this threshold, CILO uses a discriminator model $\mathcal{D}$ to discriminate between π_θ and π_ψ trajectories, which optimises Eq. 3. This yields a non-greedy sampling mechanism by using a model to classify expert and non-expert trajectories.

In summary, CILO's goal-aware function works by computing a signature $\beta(\tau)$ of a trajectory τ and feeds it into the discriminator model $\mathcal{D}$, which classifies whether the source of the trajectory is π_θ or π_ψ. If $\mathcal{D}$ classifies the source of an agent's trajectory as the expert, then CILO appends the trajectory into I^s, helping $\mathcal{M}$ better understand how the transition function T works in the environment.

3.3 Sample efficiency

Besides approximating the expert policy, IL methods focus on efficiently using expert samples. This focus happens since expert samples are hard to obtain. Thus, creating more efficient methods, *i.e.*, that require fewer samples, allows for more useable approaches. Some recent strategies [13, 30] minimise the number of required samples but depend on strong assumptions (see Section 4.2) or manual intervention for each new environment. For comparison, CILO uses 10 expert episodes – a number similar to Zhu et al. [30] and Kidambi et al. [13], but without requiring manual intervention for each environment. CILO relies on up-scaling $\mathcal{T}^{\pi_\psi}$ to increase the number of observations π_θ sees before interacting with the environment. Although trivial, this strategy works because CILO is self-supervised and has an exploration mechanism. This strategy helps in two ways: (i) for each epoch all pseudo-labels differ in all transitions due to the exploration mechanism (Line 4, Algorithm 1); and (ii) increasing the number of samples π_θ receives allows for more updates before sampling new experiences from the environment. By applying its exploration mechanism to each observation individually and sampling exploration values from a distribution, CILO ensures that each observation has unique action values, reducing the risk of misrepresenting the ground-truth action distribution.

4 Experimental Results

We compared CILO's results against three key related methods. Behavioral Cloning from Observations (BCO) [27], which is usually used as a baseline, and two of the most efficient LfO methods: Off-Policy Imitation Learning from Observations (OPOLO) [30], and Model-Based Imitation Learning From Observation Alone (MobILE) [13]. We experimented with five commonly used environments: Ant, Half Cheetah, Hopper, Swimmer, and Pendulum.[2] Each method was run for 50 episodes, with the environment reset when the agent falls or after $1,000$ steps. Each episode was run using random seeds to test the agent's ability to generalise.

4.1 Implementation and Metrics

We used PyTorch to implement our agent and optimise the loss functions in Eq. 1-3 via Adam [15] and Imitation Datasets [9] to collect the expert data. As for the exploration mechanism in Eq. 5, we use $p = 1$ for ε due to all environments actions being in the interval $[-1, 1]$, and using $p > 1$ would significantly diminish the gap between predicted and ground-truth actions (as defined in Definition 1). In the supplementary material, we provide all learning rates and discuss hyperparameter sensitivity in more detail, but we note that CILO is not very sensitive to precise hyperparameters.

We evaluated all approaches using the *Average Episodic Reward* (AER) metric (Eq. 8) and use *Performance* ($\mathcal{P}$) (Eq. 9). AER is the average accumulated reward for a policy π over n number of episodes in t number of steps:

$$AER(\pi) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{t}\gamma^t r(s_{ij}, \pi(s_{ij})). \tag{8}$$

On the other hand, $\mathcal{P}$ normalises between random and expert policies rewards, where performance 0 corresponds to random policy π_ξ performance, and 1 is for expert policy π_ψ performance.

$$\mathcal{P}_\tau(\pi) = \frac{AER(\pi) - AER(\pi_\xi)}{AER(\pi_\psi) - AER(\pi_\xi)} \tag{9}$$

Note that a negative value for $\mathcal{P}$ indicates a reward for the agent lower than a random agent's and a value higher than 1 indicates that the agent's reward is higher than the expert's. All results in Table 1 are the average and standard deviation in five different experiments. We do not report accuracy since achieving high accuracy does not necessarily translate into a high reward for the agent.

4.2 Results

We trained all methods using 10 expert trajectories. Table 1 shows how each method performed in the five environments. CILO had the best overall results in all environments. It consistently achieved results similar to the expert, surpassing it on Ant and Swimmer and achieving the maximum reward for the Pendulum environment. CILO's performance was close to the expert's in Hopper but a little lower in HalfCheetah – likely due to the higher standard deviation from the ground-truth actions in both environments. In the Swimmer environment, BCO and OPOLO achieved AER and performance similar to the expert, while CILO outperformed it by 0.29 points. The same happened in the Ant environment, where CILO surpassed the expert by ≈ 484 reward points. We hypothesise this is due to CILO's explorative nature and its ability to acquire new samples that the discriminator judges to come from the expert.

Comparing CILO to other methods, we see that OPOLO had the closest performance to CILO's in almost all environments. We attribute CILO's better performance than OPOLO's due to the fact that OPOLO's problem formulation assumes that the environment follows an injective MDP, which cannot be guaranteed with random seeds. For this work, we believe that it is more important for an agent to be able to correct its initial states into a successful trajectory than to be optimal in a single setting. Moreover, we notice that for the Pendulum environment, OPOLO only achieved the optimal reward when clipping the actions between $[-1, 1]$, which CILO does not require.

[2] The Supplementary Material briefly describes these environments and the neural networks topology.

Table 1. CILO and baselines AER and $\mathcal{P}$ results for all environments. All results are the average of 50 trajectories.

Algorithm	Metric	Ant	Pendulum	Swimmer	Hopper	HalfCheetah
Random	AER	-65.11 ± 106.16	5.70 ± 3.26	0.73 ± 11.44	17.92 ± 16.02	-293.13 ± 82.12
	$\mathcal{P}$	0	0	0	0	0
Expert	AER	5544.65 ± 76.11	1000 ± 0	259.52 ± 1.92	3589.88 ± 2.43	7561.78 ± 181.41
	$\mathcal{P}$	1	1	1	1	1
CILO	AER	$\mathbf{6091 \pm 801.2}$	$\mathbf{1000 \pm 0}$	$\mathbf{334.6 \pm 3.45}$	$\mathbf{3589 \pm 178.2}$	$\mathbf{7100.6434 \pm 90.1775}$
	$\mathcal{P}$	$\mathbf{1.0974 \pm 0.1372}$	$\mathbf{1 \pm 0}$	$\mathbf{1.2901 \pm 0.0128}$	$\mathbf{0.9998 \pm 0.0487}$	$\mathbf{0.9413 \pm 0.0115}$
OPOLO	AER	5508.6807 ± 930.7590	$\mathbf{1000 \pm 0}$	253.3297 ± 3.4771	3428.6405 ± 420.3285	7004.65 ± 568.66
	$\mathcal{P}$	0.9935 ± 0.1659	$\mathbf{1 \pm 0}$	0.9761 ± 0.0134	0.9549 ± 0.1177	0.9291 ± 0.0724
MobILE	AER	995.5 ± 25.65	111.7 ± 31.25	130.7 ± 24.36	2035 ± 192.95	4721.5 ± 364.5
	$\mathcal{P}$	0.1891 ± 0.0047	0.1066 ± 0.0313	0.5022 ± 0.0968	0.5647 ± 0.0531	0.5647 ± 0.0454
BCO	AER	1529 ± 980.86	521 ± 178.9	257.38 ± 4.28	1845.66 ± 628.41	3881.10 ± 938.81
	$\mathcal{P}$	0.2842 ± 0.1724	0.5675 ± 0.1785	0.9917 ± 0.0166	0.5177 ± 0.1765	0.5117 ± 0.1217

