Value-Aligned Imitation via Focused Satisficing

Rushit N. Shah[*1] **Nikolaos Agadakos**[*1] **Synthia Sasulski**[1]

Ali Farajzadeh[1] **Sanjiban Choudhury**[2] **Brian D. Ziebart**[1]

[1]Department of Computer Science, University of Illinois at Chicago
{rshah231,nagada2,lsasu2,afaraj5,bziebart}@uic.edu
[2]Department of Computer Science, Cornell University
sanjibanc@cornell.edu

Abstract

According to *satisficing theory*, humans often choose *acceptable* behavior based on their personal *aspirations*, rather than achieving (near-) optimality. For example, a `lunar lander` demonstration that successfully lands without crashing might be acceptable to a novice despite being slow or jerky. When human aspirations are much lower than autonomous system capabilities, this can allow learned policies that sufficiently satisfy differing human objectives. Maximizing the likelihood of demonstrator satisfaction also provides guidance for learning under competing objectives that are difficult for existing imitation learning methods to resolve. Using a margin-based objective to guide deep reinforcement learning, our **focused satisficing** approach to imitation learning seeks a policy that surpasses the demonstrator's aspiration levels—defined over trajectories—on unseen demonstrations *without explicitly learning those aspirations*. We show experimentally that this focuses the policy to imitate higher quality demonstrations better than existing imitation learning methods, providing much higher rates of guaranteed acceptability to the demonstrator, and competitive true returns on a range of environments.

1 Introduction

When faced with challenging decision tasks, *satisficing theory* [19] suggests that demonstrators produce behavior that is *acceptable* rather than (near) optimal. By viewing imitation learning through this lens, we aim for imitator behavior that is similarly *acceptable* to the demonstrator, despite never knowing the demonstrator's precise acceptability criteria (Figure 1, left)—working instead with an assumed class of cost functions that defines it. To pursue this aim, we develop **Min**imally **Sub**dominant **F**ocused **I**mitation (**MinSubFI**), which employs the subdominance [26], a margin-based measure of *insufficiency* (i.e., the distance from guaranteeing imitator-acceptability by a margin), as a training objective for policy gradient optimization (Figure 1, right). This produces policies that are maximally acceptable rather than reward-maximizing. Compared to existing inverse reward learning methods [5, 7, 23, 24, 8, 25], which are highly reliant on an estimated scalar reward function to guide reinforcement learning (e.g., using the pipeline of engineered components in Figure 2, top), our approach more directly optimizes the imitator's policy, enabling it to: **(1)** Learn context-sensitive policies without learning context-sensitive cost functions; **(2)** Ignore less optimal demonstrations without requiring explicit noise modeling; and **(3)** Provide generalization guarantees for changing acceptability (e.g., due to skill improvement or fatigue).

Under our *satisficing theory* perspective of imitation learning, policies are learned from demonstrations that are *acceptable*, according to an unknown acceptability set, rather than *near-optimal*. We

[*]Equal contribution

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

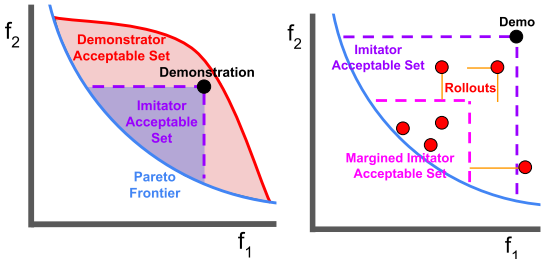

Figure 1: Left: Pareto-dominating in the cost function bases (f_1, f_2) of acceptable behavior (purple: *imitator acceptable set*) guarantees the imitator is acceptable to the demonstrator (red: *demonstrator acceptable set*). Right: The subdominance (orange lines) measures how far imitator trajectory rollouts are from guaranteeing acceptance (by a margin).

define this notion of acceptability and develop new imitation learning methods that are designed to be performant with respect to the demonstrators' unknown acceptability sets in both theory and practice.

1.1 Imitation Learning Problem Setting & Satisficing Perspective of Demonstrations

We consider the imitation learning [15] task of producing a policy $\hat{\pi}$ based on demonstrated trajectories of states and actions, $\tilde{\xi} = (\tilde{s}_1, \tilde{a}_1, \tilde{s}_2, \ldots, \tilde{s}_T)$. Demonstrations are produced from a task-indexed Markov decision process (MDP), $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \{\tau_i\}, C)$, characterized by states $\mathcal{S}$, actions $\mathcal{A}$, state transition probability distributions $\tau_i : \mathcal{S} \times \mathcal{A} \to \Delta_{\mathcal{S}}$ (with Δ representing a probability simplex), and an (unknown) cost function $C : \mathcal{S} \to \mathbb{R}_{\geq 0}$. We use $\tilde{\xi}_{i,j}$ to denote the j$^{\text{th}}$ demonstration for the i$^{\text{th}}$ task, $\tilde{\Xi}$ to denote the set of all demonstrations, and $\tilde{\Xi}_i$ to denote the set of demonstrations of task i.

According to satisficing theory [19], when faced with challenging decision tasks, humans tend to prioritize behaviors that are personally acceptable, rather than striving for optimality.

Definition 1. Trajectory ξ **satisfices** (or is **acceptable**) for a particular **aspiration**, defined by $(\mathbf{w}, \nu)$ if and only if it is less costly than the aspirational threshold ν evaluated using the cost function parameterized by $\mathbf{w}$: $\text{cost}_{\mathbf{w}}(\xi) < \nu$. It **satisfices** the **aspiration/acceptability set** $\Omega = \{(\mathbf{w}, \nu)\}$, i.e., $\xi \in \text{Satisf}_{\Omega}$, if and only if ξ satisfices each aspiration in Ω.