When the actions are not clipped, OPOLO accumulates ≈ 9.55 reward points, a performance similar to the random policy. We opted not to clip CILO's actions, so that the method would not require any previous environment knowledge.

BCO requires more expert trajectories to achieve better results. In its original work, BCO used 5×10^5 samples for $\mathcal{M}$ and more than $1,000$ expert trajectories for its policy, which may be unrealistic for many domains. Nevertheless, BCO achieved almost expert results in the Swimmer environment and higher rewards than MobILE in almost all other environments, with the exception of HalfCheetah. We believe BCO outperforms MobILE because the latter assumes that each environment has a fixed initial state, which does not happen since the gym suite alters each initial state according to some parametrised intervals and its current seed.

As for MobILE, we used the same number of trajectories as in its original work. We observe that MobILE suffers from three different issues: (i) it is ensemble; (ii) has domain knowledge embedded into the algorithm (not publicly available); and (iii) its results are difficult to reproduce, because of the large number of hyperparameters on which they depend. During our experimentation, we observed that some approaches underperform when using an expert with strict movement constraints. To some extent, when obtaining $\mathcal{T}^{\pi_\psi}$, all environments are susceptible to this, but MoBILE was especially impacted. We believe that this strict movement pattern is difficult for all methods to learn since the impact of the variations cannot be immediately perceived. The lack of reproducibility is a major drawback of MobILE, from which CILO does not suffer. By using path signatures, which is a non-parametric encoding technique, CILO is left with only two different parameters: the network size and the learning rate. We used Smith's work (2017) as a guide for finding optimal values for these parameters.[3]

Finally, in environments with broadly distributed expert actions like Ant, Pendulum, and Hopper, CILO matches expert performance in fewer iterations than the other methods. However, in environments where actions are more concentrated (Swimmer and HalfCheetah), CILO takes longer to match the expert.

5 Discussion

In this section, we consider some key aspects of CILO's behaviour: (i) how CILO learns with different sample amounts; (ii) how it approximates predictions to the ground-truth actions of the expert; (iii) how similar each signature becomes to all trajectories over time; (iv) how different action distributions affect CILO; and (v) how I^s behaves over time.

[3] We followed the original network topology for a fair comparison.

5.1 Sample Efficiency

In order to understand CILO's sample efficiency, we experimented with three different amounts of expert episodes in the Ant environment. Ant provides an ideal setting due to its balanced learning complexity and shorter training times. Table 2 shows the AER and $\mathcal{P}$ results using 1, 10, and 100 trajectories. As expected, CILO does not achieve good results when using a single trajectory. This is because π_θ has no information regarding different initialisation and trajectory deviations. This behaviour is intrinsic to behavioural cloning where, without sufficient information, the policy tends not to generalise [16].

Interestingly, CILO achieves 65 *fewer* reward points when using 100 trajectories than when it uses 10. We attribute this to the following: (i) when used in a LfO scenario, BC methods usually fail to scale according to the number of samples due to *compounding error* [26]; and (ii) increasing the number of expert samples decreases the deviation from π_ψ trajectories, resulting in overfitting and a worse π_θ. Since it achieves expert results for almost all environments, we do not consider this behaviour a limitation of CILO. Nevertheless, we hypothesise that using different strategies might result in an increase in performance when its data pool is increased. We also hypothesise that using incomplete or faulty trajectories might help CILO since it would not have so much data for all points in a trajectory, reducing overfit. Fine-tuning the exact number of expert trajectories requires some experimentation for each environment.

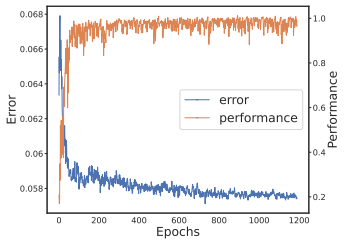
5.2 Ground-truth error over time

A concern for self-supervised IL methods is how to approximate pseudo-labels to ground-truth actions from the expert. However, approximating I^s samples to those from the experts is not always best. There might be samples that are between the experts' and I^{pre} that can help $\mathcal{M}$ smoothly close the gap between equally distributed and the ground-truth distribution [10]. Since CILO uses exploration to learn and this exploration mechanism relies on $\mathcal{M}$'s error, achieving lower error margins early might lead to less exploration and poorer results. It would therefore be better to have a consistent stream of new samples, to maintain the error marginally high but not a significant number of new samples since this could keep $\mathcal{M}$'s error too high or even collapse the network, *i.e.*, updating all weights drastically and requiring a higher learning rate.

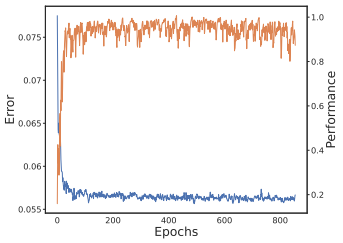
Table 3 shows that using two different procedures and achieving two different policies with different error margins and weights yields

Table 2. CILO's AER and $\mathcal{P}$ values for different Ant dataset sizes.

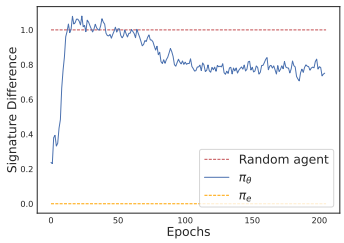
Trajectories	AER	$\mathcal{P}$
1	1003 ± 1999	0.18
10	$\mathbf{6091 \pm 801.2}$	**1.1**
100	6026 ± 725.86	1.09



(a) π_θ trained with a scheduler.



(b) π^*_θ trained with no scheduler.



(c) Signature difference in Ant-v3.

Figure 2. (a) and (b) show ground-truth error for $\mathcal{M}$ and $\mathcal{P}$. (c) shows the normalised difference between π_e, π_θ and random signatures: 0 is equivalent to expert, and $\geqslant 1$ means equal or worse than random policy signature.

similar results for the error margins in both methods but not performance and AER. Figure 2a shows the learning results (error and performance) for π_θ, which uses weight decay and a scheduler during training, and Figure 2b shows π^*_θ, with no weight decay and a learning rate scheduler. We observe that both policies achieve a similarly consistent error margin. However, when comparing the average performances in a single episode, π_θ achieves 29 more reward points with a lower variation. While using different strategies for classifying the action might help CILO with this behaviour, we use a similar topology to the one used by the models compared.

Table 3. $\mathcal{M}$'s ground-truth error and policy's AER and $\mathcal{P}$ for Ant.

Method	Error	AER	$\mathcal{P}$
π_θ	**0.0571**	**5610**	**1.0116**
π^*_θ	0.0556	5581	1.0065

5.3 *Signature approximation over time*

Given Definition 2, trajectories that are similar should be closer in the feature space, while those that do not share any states should be farther apart. Figure 2c shows the Manhattan distance between π_θ and π_ψ trajectories during the first 200 iterations. The difference is normalised between trajectories from random and expert agents. Hence, a difference greater than 1 means that the agent's signature path is farther from π_ψ than a random agent's. As expected, during early iterations, CILO produces episodes that are farther than the random agent since $\mathcal{M}$ has to learn state transitions before π_θ can learn how to behave in the environment. We see similar behaviour from the discriminator $\mathcal{D}$. In the initial iterations, it allows multiple trajectories to be appended to I^s due to its poor performance in discriminating between generated and expert trajectories. Once $\mathcal{D}$ learns to classify correctly, it is only 'fooled' by $\approx 18\%$ of trajectories.

As π_θ increases its performance, the distance between π_θ and π_ψ signatures decreases. Similarly, $\mathcal{D}$ has a harder time distinguishing from expert and π_θ. We observe that $\mathcal{D}$'s results are as expected. By allowing these early trajectories to append into I^s, which had not achieved any goals, it allows $\mathcal{M}$ to learn from samples outside its randomly distributed ones. Since it only allows a few samples, $\mathcal{M}$ does not stop to predict actions due to skewed samples. But as $\mathcal{D}$ improves its classification performance, it forces π_θ indirectly to be closer to the expert behaviour, therefore, achieving higher rewards. Using the gradient signal from $\mathcal{D}$ is likely to improve π_θ's performance further, but this adaptation would require the policy also to predict the next state, *e.g.*, in a mechanism similar to the one used in Edwards et al.'s work [5].

5.4 *Effects of Gaussian exploration*

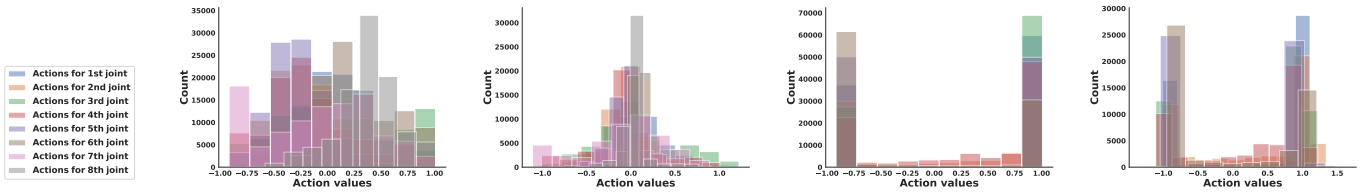
Since we observed that CILO has a different behaviour for environments with different action distributions, we analyse Ant and HalfCheetah to understand the disparities between π_θ and π_ψ action predictions. Figure 3 displays all distributions for 50 trajectories from the expert and trained policies. Note that for these actions, π_θ is not using its exploration mechanism, that is, the policy is greedy. In all environments, the distribution from π_θ actions differs from the expert ones. However, we observe that π_θ actions have a higher intra-cluster variance than the expert ones. We believe this behaviour is due to CILO's exploration mechanism sampling from a Gaussian distribution, making it learn to have a higher variance around the average of an action (considering the error rate from Table 3). Therefore, the exploration mechanism makes it difficult to approximate distributions that do not follow this pattern, such as HalfCheetah.

We also note that CILO has more difficulty achieving better results in environments with sparse action distributions. If we compare Figures 3c and 3d, it is evident that CILO achieves actions near both limits, *i.e.*, -1 and 1; however, it has a harder time predicting actions near the limit. In contrast, although both distributions from Figures 3a and 3b are unequal, we observe a more concentrated action cluster around 0, which helps π_θ achieve better results. We see this behaviour as a limitation of CILO since selecting a new sampling distribution would require knowing beforehand how an expert behaves. However, we also hypothesise that training for a period without exploration and fine-tuning π_θ with $\mathcal{M}$'s pseudo-labels would minimise this impact. Further training π_θ with no exploration, we observe an increase in all environments, although not significantly.

5.5 *I^s size over time*

The use of the discriminator $\mathcal{D}$ allows CILO to start with fewer random samples since it appends samples on almost every iteration. However, increasing I^s on each epoch can create issues if the number of samples grows exponentially. Therefore, we plot in Figure 4 the size of I^s for each environment and epoch for the first 450 epochs. It is important to note that CILO usually reaches expert performance before its first 100 epochs. We observe that for most environments, CILO has a lower slope for appending I^{pos} into its dataset. This behaviour is excellent since it means that CILO is less likely to create data pool sizes that would transform it to be inefficient. Furthermore, when we consider that in Torabi et al.'s work [27], 5×10^5 transitions are needed to learn the inverse dynamic model (≈ 50 epochs), this behaviour allows for less preparation when learning an agent. Nevertheless, Figure 4 also present two other behaviours.

For the InversePendulum environment, CILO gets almost no sample variation when compared to the other methods in the early stages. However, after approximately 150 epochs, π_θ yields trajectories



(a) Ant (π_ψ) (b) Ant (π_θ) (c) HalfCheetah (π_ψ) (d) HalfCheetah (π_θ)

Figure 3. Distribution of expert actions for Ant and HalfCheetah environments.

more similar to the expert, which deteriorates $\mathcal{D}$ accuracy and increases I^s quite substantially. In this environment, we use this behaviour as a form of signal to stop since the reward does not improve. Figure 4 has an inset graph showing the first 150 epochs for the InvertedPendulum environment. In it, we observe during its first epochs, CILO appends samples in a lower rhythm.

For both HalfCheetah and Ant environments, we observe a linear pattern from the samples added into I^s. This behaviour is not desired, resulting in a training procedure that takes around 3 and 1.5 times longer to finish than all the other environments for HalfCheetah and Ant. To mitigate this problem, CILO could implement a forgetting mechanism to get rid of some samples in each epoch either by random selection or using the chronological order of insertion. However, it should not keep its initial sample pool size, considering it has a smaller dataset and changing it could make $\mathcal{M}$ susceptible to covariate shift. We hypothesise that adding samples up until an upper limit would be a better approach, eliminating samples from I^s in each epoch as needed to keep the pool size within the limit.