Note that the aspiration set can be context-dependent and vary for each demonstration. For example, it may change with the growing experience (or fatigue) of the demonstrator, or based on available side information (e.g., the weather conditions when controlling a vehicle). Aspiration sets—and their relationships to available contextual information—are generally unknown. Our aim is not to learn them explicitly. Instead, we seek a policy that produces trajectories $\xi \sim \pi \times \tau$, with **maximal probability of acceptance**, $P(\xi \in \text{Satisf}_{\tilde{\xi}})$, for $\tilde{\xi}$'s implicit satisfaction set. Further, in Appendix A we show that existing imitation learning methods **do not provide acceptability guarantees** with respect to the (unknown) acceptability sets of demonstrations.

1.2 Subdominance Minimization and Satisficing

The subdominance measures how far trajectory ξ is from Pareto-dominating a demonstrated trajectory $\tilde{\xi}$ by a margin (Figure 1, right) and has been previously employed for inverse optimal control [26]:

$$\text{subdom}_{\boldsymbol{\alpha}}(\xi, \tilde{\Xi}) = \frac{1}{|\tilde{\Xi}|} \sum_{\tilde{\xi} \in \tilde{\Xi}} \underbrace{\sum_{k} \overbrace{\left[\alpha_k (f_k(\xi) - f_k(\tilde{\xi})) + 1\right]_+}^{(\text{feature } k)\ \text{subdom}^{k}_{\alpha_k}(\xi, \tilde{\xi})}}_{(\text{aggregated})\ \text{subdom}_{\boldsymbol{\alpha}}(\xi, \tilde{\xi})}, \tag{1}$$

with $[x]_+ \triangleq \max(x, 0)$ as the hinge function, and trajectory cost features $\mathbf{f} : \Xi \to \mathbb{R}^K_{\geq 0}$. See Appendix B for further details. Importantly, minimizing the subdominance to zero guarantees that the imitator's behavior is acceptable to the demonstrator.

Our objective is to minimize the subdominance by finely optimizing over a flexible class of policies. To generalize to unseen data, we additionally seek a margin of improvement over the demonstrator, i.e., subdom_{α}, throughout our formulation. With this added margin, the subdominance is a convex function (in trajectory features) that upper bounds the $\text{Satisf}_{\tilde{\xi}}$ non-membership, measuring how far the trajectory is from being guaranteed to satisfy the demonstrator's aspirations by a margin.

Definition 2. The minimally subdominant stochastic policy $\pi_{\boldsymbol{\theta}} : \mathcal{S} \to \Delta_{\mathcal{A}}$ minimizes the expected subdominance of the minimum cost trajectory, $\xi^*(\pi_{\boldsymbol{\theta}})$ induced by the weights θ of policy π, with

respect to the set of demonstration trajectories $\tilde{\xi}_i$ using hinge slopes $\boldsymbol{\alpha}$:

$$\min_{\boldsymbol{\theta}} \min_{\boldsymbol{\alpha} \succeq 0} \sum_{\text{task } i} \frac{|\tilde{\Xi}_i|}{|\tilde{\Xi}|} \mathbb{E}_{\xi \sim \pi \times \tau} \Big[\text{subdom}_{\boldsymbol{\alpha}}(\pi_{\boldsymbol{\theta}}, \tilde{\Xi}_i) \Big] + \frac{\lambda_\alpha}{2} ||\boldsymbol{\alpha}|| + \frac{\lambda_\theta}{2} ||\boldsymbol{\theta}||. \tag{2}$$

This optimization seeks hinge loss slopes $\boldsymbol{\alpha}$ and a policy $\pi_{\boldsymbol{\theta}}$ that both minimize the subdominance. Further details of the optimization of $\boldsymbol{\theta}$ and $\boldsymbol{\alpha}$ are provided in Appendices C and G, respectively.

1.3 Learning a Cost Feature Representation

Though shaping the imitator's behavior from demonstrations is much less dependent on a highly-expressive cost model/features under our approach, hand-engineering features can still be a significant burden in many domains. To mitigate this, we learn a set of cost features $\mathbf{f}_\psi$ from pairwise preferences over demonstrations (juxtaposition with our method and TREX is presented in Appendix F).

Definition 3. Given pairwise preferences over demonstrations $\tilde{\mathcal{D}} = \{\tilde{\xi}_i \prec \tilde{\xi}_j | \tilde{\xi}_i, \tilde{\xi}_j \in \tilde{\Xi}\}$, and a sufficiently-rich function class $\mathcal{F}$, a preference-preserving (latent) representation $\mathbf{f}_\psi : \mathcal{S} \to \mathbb{R}^{K'}_{\geq 0}$ (of dimensionality K') can be learned by minimizing: $\arg\min_{\mathbf{f}_\psi \in \mathcal{F}} \mathbb{E}_{(\tilde{\xi}_i \prec \tilde{\xi}_j) \sim \tilde{\mathcal{D}}} \left[-\log \frac{e^{c_{i,j}}}{e^{c_{i,j}} + e^{c_{j,i}}} \right]$, where $c_{i,j} = \text{subdom}_\alpha(\mathbf{f}_\psi(\tilde{\xi}_i), \mathbf{f}_\psi(\tilde{\xi}_j))$.

2 Experiments

2.1 Demonstrations, Baseline Methods, and Training Details

We conduct experiments using a mix of simple, classic control environments (`cartpole`, `lunarlander`) and complex robotics environments (Mujoco `hopper`, `halfcheetah`, `walker`) from OpenAI Gym [4]. For each environment, we obtain 200 (mostly suboptimal) demonstrations (100 for training, 100 held-out for evaluation) from a suboptimal, RL PPO policy. Methodology and detailed demonstration statistics are provided in Table 3 in the Appendix. The baselines we employ are: behavior cloning (BC), generative adversarial imitation learning (GAIL) [11], adversarial inverse reinforcement learning (AIRL) [10], an unaltered version of T-REX [5], and a modified version of TREX with a cost function C that is a linear combination of cost features $\mathbf{f}$ and cost function weights $\hat{w}$, rather than as a function mapping from the observation vector $\boldsymbol{\phi}$ to cost C (abbreviated TREX_{CF}).