6 Related Work

The simplest form of imitation learning from observation is Behavioral Cloning (BC) [20], which treats imitation learning as a supervised problem. It uses samples (s_t, a, s_{t+1}) from an expert consisting of a state, action and subsequent state to learn how to approximate the agent's trajectory to the expert's. However, such an approach becomes costly for more complex scenarios, requiring more samples and information about the action effects on the environment. For example, solving Atari requires approximately 100 times more samples than CartPole. Generative Adversarial Imitation Learning (GAIL) [11] solves this issue by matching the state-action frequencies from the agent to those seen in the demonstrations, creating a policy with action distributions that are closer to the expert. GAIL uses adversarial training to discriminate state-actions either from the agent or the expert while minimising the difference between both.

Recent self-supervised approaches [27, 7] that learn from observations use the expert's transitions $(s_t^{\pi_\psi}, s_{t+1}^{\pi_\psi})$ and leverage random transitions (s_t, a, s_{t+1}) to learn the inverse dynamics of the environment, and afterwards generate pseudo-labels for the expert's trajectories. Imitating Latent Policies from Observation (ILPO) [5] differs from such work by trying to estimate the probability of a latent action given a state. Within a limited number of environment steps, it remaps latent actions to corresponding ones. More recently, Off-Policy Learning from Observations (OPOLO) [30] uses a dual-form of the expectation function and an adversarial structure to achieve off-policy LfO. Model-Based Imitation Learning from Observation Alone (MobILE) [13] uses the same adversarial techniques, which rely on an objective discriminator coupled with exploration to diverge from its actions when far from the expert.

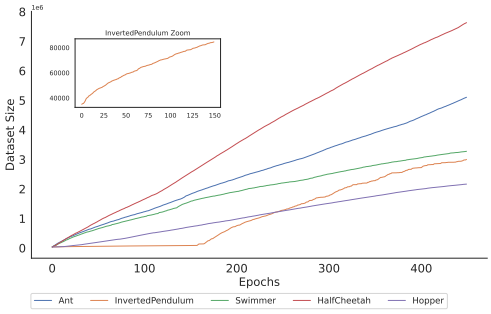


Figure 4. Size of $I^s \times$ epochs for all environments.

7 Conclusions and Future Work

In this paper, we proposed Continuous Imitation Learning from Observation (CILO), a new LfO method combining an exploration mechanism and path signatures. CILO (i) does not require prior domain knowledge or information about the expert's actions; (ii) has sample efficiency superior or equal to the state-of-the-art LfO alternatives; and (iii) approximates (sometimes surpassing) expert performance. CILO achieves these results due to two key contributions. Firstly, the use of a discriminator paired with path signatures, allows CILO to acquire more diverse state transition samples while increasing sample quality. Secondly, the exploration mechanism, which uses the model's error rate to sample from a normal distribution, allows for a more dynamic exploration of the environment. As a result, the exploration ratio decreases as the model learns to approximate from the ground-truth labels. More importantly, these two innovations are completely model-agnostic, allowing them to be used in other IL methods without requiring major changes. We would argue that the innovations we proposed pave the way for IL models that generalise better and require less expert training data.

Our next step is to investigate different exploration mechanisms to better fit the policy needs of specific environments. We would also like to experiment with different forms of adversarial learning to embed CILO's current discriminator into the policy loss function. Considering the path signatures are differentiable, it would be possible to backpropagate the gradients from the discriminator into the policy. This change would allow us to see if a direct signal from the enhanced loss function could improve the action prediction of the inverse dynamic model.

Acknowledgements

This work was supported by UK Research and Innovation [grant number EP/S023356/1], in the UKRI Centre for Doctoral Training in Safe and Trusted Artificial Intelligence (www.safeandtrustedai.org) and made possible via King's Computational Research, Engineering and Technology Environment (CREATE) [4].

References

[1] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, 1(5):834–846, Sep 1983.

[2] I. Chevyrev and A. Kormilitzin. A primer on the signature method in machine learning. *arXiv preprint arXiv:1603.03788*, 2016.

[3] R. Coulom. *Reinforcement learning using neural networks, with applications to motor control*. PhD thesis, Institut National Polytechnique de Grenoble-INPG, 2002.

[4] K. C. L. e Research team. King's computational research, engineering and technology environment (create), 2023. URL https://doi.org/10.18742/rnvf-m076.

[5] A. D. Edwards, H. Sahni, Y. Schroecker, and C. L. Isbell. Imitating latent policies from observation. In *Proceedings of the 36th International Conference on Machine Learning*, ICML 2019, pages 1755–1763. Proceedings of the 36th International Conference on Machine Learning, 2019.

[6] T. Erez, Y. Tassa, and E. Todorov. Infinite horizon model predictive control for nonlinear periodic tasks. *Manuscript under review*, 4, 2011.

[7] N. Gavenski, J. Monteiro, R. Granada, F. Meneguzzi, and R. C. Barros. Imitating unknown policies via exploration. *arXiv preprint arXiv:2008.05660*, 2020.

[8] N. Gavenski, J. Monteiro, A. Medronha, and R. C. Barros. How resilient are imitation learning methods to sub-optimal experts? In J. C. Xavier-Junior and R. A. Rios, editors, *Intelligent Systems*, pages 449–463, Cham, 2022. Springer International Publishing. ISBN 978-3-031-21689-3.

[9] N. Gavenski, O. Rodrigues, and M. Luck. Imitation learning: A survey of learning methods, environments and metrics. *arXiv preprint arXiv:2404.19456*, 2024.

[10] N. S. Gavenski. Self-supervised imitation learning from observation. Master's thesis, Pontifícia Universidade Católica do Rio Grande do Sul, 2021.

[11] J. Ho and S. Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573. Advances in neural information processing systems, 2016.

[12] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys*, 50(2):21:1–21:35, 2017.

[13] R. Kidambi, J. Chang, and W. Sun. Mobile: Model-based imitation learning from observation alone. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 28598–28611. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/f06048518ff8de2035363e00710c6a1d-Paper.pdf.

[14] P. Kidger and T. Lyons. Signatory: differentiable computations of the signature and logsignature transforms, on both CPU and GPU. In *International Conference on Learning Representations*, 2021. https://github.com/patrick-kidger/signatory.

[15] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] H. M. Le and Y. Yue. Imitation learning tutorial, 2018. URL https://sites.google.com/view/icml2018-imitation-learning. ICML Presentation.

[17] T. Lyons. Rough paths, signatures and the modelling of functions on streams. *arXiv preprint arXiv:1405.4537*, 2014.

[18] J. Monteiro, N. Gavenski, R. Granada, F. Meneguzzi, and R. Barros. Augmented behavioral cloning from observation. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.

[19] B. S. Pavse, F. Torabi, J. Hanna, G. Warnell, and P. Stone. RIDM: Reinforced inverse dynamics modeling for learning from a single observed demonstration. *IEEE Robotics and Automation Letters*, 5(4): 6262–6269, oct 2020. doi: 10.1109/lra.2020.3010750.

[20] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Proceedings of the 1st Conference on Neural Information Processing Systems*, NIPS 1988, pages 305–313. Proceedings of the 1st Conference on Neural Information Processing Systems, 1988.

[21] A. Raffin. Rl baselines3 zoo. https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

[22] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

[23] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[24] L. N. Smith. Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pages 464–472. IEEE, 2017.

[25] R. S. Sutton and A. G. Barto. *Reinforcement learning: An Introduction*, volume 2. MIT press Cambridge, 2018.

[26] G. Swamy, S. Choudhury, J. A. Bagnell, and S. Wu. Of moments and matching: A game-theoretic framework for closing the imitation gap. In *International Conference on Machine Learning*, pages 10022–10032. PMLR, 2021.

[27] F. Torabi, G. Warnell, and P. Stone. Behavioral cloning from observation. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, pages 4950–4957, 2018.

[28] P. Wawrzyński. A cat-like robot real-time learning to run. In *International Conference on Adaptive and Natural Computing Algorithms*, pages 380–390. Springer, 2009.

[29] W. Yang, T. Lyons, H. Ni, C. Schmid, and L. Jin. Developing the path signature methodology and its application to landmark-based human action recognition. In *Stochastic Analysis, Filtering, and Stochastic Optimization*, pages 431–464. Springer, 2022.

[30] Z. Zhu, K. Lin, B. Dai, and J. Zhou. Off-policy imitation learning from observations. *Advances in Neural Information Processing Systems*, 33: 12402–12413, 2020.

A Environments and samples

In this work, we experiment with five different environments. We now briefly describe each environment and how the expert samples were gathered. We used Stable Baselines 3 [22] coupled with RL Zoo3 [21] to gather expert samples and its weights loaded from HuggingFaces [4]. We believe this will facilitate reproducibility by allowing future work to use the exact same experts. All expert results are displayed in Table 1 in Section 4.2 of the paper. We used a random sample pool of $50,000$ states for all environments (partitioned into $35,000$ states for training and the remaining $15,000$ for validation). It is important to note, that in each environment, a dimension d of these state vectors $\vec{v}$ represents an internal attribute of the robot. Therefore, although they might share a similar number of dimensions, they may carry different meanings. Since I^s grows in size in each iteration, unlike Torabi et al.'s work [27], CILO does not rely on higher sample pools. Figure 5 shows a frame for each environment.

A.1 A note about the expert samples

During our experiments, we observed that not all experts are created equally. Although most experts trained or loaded from HuggingFace share similar results, the behaviour of each expert varies drastically. One could argue that humans also deviate for each trajectory, but using episodes with a more human-like trajectory (less hectic) yielded better results for all IL approaches. By presenting less hectic and more constant movements, we think each policy receives trajectories that vary more and generalise better. All samples used in this work are available in https://github.com/NathanGavenski/CILO.

A.2 Ant-v2

Ant-v2 consists of a robot ant made out of a torso with four legs attached to it, with each leg having two joints [23]. The goal of this environment is to coordinate the four legs to move the ant to the right of the screen by applying force on the eight joints. Ant-v2 requires eight actions per step, each limited to continuous values between -1 and 1. Its observation space consists of 27 attributes for the x, y and z axis of the 3D robot. We use Stable Baselines 3's TD3 weights. The expert sample contains 10 trajectories, each with $1,000$ states consisting of 111 attributes.[5] Ant-v2 shares distribution behaviour with InvertedPendulum-v2, and Hopper-v2, having action spaced in a bell-curve.

A.3 InvertedPendulum-v2

This environment is based on the CartPole environment from Barto et al. [1]. It involves a cart that can move linearly, with a pole attached to it. The agent can push the cart left or right to balance the pole by applying forces on the cart. The goal of the environment is to prevent the pole from reaching a particular angle on either side. The continuous action space varies between -3 and 3, the only one within the five environments outside of the -1 to 1 limit. Its observation space consists of 4 different attributes. We use Stable Baselines 3's PPO weights. The expert sample size is 10 trajectories, which consist of $10,000$ states (with their 4 attributes) and actions (with a single action value per step). The invertedPendulum-v2 environment is the only one that has an expert with the environment's maximum reward. Therefore achieving $\mathcal{P}$ higher than 1 is impossible.

[4] https://huggingface.co/

[5] With their 111 different attributes - MuJoCo implementation has 27 positions with values and the rest with 0).

A.4 Swimmer-v2

This environment was proposed by [3]. It consists of a robot with s segments ($s \geqslant 3$) and $j = s - 1$ joints. Following [30], in our experiments we use the default setting $s = 3$ and $j = 2$. The agent applies force to the robot's joints, and each action can range from $[-1, 1] \in \mathbb{R}$. A state is encoded by an 8-dimensional vector representing the angle, velocity and angular velocity of all segments. Swimmer distributions present the same distribution of HalfCheetah-v2 (centred around the lower and upper limits). We used Stable Baselines 3's TD3 weights. The expert sample contains 4 trajectories, with $1,000$ states each plus actions for the $j = 2$ joints. The goal of the agent in this environment is to move as fast as possible towards the right by applying torque on the joints and using the fluid's friction.

A.5 Hopper-v2

Hopper-v2 is based on the work done by [6]. Its robot is a one-legged two-dimensional body with four main parts connected by three joints: a torso at the top, a thigh in the middle, a leg at the bottom, and a single foot facing the right. The environment's goal is to make the robot hop and move forward (continuing on the right trajectory). A state consists of 11 attributes representing the z-position, angle, velocity and angular velocity of the robot's three joints. We used Stable Baselines 3's TD3 weights and 10 expert episodes, each with $1,000$ states and actions for the three joints. Each action is limited between $[-1, 1] \in \mathbb{R}$.

A.6 HalfCheetah-v2

HalfCheetah-v2's environment was proposed in [28]. It has a 2-dimensional cheetah-like robot with two "paws". The robot contains 9 segments and 8 joints. Its actions are a vector of 6 dimensions, consisting of the torque applied to the joints to make the cheetah run forward ("thigh", "shin", and "paw" for the front and back parts of the body). All states consist of the robot's position and angles, velocities and angular velocities for its joints and segments. HalfCheetah-v2's goal is to run forward (i.e., to the right of the screen) as fast as possible. A positive reward is allocated based on the distance traversed, and a negative reward is awarded when moving to the left of the screen. We used Stable Baselines 3's TD3 weights. The expert sample size is 10 trajectories, each consisting of $1,000$ states and actions. Each action is limited between the interval of $[-1, 1] \in \mathbb{R}$.