Using Algorithm 1 and analytically computed α values in step 4, we train all variants of our MinSubFI policy with a behavior-cloned policy initialization; the non-random policy initialization is motivated by the sample efficiency it provides (Figure 3 in Appendix H). We train two MinSubFI models: an offline version using only demonstration samples without environment interactions ($\text{MinSubFI}_{\text{OFF}}$), and an online version ($\text{MinSubFI}_{\text{ON}}$). We additionally train $\text{MinSubFI}_{\text{LCF}}$ an online subdominance minimizer with a learned cost feature space of $K' = 3$ dimensions via Definition 3. Architecture details are provided in Appendix F. Cost features are environment-specific and are provided in Table 4 in Appendix J.2, and we employ a quadratic expansion of these during training.

Table 1: Relative γ-satisficing values of different versions of MinSubFI, **on held out demonstrations**, on **basic cost features** (values greater than 1 are formatted in **bold** and the best of each environment is colored green).

Environment	BC	TREX	TREX_{CF}	AIRL	GAIL	$\text{MinSubFI}_{\text{OFF}}$	$\text{MinSubFI}_{\text{ON}}$	$\text{MinSubFI}_{\text{LCF}}$
`cartpole`	0.19	0.04	0.00	0.09	**2.24**	**2.62**	**2.53**	**1.99**
`lunarlander`	0.00	0.00	0.00	0.02	0.00	0.49	0.38	**1.72**
`hopper`	0.00	0.00	0.00	0.02	**6.40**	0.86	**1.69**	**1.99**
`halfcheetah`	0.00	0.00	0.00	**1.12**	**3.99**	**1.93**	**1.80**	0.87
`walker2d`	**8.73**	0.00	0.00	0.00	0.00	**2.15**	0.46	**1.97**

2.2 Demonstrator Acceptability Analysis

In Table 1, we evaluate the rate that the imitator satisfices demonstrations (Definition 8), guaranteeing demonstrator satisfaction, relative to the rate that a randomly chosen demonstration satisfices other

demonstrations, $\gamma_{\text{rel}} \triangleq P(\xi \in \Omega_{\tilde{\xi}})/P(\tilde{\xi}' \in \Omega_{\tilde{\xi}})$, using trajectory-level cost features. Imitation learning methods designed to minimize predictive losses (BC) or learned cost functions (TREX) produce trajectories with very different cost features than demonstrations, leading to small values in this analysis (with a few exceptions, e.g., BC on `walker2d`). Reinforcement learning using an estimated cost function often focuses too narrowly on minimizing a small subset of cost features at the expense of ignoring one or more other features, allowing them to take unacceptable values. For example, though TREX produces `cartpole` policies keeping the pole upright (near optimally), it does so with much larger amounts of horizontal motion than demonstrations exhibit, making it potentially unacceptable to the demonstrator. Despite imitating suboptimal demonstrations, GAIL surprisingly achieves high acceptability rates on some environments, although in some cases it does so at the cost of lower true returns (e.g., on `hopper` in Table 2).

In contrast, since MinSubFI minimizes an upper bound on the imitator's satisficing value, it consistently guarantees demonstrator acceptability much more frequently, with the exception of `lunarlander`, which all methods struggle with due to the difficulty of optimizing some of its sparse cost features. We find that online subdominance minimization tends to provide more frequent acceptability guarantees than the offline variant.

In addition, though $\text{MinSubFI}_{\text{LCF}}$ learns its own space of cost features, it still provides large rates of guaranteed demonstrator acceptance in the original, provided cost feature space for most environments. This suggests that knowing the demonstrator's cost feature space is unnecessary for providing demonstrator-acceptable behavior.

Table 2: Mean (and standard deviation) of the true episode returns of the **held out demonstrations** and trajectories sampled from different imitation learning methods' learned policies.

		Baselines					Ours		
Environment	Demonstrations	BC	TREX	TREX_{CF}	AIRL	GAIL	$\text{MinSubFI}_{\text{OFF}}$	$\text{MinSubFI}_{\text{ON}}$	$\text{MinSubFI}_{\text{LCF}}$
`cartpole`	116 (74)	70 (37)	199 (0.1)	199 (0.1)	15 (4)	**200 (0.0)**	**200 (0.1)**	199 (1)	**200 (0.0)**
`lunarlander`	113 (132)	164 (27)	-171 (3)	195 (7)	-416 (30)	256 (9)	**268 (0.5)**	**268 (0.9)**	-562 (227)
`hopper`	858 (884)	671 (80)	1335 (15)	**2657 (28)**	11 (4)	601 (30)	570 (33)	1433 (146)	2001 (92)
`halfcheetah`	686 (584)	1283 (53)	1017 (7)	1535 (49)	768 (47)	1595 (4)	**1626 (10)**	1582 (15)	890 (332)
`walker2d`	891 (1141)	526 (99)	20 (0.0)	90 (5)	-3 (0.1)	489 (82)	1461 (449)	2306 (391)	**2374 (135)**

2.3 True Returns Using Full Demonstration Set

Though MinSubFI seeks to achieve demonstrator acceptability for all cost functions defined by its cost features (Table 1), it also provides improvements over demonstrations in terms of true return when the true return can be (approximately) defined by the cost features. In Table 2, we evaluate the true returns of the demonstrations and each of the imitation learning methods averaged over ten random seeds. Behavioral Cloning (BC) and AIRL often underperform relative to the demonstrations (except for `halfcheetah`), while the relative performance for TREX and GAIL is more mixed. In contrast, the various forms of MinSubFI tend to consistently outperform the demonstrations with few exceptions (e.g., $\text{MinSubFI}_{\text{OFF}}$ on `walker2d`). Different variants of MinSubFI provide the highest returns except for `hopper`, in which TREX_{CF} provides the largest returns. Interestingly, TREX benefits from using the cost features as the basis for its cost function estimate (TREX_{CF}) rather than using the learned scalar reward of the original formulation (TREX).