B Network Topology

We followed the same network topologies employed in the original works. Each model ($\mathcal{M}$ and π_θ) are MLP with 4 fully connected layers, each with 512 neurons, with the exception of the last layer whose size is the same as the number of environment actions, Table 4 displays the topologies alongside the input and output sizes of each layer. Following the implementation in [7], we used a self-attention module after the first and second layers. We experimented with normalisation layers during development, which did not increase the agents' results but helped with weight updates. Although we understand that having more complex architectures could increase our method's performance, for consistency we used the same original architecture to show that CILO achieves expert results and does not rely on the architecture. The implementation of our method can be found within https://github.com/NathanGavenski/CILO.

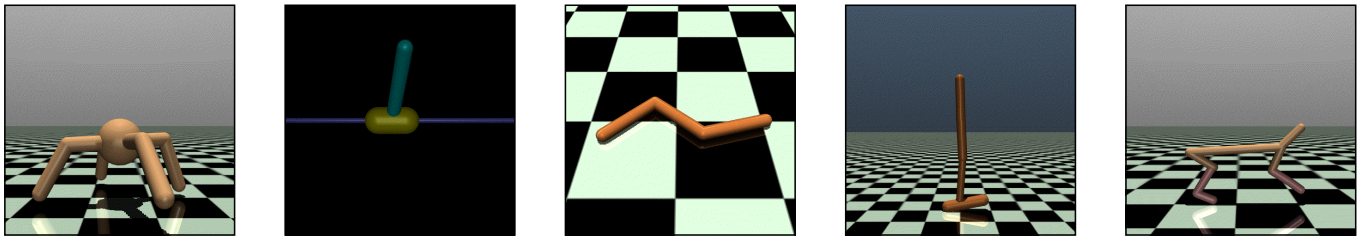


Figure 5. A single frame for each environment used in this work.

Table 4. Layers for each neural network used in this work, where d is the number of dimensions for each state, $\mid a \mid$ is the number of actions, and $\mid \beta \mid$ is given by Eq. 15.

$\mathcal{M}$		π_θ		$\mathcal{D}$	
Layer Name	Input × Output	Layer Name	Input × Output	Layer Name	Input × Output
Input	$2d \times 512$	Input	$d \times 512$	Input	$\mid \beta \mid \times 512$
Activation (Tanh)	-	Activation (Tanh)	-	Activation (Tanh)	-
Fully Connected 1	512×512	Fully Connected 1	512×512	Fully Connected 1	512×512
Activation (Tanh)	-	Activation (Tanh)	-	Activation (Tanh)	-
Self-Attention 1	512×512	Self-Attention 1	512×512	Dropout	0.5%
Fully Connected 2	512×512	Fully Connected 2	512×512	Fully Connected 2	512×512
Activation (Tanh)	-	Activation (Tanh)	-	Activation (Tanh)	-
Self-Attention 2	512×512	Self-Attention 2	512×512	Dropout	0.5%
Fully Connected 3	512×512	Fully Connected 3	512×512	Output	512×2
Activation (Tanh)	-	Activation (Tanh)	-		
Fully Connected 4	512×512	Fully Connected 4	512×512		
Output	$512 \times \mid a \mid$	Output	$512 \times \mid a \mid$		

C Training and Learning Rate

For training, we used a Nvidia A100 40GB GPU and PyTorch. Although we used this GPU, such hardware is not strictly required since CILO uses $\approx$ 2GB to train with a 1024 mini-batch size. The learning rates for $\mathcal{M}$ and π_θ are shown in Table 5. We note that CILO is robust to different learning rates for π_θ. However, $\mathcal{M}$ is more sensitive since I^s changes at almost every iteration, assuming there is at least one agent's trajectory that $\mathcal{D}$ classifies as expert. Having a high learning rate can make $\mathcal{M}$'s weights update too harshly and result in CILO never learning how to label the $\mathcal{T}^{\pi_\psi}$ properly.

Table 5. Different learning rates for $\mathcal{M}$ and π_θ for all environments.

Environment	$\mathcal{M}$	π_θ	Signature k
Ant	1×10^3	1×10^3	2
InvertedPendulum	1×10^3	1×10^3	4
Swimmer	3×10^3	7×10^4	4
Hopper	5×10^3	1×10^3	4
HalfCheetah	1×10^3	7×10^4	4

D Path Signatures

In this work we rely on several path signature definitions to discriminate over agent and expert trajectories. In Section 3.2 of our paper, we briefly defined a trajectory τ, in which each state is a vector $\vec{v}$ in $\mathbb{R}^d$, and how to compute the path signature $\beta(\tau)^{i_1,\cdots,i_k}_{1,n}$, where n is the length of the trajectory, and $i_1, \cdots, i_k \in \{1, \cdots, d\}$ $(k > 1)$ are indices to elements in $\vec{v}$. Here, we provide some additional information on the process of computing a path signature and the intuition behind it.

D.1 Computing the Path Signature

Given a trajectory τ and a function f that interpolates τ into a continuous map $f : \mathbb{R} \to \mathbb{R}$, the integral of the the trajectory against f can be defined as:

$$\int_1^n f(\tau_t) d\tau_t = \int_1^n f(\tau_t)\dot{\tau}_t dt, \tag{10}$$

where $\dot{\tau}_t = \frac{d\tau_t}{dt}$ for any time $t \in [1, n]$. Note that $f(\tau_t)$ is a real-valued path defined on $[1, n]$, which can be considered the integral of a trajectory τ. Moreover, if we consider that $f(\tau_t) = 1$ for all $t \in [1, n]$, then the path integral of f against any trajectory $\tau : [1, n] \to \mathbb{R}$ is simply the increment of τ:

$$\int_1^n d\tau_t = \int_1^n \dot{\tau} dt = \tau_n - \tau_1. \tag{11}$$

Therefore, by assuming that β is a function of real-valued paths, we can define the signature for any single index $i_k \in \{1, \cdots, d\}$ as:

$$\beta(\tau)^{i_k}_{1,n} = \int_{1<s\leqslant n} d\tau_s^{i_k} = \tau_n^{i_k} - \tau_1^{i_k}, \tag{12}$$

which is the increment of the i_k-th dimension of the path. Now, if we move to any pair of indexes $i_k, j_k \in \{1, \cdots, d\}$, we have to consider the double-iterated integral:

$$\beta(\tau)^{i_k,j_k}_{1,n} = \int_{1<s\leqslant n} \beta(\tau)^{i_k}_{1,s} d\tau_s^{j_k} = \int_{1<r\leqslant s\leqslant n} d\tau_t^{i_k} d\tau_s^{j_k}, \tag{13}$$

where $\beta(\tau)^{i_k}_{1,s}$ is given by Eq. 12. Considering that $\beta(\tau)^{i_k,j_k}_{1,n}$ continues to be a real-values path, then we can define recursively the signature function for any number of indexes $k \geqslant 1$ in the collection of indexes $i_1, \cdots, i_k \in \{1, \cdots, d\}$ as:

$$\beta(\tau)^{i_1,\cdots,i_k}_{1,n} = \int_{1<s\leqslant n} \beta(\tau)^{i_1,\cdots,i_{k-1}}_{1,s} d\tau_s^{i_k}, \tag{14}$$

which in our paper is Eq. 6. It is important to note that k is the depth up to which the signature is generated (not its length). At each level $i \leqslant k$, "words" of length i are generated from the alphabet D according to Eq. 6 (main work) to produce the terms of the signature. Figure 6c shows all possible terms for a trajectory $\tau : [1, n] \to \mathbb{R}^d$ for different depths. For example, a signature with depth 2 will have all the terms in the levels $i = 0, 1, 2$. In an alphabet with d letters, we can construct one word of length 0, d words of length 1, and d^2 words of length 2, giving $1 + d + d^2$ words in total (i.e., the number of terms in the signature). In general, the length of a signature with alphabet size d and depth k is:

$$\sum_{i=0}^{k} d^i = \frac{d^{k+1} - 1}{d - 1}. \tag{15}$$

We observe that signatures can be computed for any depth k, and are not restricted to $k \leqslant d$.

We give two examples to illustrate how the terms in a signature are computed (we omit the level 0 whose single value 1 is fixed). Figure 6a shows how to generate a signature of depth 1 (with the terms in the first and second columns of Figure 6c). Considering that the length of a signature grows exponentially with the depth k desired ($\frac{d^{k+1}-1}{d-1}$), Figure 13 only shows how to calculate the terms of a signature of depth 2 for a 2-dimensional dictionary, with the first index fixed in 2.

D.2 A Numerical Example

Let us consider a trajectory τ with two two-dimensional states $\{\tau_t^1, \tau_t^2\}$, and the set of multi-indexes $W = \{ (i_1, \cdots, i_k) \mid k \geqslant 1, i_1, \cdots, i_k \in \{1, 2\} \}$, which is the set of all finite sequences of 1's and 2's. Given the trajectory $\tau : [1, 10] \to \mathbb{R}^2$ illustrated in Figure 7, where the path function for τ is computed according to the function:

$$\tau_t = \{\tau_t^1, \tau_t^2\} = \{5 + t, (5 + t)^2 \mid t \in [1, 10]\} \tag{16}$$

For the depth k desired, the computation of the signature would be computed as shown in Figure 8. For example, given that states in τ are two-dimensional ($d = 2$), the path signature for τ with depth $k = 2$ will have the $\frac{d^{k+1}-1}{d-1} = \frac{2^3-1}{1} = 7$ terms in the vector $\beta(\tau)_{1,10} = [\, 1, 9, 189, 40.5, 970.5, 730.5, 17860.5 \,]$.

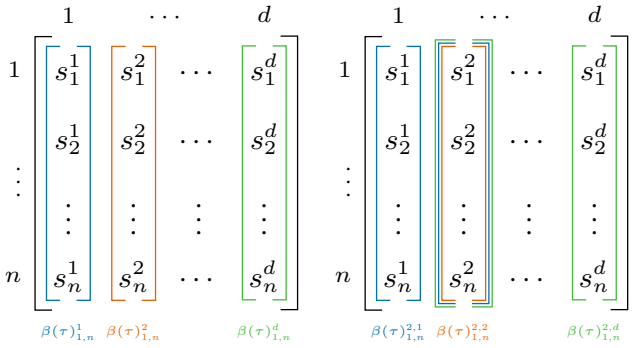
D.3 Signature Properties

We now describe properties of path signatures that are most relevant to our work. The description is not comprehensive. We recommend the work from Yang et al. [29] and Chevyrev and Kormilitzin [2] for a more in-depth approach to path signatures.

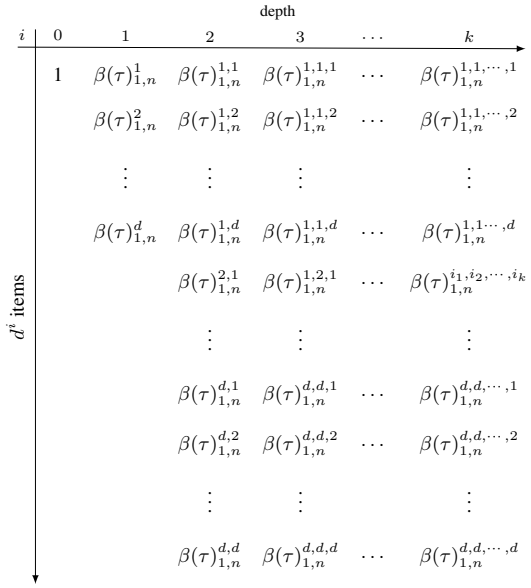
Uniqueness: This property relates to the fact that no two trajectories τ and τ' of bounded variation have the same signature unless the trajectories are tree-equivalent. In light of the invariance under reparametrisations [17], we note that path signatures have no tree-like sections to monotone dimensions, such as acceleration.

Generic nonlinearity of the signature: The second property refers to the product of two terms $\beta(\tau)^{i_1,\cdots,i_k}$ and $\beta(\tau)^{j_1,\cdots,j_k}$, which can also be expressed as:

$$\begin{aligned} \beta(\tau)^1_{1,n} \cdot \beta(\tau)^2_{1,n} &= \beta(\tau)^{1,2}_{1,n} + \beta(\tau)^{2,1}_{1,n} \text{ or,} \\ \beta(\tau)^{1,2}_{1,n} \cdot \beta(\tau)^1_{1,n} &= \beta(\tau)^{1,1,2}_{1,n} + \beta(\tau)^{1,2,1}_{1,n}. \end{aligned} \tag{17}$$



(a) $\beta(\tau)^i_{1,n} \mid \forall_i \in \{1, \cdots, d\}$. (b) $\beta(\tau)^{2,j}_{1,n} \mid \forall_j \in \{1, \cdots, d\}$.



(c) Collection of signatures for τ, where $k \in [0, \infty)$.

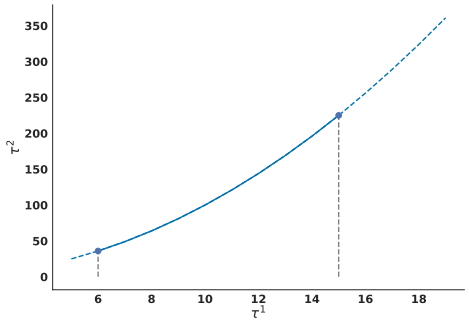


Figure 7. Trajectory $\tau : [1, 10] \to \mathbb{R}^2$.