3 Conclusions & Future Work

In this paper, we reformulated imitation learning using satisficing theory, defining a new objective for imitation learning: producing policies that are maximally acceptable to (dynamic) demonstrators *without* explicitly learning demonstrator notions of acceptability. We introduced MinSubFI, a policy gradient approach for imitation learning that minimizes policy subdominance to ensure the imitator's behavior aligns with the demonstrator's expectations. We demonstrated its effectiveness in both online and offline learning scenarios, highlighting its robust alignment compared to existing methods. Future work includes improving feature representation learning, integrating additional information from demonstrators about their notions of acceptability, and developing methods to enhance demonstrator acceptability in dynamic and subjective contexts.

References

[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pages 1–8, 2004.

[2] Stuart Armstrong, Jan Leike, Laurent Orseau, and Shane Legg. Pitfalls of learning a reward function online. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2021.

[3] Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[5] Daniel Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *International Conference on Machine Learning*, pages 783–792. PMLR, 2019.

[6] Daniel S. Brown, Wonjoon Goo, and Scott Niekum. Better-than-demonstrator imitation learning via automatically-ranked demonstrations. In *Proceedings of the Conference on Robot Learning*, pages 330–359, 2020.

[7] Benjamin Burchfiel, Carlo Tomasi, and Ronald Parr. Distance minimization for reward learning from scored trajectories. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), 2016.

[8] Letian Chen, Rohan Paleja, and Matthew Gombolay. Learning from suboptimal demonstration via self-supervised reward regression. *arXiv preprint arXiv:2010.11723*, 2020.

[9] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in Neural Information Processing Systems*, 30, 2017.

[10] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adverserial inverse reinforcement learning. In *International Conference on Learning Representations*, 2018.

[11] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

[12] Rudolf E. Kalman. When is a linear control system optimal? *Trans ASME, J. Basic Eng.*, pages 51–60, 1964.

[13] Omid Memarrast, Linh Vu, and Brian D Ziebart. Superhuman fairness. In *Proceedings of the International Conference on Machine Learning*, volume 202, pages 24420–24435. PMLR, 23–29 Jul 2023.

[14] Andrew Y Ng and Stuart J Russell. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, pages 663–670, 2000.

[15] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, and Jan Peters. An algorithmic perspective on imitation learning. *arXiv preprint arXiv:1811.06711*, 2018.

[16] Dean A. Pomerleau. Efficient Training of Artificial Neural Networks for Autonomous Navigation. *Neural Computation*, 3(1):88–97, 03 1991.

[17] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[18] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[19] Herbert A Simon. Rational choice and the structure of the environment. *Psychological review*, 63(2):129, 1956.

[20] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, 12, 1999.

[21] Umar Syed and Robert E Schapire. A game-theoretic approach to apprenticeship learning. *Advances in Neural Information Processing Systems*, 20, 2007.

[22] Vladimir Vapnik and Olivier Chapelle. Bounds on error expectation for support vector machines. *Neural Computation*, 12(9):2013–2036, 2000.

[23] Christian Wirth, Riad Akrour, Gerhard Neumann, and Johannes Fürnkranz. A survey of preference-based reinforcement learning methods. *Journal of Machine Learning Research*, 18(136):1–46, 2017.

[24] Yueh-Hua Wu, Nontawat Charoenphakdee, Han Bao, Voot Tangkaratt, and Masashi Sugiyama. Imitation learning from imperfect demonstration. In *International Conference on Machine Learning*, pages 6818–6827. PMLR, 2019.

[25] Songyuan Zhang, Zhangjie Cao, Dorsa Sadigh, and Yanan Sui. Confidence-aware imitation learning from demonstrations with varying optimality. *arXiv preprint arXiv:2110.14754*, 2021.

[26] Brian D. Ziebart, Sanjiban Choudhury, Xinyan Yan, and Paul Vernaza. Towards uniformly superhuman autonomy via subdominance minimization. In *Proceedings of the International Conference on Machine Learning*, pages 27654–27670, 2022.

A Existing Imitation Learning Methods and Satisificing

Behavioral cloning approaches [16] directly estimate a (stochastic) policy $\pi_\theta : \mathcal{S} \to \Delta_{\mathcal{A}}$ from demonstrated state-action pairs, (s_t, a_t). The simplicity of this approach allows the full range of supervised machine learning techniques to be employed to estimate the policy. For example, generative adversarial imitation learning (GAIL) [11] employs a discriminator to distinguish between human and automated action choices, and guide policy learning to minimize any differences. Unfortunately, behavioral cloning methods cannot outperform the demonstration policy beyond being Bayes optimal for a predictive loss that may not align with the acceptability set cost function(s).This prevents behavioral cloning methods from providing satisficing guarantees.

Inverse reinforcement learning [12] estimates the cost function $C(s)$ that explains or rationalizes demonstrations (making them near optimal). It is common to assume that the cost function is linear in a set of state features, $\mathbf{f} : \mathcal{S} \to \mathbb{R}^K$, or state-action features, $\mathbf{f} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^K$ [14]. Under this assumption, **feature matching** [1] guarantees the estimated policy $\hat{\pi}$ has expected cost under the demonstrator's unknown fixed cost function weights $\tilde{w} \in \mathbb{R}^K$ equal to the average of the demonstration policies π if the expected feature counts match:

$$\begin{aligned} &\mathbb{E}_{\substack{\tau_i \sim \tilde{\Xi}, \\ \xi \sim \pi \times \tau_i}} [f_k(\xi)] = \frac{1}{|\tilde{\Xi}|} \sum_{\tilde{\xi}_{i,j} \in \tilde{\Xi}} f_k(\tilde{\xi}_{i,j}), \forall k \\ &\implies \mathbb{E}_{\substack{\tau_i \sim \tilde{\Xi}, \\ \xi \sim \pi_\theta \times \tau_i}} [C_{\hat{w}}(\xi)] = \frac{1}{|\tilde{\Xi}|} \sum_{\tilde{\xi}_{i,j} \in \tilde{\Xi}} C_{\tilde{w}}(\tilde{\xi}_{i,j}), \end{aligned} \tag{3}$$