Thus, the nonlinearity of the signature in terms of low-level terms can be expressed by the linear combination of higher-level terms, which adds more nonlinear previous knowledge to the feature vector. This behaviour is better exemplified in the second level of signatures where for any $\beta(\tau)^{i_k,i_k}_{1,n}$, the result will be $\left(\tau_n^{i_k} - \tau_1^{i_k}\right)^2/2$.

Fixed dimension under length variations: The last property refers to the path signature's length invariance under different trajectory lengths. In Section D.1, we showed that the signature length is a function of the signature depth (k) and the number of dimensions in a state (d). Therefore, path signatures become practical feature vectors for trajectories in machine learning tasks, requiring different inputs

$k = 0$

$$\beta(\tau)_{1,n} = 1$$

$k = 1$ – Eq. 3

$$\beta(\tau)^1_{1,10} = \int_{1<t\leqslant 10} dt = \tau^1_{10} - \tau^1_1 = 9$$

$$\beta(\tau)^1_{1,10} = \int_{1<t\leqslant 10} dt = \tau^2_{10} - \tau^2_1 = 189$$

$k = 2$ – Eq. 4

$$\beta(\tau)^{1,1}_{1,10} = \iint_{1<t_1\leqslant t_2\leqslant 10} d\tau^1_{t_1}, d\tau^1_{t_2} = \int_1^{10}\left[\int_1^{t_2} dt_1\right] dt_2 = 40.5$$

$$\beta(\tau)^{1,2}_{1,10} = \iint_{1<t_1\leqslant t_2\leqslant 10} d\tau^1_{t_1}, d\tau^2_{t_2} = \int_1^{10}\left[\int_1^{t_2} dt_1\right] 2(5+t_2)dt_2 = 970.5$$

$$\beta(\tau)^{2,1}_{1,10} = \iint_{1<t_1\leqslant t_2\leqslant 10} d\tau^2_{t_1}, d\tau^1_{t_2} = \int_1^{10}\left[\int_1^{t_2} 2(5+t)dt_1\right] dt_2 = 730.5$$

$$\beta(\tau)^{2,2}_{1,10} = \iint_{1<t_1\leqslant t_2\leqslant 10} d\tau^2_{t_1}, d\tau^2_{t_2} = \int_1^{10}\left[\int_1^{t_2} 2(5+t)dt_1\right] 2(5+t)dt_1 = 17,860.5$$

$k \geqslant 3$ – Eq. 5

$$\beta(\tau)^{1,1,1}_{1,10} = \iiint_{1<t_1\leqslant t_2\leqslant t_3\leqslant 10} d\tau^1_{t_1} d\tau^1_{t_2} d\tau^1_{t_3} = \int_1^{t_1}\left[\int_1^{t_2}\left[\int_1^{t_3} dt_1\right] dt_2\right] dt_3 = 121.5$$

$$\vdots$$

Figure 8. Step-by-step computation of a path signature

to share the same size without recurrent neural networks.

D.4 Motivation for Signatures

Given the nature of deep learning methods operating on vectorial data, which requires the input data to be of a predetermined fixed length, many techniques, such as word embeddings (where a word is represented by a vector), are used to circumvent this length requirement. Moreover, imitation learning tasks, by definition, have to effectively represent expert demonstrations to capture relevant information for learning a desired behaviour. Path signatures provide a solution to represent sequential or trajectory-based expert demonstration in a principled and efficient manner.

In imitation learning, expert demonstration often takes the form of trajectories or sequences of states over time. Path signatures offer a way to encode these trajectories into high-dimensional feature representations that capture the expert behaviour in a geometric and analytic way. Furthermore, the uniqueness property ensures that essential information about the expert demonstrations is preserved in the path signature representation, enabling accurate discrimination over different trajectories' signatures. Lastly, path signatures provide a single hyperparameter (the number of desired collections k). By adjusting k, we can control the trade-off between representational quality and computational complexity, allowing for efficient learning and generalisation. However, we observe that increasing k leads to an exponential increase in the length of the signature, which imposes a limit to agents with limited computation resources.

D.5 Signature Time Complexity

We now briefly discuss the upper-bound complexity for computing path signatures and compare it to Pavse et al.'s work [19], which computes the averages of the trajectory states. Given Eq. 14 and Fig. 6, it should be easy to see that path signatures can be computed in time $\mathcal{O}(t \cdot d^k)$, where t is the number of samples in a trajectory, d is the number of dimensions, and k is the depth/length of the signature. In contrast, Pavse et al.'s work [19] uses the average over the current and previous states. This does not work well when different trajectories average to the same value, but it is not an issue for signatures due to their uniqueness. Using averages is not an issue in Pavse et al.'s work (or in IRL in general), which computes an artificial reward signal at each timestep. However, this also quickly becomes costly because trajectories are traversed multiple times since the method computes the average of all sub-trajectories t times at each epoch ($\mathcal{O}(d \cdot t^3)$). The cost of computing signatures increases linearly with respect to the episode length, whereas the cost of computing the averages increases exponentially for Pavse et al.. Moreover, path signatures increase exponentially according to the number of dimensions d, which is constant for all environments. Therefore, the main parameter CILO has to be concerned about is the depth k, for which Fig. 9 provides a comparison in Ant-v2 — the environment with the largest state representation. Fig. 9 shows that the cost of computing signatures is lower than that of computing averages for $k \leqslant 5$ in episodes containing $1,000$ timesteps (the total length for all MuJoCo environments used in this work).

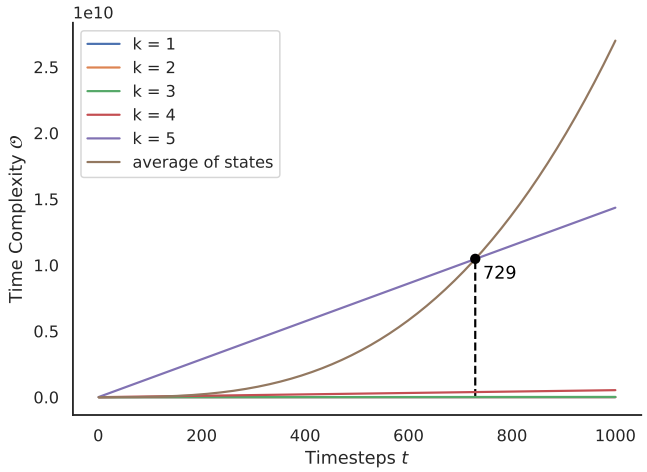


Figure 9. Upper-bound time complexity for signature computation and average.