where $f_k(\xi) \triangleq \sum_{s_t, a_t \in \xi} f_k(s_t, a_t)$ and $C_{\hat{w}}(\xi) \triangleq \sum_{s_t, a_t \in \xi} C_{\hat{w}}(s_t, a_t)$. This feature-matching constraint (3) can be enforced using a potential term measuring the demonstration $\tilde{\xi}$'s suboptimality relative to induced behavior ξ. Closer to our approach, game-theoretic apprenticeship learning [21] assumes the sign of the linear cost function's weights are known and produces a policy that is guaranteed to be better in expectation than the demonstration average under worst-case weights.

Unfortunately, matching the demonstrator's unknown expected rewards (or outperforming on average) only guarantees that the imitator achieves the aspiration level in expectation. If the demonstrators' aspirations depend on context that is not incorporated in the learned cost function, better levels of aspiration will not be guaranteed. Thus, inverse reinforcement learning does not provide useful guarantees for per-demonstration satisficing; it is not a discriminative enough policy optimization method.

B Minimally Subdominant Inverse Optimal Control

Subdominance has been employed previously in inverse optimal control to make the optimal trajectory induced by learned linear cost function weights $\boldsymbol{w} \in \mathbb{R}^K_{\geq 0}$, outperform sets of task-specific demonstrations $\{\tilde{\Xi}_i\}$ [26]:

$$\min_{\boldsymbol{w} \geq 0} \min_{\boldsymbol{\alpha} \geq 0} \sum_{i=1}^{N} \frac{|\tilde{\Xi}_i|}{|\tilde{\Xi}|} \text{subdom}_{\boldsymbol{\alpha}}(\xi_i^*(\mathbf{w}), \tilde{\Xi}_i) + \frac{\lambda}{2}||\boldsymbol{\alpha}||, \text{ where:}$$

$$\text{subdom}_{\boldsymbol{\alpha}}(\xi, \tilde{\Xi}) = \frac{1}{|\tilde{\Xi}|} \sum_{\tilde{\xi} \in \tilde{\Xi}} \underbrace{\sum_k \overbrace{\left[\alpha_k (f_k(\xi) - f_k(\tilde{\xi})) + 1\right]_+}^{\text{(feature } k\text{) subdom}^k_{\alpha_k}(\xi, \tilde{\xi})}}_{\text{(aggregated) subdom}_{\boldsymbol{\alpha}}(\xi, \tilde{\xi})}, \tag{4}$$

with $[x]_+ \triangleq \max(x, 0)$ as the hinge function, and trajectory cost features $\mathbf{f} : \Xi \to \mathbb{R}^K_{\geq 0}$. Other variants include defining the subdominance using relative cost features, $\text{relsubdom}^k_{\alpha_k}(\xi, \tilde{\xi}) \triangleq \left[\alpha_k \left(\frac{f_k(\xi)}{f_k(\tilde{\xi})} - 1\right) + 1\right]_+$, and/or aggregating over feature dimensions using maximization, $\text{subdom}_{\boldsymbol{\alpha}}(\xi, \tilde{\xi}) \triangleq \max_k \text{subdom}^k_{\alpha_k}(\xi, \tilde{\xi})$ [26]. Like support vector machines [22], only a subset

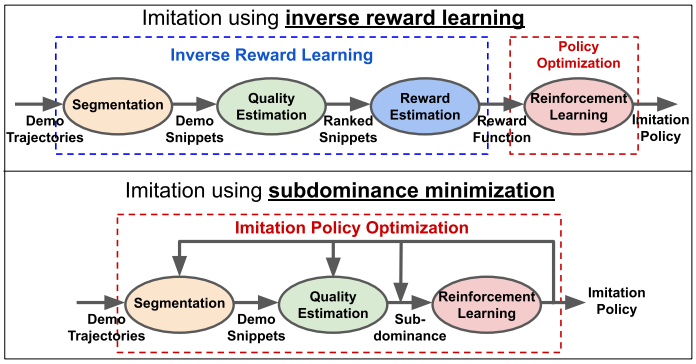

Figure 2: Existing reward-based imitation methods, e.g., TREX [5], seek to outperform demonstrations using a pipeline of engineered components (top) to first segment trajectories into "snippets," and to ultimately estimate a reward function that is then optimized using reinforcement learning. Our approach (bottom) uses the **subdominance** as the reinforcement learning objective, which is defined by the relative performance of the imitator compared to the demonstrations in each cost feature. This effectively uses feedback from the learned imitator policy to guide additional reinforcement learning without an explicit reward function.

of *support demonstrations*, $\tilde{\Xi}_i^{\mathrm{SV}_k}(\xi_i) \subseteq \tilde{\Xi}_i$, for each task i and feature k, actively influence $\boldsymbol{\theta}$:

$$\tilde{\xi}_{i,j} \in \tilde{\Xi}_i^{\mathrm{SV}_k}(\xi_i) \iff f_k(\xi_i) + \frac{1}{\alpha_k} \geq f_k(\tilde{\xi}_{i,j}). \tag{5}$$

Unfortunately, optimal control is impractical for many realistic imitation learning problems of interest. Additionally, it makes the learned cost/reward function (Fig. 2) a bottleneck that can prevent the imitation policy from better fitting to (or outperforming) demonstrations.

C Subdominance Policy Gradient Algorithms

C.1 Online MinsubFI

Algorithm 1 outlines our high-level approach for optimization. For each task (i), a trajectory is rolled out by sampling from the current learned policy (Line 2). The cost features of the sampled trajectory and the demonstrated trajectory are compared to determine which dimensions the sampled trajectory does not sufficiently outperform the demonstration, and are thus support vectors (Line 4). Here, the α values defining margin slopes (Eq. (5)) can either be optimized numerically (e.g., using stochastic gradient descent) or analytically [13] as described in appendix section G. A policy update is then employed to reduce the subdominance (Line 7).

Algorithm 1 Online subdominance policy gradient

1: **while** $\boldsymbol{\theta}$ not converged **do**
2: Sample a set of M trajectories $\Xi_i = \{\xi_i^{(m)}\}_{m=0}^{M}$ from policy $\pi_{\boldsymbol{\theta}} \times \tau_i$ for each task i
3: **for each** $\xi_i^{(m)} \in \Xi_i$ **do**
4: Find support vectors $\tilde{\Xi}_{i,m}^{\mathrm{SV}_k}$ (and $\boldsymbol{\alpha}$) given $\xi_i^{(m)}$
5: Compute loss $\mathcal{L}(\xi_i^{(m)}) = \mathrm{subdom}_{\boldsymbol{\alpha}}(\xi_i^{(m)}, \tilde{\Xi}_i)$
6: **end for**
7: Update $\boldsymbol{\theta}$ via policy gradient update rule on $\mathcal{L}(\xi_i^{(m)})$
8: **end while**

Theorem 4. *Policy $\pi_{\boldsymbol{\theta}}$'s subdominance with respect to demonstration set $\{\tilde{\Xi}_i\}$ has policy gradient:*

$$\nabla_{\boldsymbol{\theta}} \sum_i \frac{|\tilde{\Xi}_i|}{|\tilde{\Xi}|} \mathbb{E}_{\xi_i \sim \pi_\theta \times \tau_i} \Big[\text{subdom}_{\boldsymbol{\alpha}}(\xi_i, \tilde{\Xi}_i) \Big]$$
$$= \sum_i \frac{|\tilde{\Xi}_i|}{|\tilde{\Xi}|} \mathbb{E}_{\xi_i \sim \pi_\theta \times \tau_i} \Big[\text{subdom}_{\boldsymbol{\alpha}}(\xi_i, \tilde{\Xi}_i) \sum_{(s,a) \in \xi_i} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) \Big],$$

For a set of single trajectory samples, $\xi_i \sim \pi_\theta \times \tau_i$, for each task i, the policy parameters $\boldsymbol{\theta}$ can be (stochastically) updated via gradient descent: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \sum_i \sum_{(a_t,s_t) \in \xi_i} G_t \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t)$, where G_t is any function of the full or future expected subdominance, $\text{subdom}_{\boldsymbol{\alpha}}(\xi_i, \tilde{\Xi}_i)$*, such as the Q-value, the advantage estimate, or the trajectory return [20].*

Proof of Theorem 4. The general form of the gradient in the policy gradient update may be written as

$$g = \mathbb{E}_{\xi \sim \pi \times \tau} \left[\sum_{(s_t,a_t) \in \xi} G_t \nabla_\theta \log \pi_\theta(a_t|s_t) \right], \tag{6}$$

where G_t measures the quality of acting under policy π_θ in state s_t (e.g., discounted sum of future costs/rewards $\sum_{t+1}^{T} \gamma^t r(S_t)$, state-action value function $Q^{\pi_\theta}(S_t, A_t)$, advantage estimate $A^{\pi_\theta}(S_t, A_t)$, or a measure of expected future returns.

Further, the absolute subdominance of a trajectory can be decomposed over its states according to the equations of Corollary 5, which we rewrite for notational simplicity as:

$$\text{subdom}_{\boldsymbol{\alpha}}^{[\Sigma]}(\xi, \tilde{\Xi}) = \sum_{s_t \in \xi} \text{subdom}_{\boldsymbol{\alpha}}^{[\Sigma]}(s_t, \tilde{\Xi}) \tag{7}$$
$$= \sum_{s_t \in \xi, k} \text{subdom}_{\boldsymbol{\alpha}}^{[\Sigma],k}(s_t, \tilde{\Xi}), \tag{8}$$

where $\sum_{s_t \in \xi, k} \text{subdom}_{\boldsymbol{\alpha}}^{[\Sigma],k}(s_t, \tilde{\Xi})$ is the *contribution* of each state $s_t \in \xi$ towards the *total* subdominance the trajectory ξ. It can be computed as:

$$\text{subdom}_{\boldsymbol{\alpha}}^{[\Sigma],k}(s_t, \tilde{\Xi}) = \frac{\tilde{C}^k}{|\xi|} + \tilde{C}^k \alpha_k f_k(s_t) - \frac{\alpha_k \tilde{f}_{k,\text{abs}}^{(j)}}{|\xi||\tilde{\Xi}|},$$

where $\tilde{C}^k = \frac{|\tilde{\Xi}^{\text{SV}_k}|}{|\tilde{\Xi}|}$, $\tilde{f}_{k,\text{abs}}^{(j)} = \sum_{\tilde{\xi}_j \in \tilde{\Xi}^{\text{SV}_k}} \sum_{s'_t \in \tilde{\xi}_j} f_k(s'_t)$.

Assume G_t to be the total trajectory return in Equation (6) $G_t = \sum_{s_t \in \xi} r(s_t)$, and assume the subdominance *contribution* of each state to be its *negative* reward,

$$r(s_t) = -\text{subdom}_{\boldsymbol{\alpha}}^{[\Sigma]}(s_t, \tilde{\Xi})$$

Then Equation (6) may be rewritten as

$$g = \mathbb{E}_{\xi \sim \pi \times \tau} \Big[\sum_{(s_t,a_t) \in \xi} G_t \nabla_\theta \log \pi_\theta(a_t|s_t) \Big]$$
$$= \mathbb{E}_\xi \Big[\sum_{(s_t,a_t) \in \xi} \sum_{s_t \in \xi} r(s_t) \nabla_\theta \log \pi_\theta(a_t|s_t) \Big]$$
$$= \mathbb{E}_\xi \Big[\sum_{(s_t,a_t) \in \xi} \sum_{s_t \in \xi} -\text{subdom}_{\boldsymbol{\alpha}}^{[\Sigma]}(s_t, \tilde{\Xi}) \nabla_\theta \log \pi_\theta(a_t|s_t) \Big]$$
$$= \mathbb{E}_\xi \Big[\sum_{(s_t,a_t) \in \xi} -\text{subdom}_{\boldsymbol{\alpha}}^{[\Sigma]}(\xi, \tilde{\Xi}) \nabla_\theta \log \pi_\theta(a_t|s_t) \Big]$$
$$= \mathbb{E}_\xi \Big[-\text{subdom}_{\boldsymbol{\alpha}}^{[\Sigma]}(\xi, \tilde{\Xi}) \sum_{(s_t,a_t) \in \xi} \nabla_\theta \log \pi_\theta(a_t|s_t) \Big],$$

where $\xi \sim \pi \times \tau$, and the final expression follows from the fact that the *total* subdominance $\text{subdom}_{\boldsymbol{\alpha}}^{[\Sigma]}(\xi, \tilde{\Xi})$ of trajectory ξ is the same for each state $s_t \in \xi$. Substituting this gradient expression g into the policy gradient update rule over multiple tasks i, we get the subdominance policy gradient update rule in Theorem 4. Alternatively, decomposing the relative subdominance according to its definition in Corollary 5 gives us the equivalent result for the relative definition of subdominance. □

C.2 Per-State Cost Decomposition

Corollary 5. *The absolute and relative subdominances for a trajectory from a single task i with respect to a set of demonstrations can be further expanded as:*

$$\text{subdom}_{\boldsymbol{\alpha}}(\xi_i, \tilde{\Xi}_i) = \sum_{s_t \in \xi_i, k} \left(\frac{\tilde{C}_i^k}{|\xi_i|} + \tilde{C}_i^k \alpha_k f_k(s_t) - \frac{\alpha_k \tilde{f}_{k,\text{abs}}^{(i,j)}}{|\xi_i||\tilde{\Xi}_i|} \right);$$

$$\text{relsubdom}_{\boldsymbol{\alpha}}(\xi_i, \tilde{\Xi}_i) = \sum_{s_t \in \xi_i, k} \left(\frac{\tilde{C}_i^k (1-\alpha_k)}{|\xi_i|} + \frac{\alpha_k f_k(s_t) \tilde{f}_{k,\text{rel}}^{(i,j)}}{|\tilde{\Xi}_i|} \right),$$

where $\tilde{C}_i^k = \frac{|\tilde{\Xi}_i^{\text{SV}_k}|}{|\tilde{\Xi}_i|}$, $\tilde{f}_{k,\text{abs}}^{(i,j)} = \sum_{\tilde{\xi}_{i,j} \in \tilde{\Xi}_i^{\text{SV}_k}} \sum_{s'_t \in \tilde{\xi}_{i,j}} f_k(s'_t)$, *and* $\tilde{f}_{k,\text{rel}}^{(i,j)} = \sum_{\tilde{\xi}_{i,j} \in \tilde{\Xi}_i^{\text{SV}_k}} \left(\sum_{s'_t \in \tilde{\xi}_{i,j}} f_k(s'_t) \right)^{-1}$.

Proof of Corollary 5 (Absolute). The absolute, sum-aggregated subdominance is defined as:

$$\begin{aligned}
&\text{subdom}_{\boldsymbol{\alpha}}^{\Sigma}(\xi_{i,n}, \tilde{\Xi}_i) = \sum_k \text{subdom}_{\alpha_k}^k(\xi_{i,n}, \tilde{\Xi}_i) = \sum_k \frac{1}{|\tilde{\Xi}_i|} \sum_{\tilde{\xi}_{i,j} \in \tilde{\Xi}_i} \Big[\alpha_k (f_k(\xi_{i,n}) - f_k(\tilde{\xi}_{i,j})) + 1 \Big]_+ \\
&= \sum_k \frac{1}{|\tilde{\Xi}_i|} \sum_{\tilde{\xi}_{i,j} \in \tilde{\Xi}_{i,n}^{\text{SV}_k}} \Big(\alpha_k (f_k(\xi_{i,n}) - f_k(\tilde{\xi}_{i,j})) + 1 \Big) \qquad (9) \\
&= \sum_k \frac{\alpha_k}{|\tilde{\Xi}_i|} \sum_{\tilde{\xi}_{i,j} \in \tilde{\Xi}_{i,n}^{\text{SV}_k}} \left(\frac{1}{\alpha_k} + \sum_{s_t \in \xi_{i,n}} f_k(s_t) - \sum_{s'_t \in \tilde{\xi}_{i,j}} f_k(s'_t) \right) \\
&= \sum_{s_t \in \xi_{i,n}, k} \frac{\alpha_k}{|\tilde{\Xi}_i|} \sum_{\tilde{\xi}_{i,j} \in \tilde{\Xi}_{i,n}^{\text{SV}_k}} \left(\frac{1}{\alpha_k |\xi_{i,n}|} + f_k(s_t) - \frac{\sum_{s'_t \in \tilde{\xi}_{i,j}} f_k(s'_t)}{|\xi_{i,n}|} \right) \\
&= \sum_{s_t \in \xi_{i,n}, k} \frac{\alpha_k |\tilde{\Xi}_{i,n}^{\text{SV}_k}|}{|\tilde{\Xi}_i|} \left(\frac{1}{\alpha_k |\xi_{i,n}|} + f_k(s_t) - \frac{\sum_{\tilde{\xi}_{i,j} \in \tilde{\Xi}_{i,n}^{\text{SV}_k}} \sum_{s'_t \in \tilde{\xi}_{i,j}} f_k(s'_t)}{|\xi_{i,n}||\tilde{\Xi}_{i,n}^{\text{SV}_k}|} \right) \\
&= \sum_{s_t \in \xi_i, k} \frac{\tilde{C}_i^k}{|\xi_i|} + \tilde{C}_i^k \alpha_k f_k(s_t) - \frac{\alpha_k \tilde{f}_{k,\text{abs}}^{(i,j)}}{|\xi_i||\tilde{\Xi}_i|},
\end{aligned}$$

where $\tilde{C}_i^k = \frac{|\tilde{\Xi}_i^{\text{SV}_k}|}{|\tilde{\Xi}_i|}$, and $\tilde{f}_{k,\text{abs}}^{(i,j)} = \sum_{\tilde{\xi}_{i,j} \in \tilde{\Xi}_i^{\text{SV}_k}} \sum_{s'_t \in \tilde{\xi}_{i,j}} f_k(s'_t)$. □

Proof of Corollary 5 (Relative). The relative, sum-aggregated subdominance is defined as:

$$\begin{aligned}
&\text{relsubdom}^{\Sigma}_{\boldsymbol{\alpha}}(\xi_{i,n}, \tilde{\Xi}_i) = \sum_k \text{relsubdom}^{k}_{\alpha_k}(\xi_{i,n}, \tilde{\Xi}_i) \\
&= \sum_k \frac{1}{|\tilde{\Xi}_i|} \sum_{\tilde{\xi}_{i,j} \in \tilde{\Xi}_i} \left[\alpha_k \left(\frac{f_k(\xi_{i,n})}{f_k(\tilde{\xi}_{i,j})} - 1 \right) + 1 \right]_+ \\
&= \sum_k \frac{1}{|\tilde{\Xi}_i|} \sum_{\tilde{\xi}_{i,j} \in \tilde{\Xi}_i^{\text{SV}_k}} \left(\alpha_k \left(\frac{f_k(\xi_{i,n})}{f_k(\tilde{\xi}_{i,j})} - 1 \right) + 1 \right) \\
&= \sum_k \frac{1}{|\tilde{\Xi}_i|} \sum_{\tilde{\xi}_{i,j} \in \tilde{\Xi}_i^{\text{SV}_k}} \left((\beta_k - \alpha_k) + \alpha_k \frac{\sum_{s_t \in \xi_{i,n}} f_k(s_t)}{\sum_{s'_t \in \tilde{\xi}_{i,j}} f_k(s'_t)} \right) \\
&= \sum_{s_t \in \xi_{i,n}, k} \left(\frac{(1-\alpha_k)|\tilde{\Xi}_i^{\text{SV}_k}|}{|\xi_i||\tilde{\Xi}_i|} + \frac{\alpha_k f_k(s_t)}{|\tilde{\Xi}_i|} \sum_{\tilde{\xi}_{i,j} \in \tilde{\Xi}_i^{\text{SV}_k}} \sum_{s'_t \in \tilde{\xi}_{i,j}} \frac{1}{\sum f_k(s'_t)} \right) \\
&= \sum_{s_t \in \xi_i, k} \frac{\tilde{C}_i^k (1-\alpha_k)}{|\xi_i|} + \frac{\alpha_k f_k(s_t) \tilde{f}^{(i,j)}_{k,\text{rel}}}{|\tilde{\Xi}_i|},
\end{aligned} \tag{10}$$

where $\tilde{C}_i^k = \frac{|\tilde{\Xi}_i^{\text{SV}_k}|}{|\tilde{\Xi}_i|}$, and $\tilde{f}^{(i,j)}_{k,\text{rel}} = \sum_{\tilde{\xi}_{i,j} \in \tilde{\Xi}_i^{\text{SV}_k}} \left(\sum_{s'_t \in \tilde{\xi}_{i,j}} f_k(s'_t) \right)^{-1}$. □

Subdominance using the maximum over each feature to aggregate per-feature subdominances takes a similar form. It becomes identical to the expression starting from Equation (9) with the key difference that each demonstration can only be a support vector for a single feature dimension k for max-aggregated subdominance (and conversely, each demonstration can be a support vector for multiple feature dimensions k for sum-aggregated subdominance).

This decomposition enables state-of-the-art reinforcement learning algorithms [18] that assign credit to actions in a causally consistent manner (i.e., only future returns influence an action's updates) to be employed. Further flexibility is gained via the choice of policy representation. When deploying or simulating a policy is expensive, offline policy gradient methods that are based entirely on the set of demonstrated trajectories can instead be employed.

C.3 Offline MinSubFI

Corollary 6. *Offline policy gradient (MinSubFI$_{OFF}$) employs importance weighting to estimate the gradient for online subdominance minimization from available demonstrations::*

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \sum_{i, \tilde{\xi}_{i,j} \in \tilde{\Xi}_i} \tilde{r}_\theta^{(i,j)} \text{subdom}_{\boldsymbol{\alpha}}(\tilde{\xi}_{i,j}, \tilde{\Xi}_i) \sum_{(s,a) \in \tilde{\xi}_{i,j}} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s)\,, \tag{11}$$

where $\tilde{r}_\theta^{(i,j)} = \frac{\pi_{\boldsymbol{\theta}}(\tilde{\xi}_{i,j})}{\tilde{\pi}(\tilde{\xi}_{i,j})}$ is the importance ratio, and $\tilde{\pi}$ is an estimate of the demonstrator's policy.

The high-level approach of the offline policy gradient method outlined in Corollary 6 is outlined in Algorithm 2.

D Generalization Bound

Our generalization bound relies on the absence of distinct local optima of the objective function. Formally, this is provided by the property of quasiconvexity, which guarantees that regions achieving a particular level of subdominance (or lower) are convex.

Lemma 7. *When the set of features $\mathcal{F}$ realizable by the class of policies ($\mathcal{F} : \mathbf{f}(\xi) \in \mathcal{F}, \forall \xi \in \Pi$) is convex, the subdominance of realizable features is a quasiconvex function.*

Algorithm 2 Offline, joint stochastic sub-gradient optimization

Estimate $\tilde{\pi}$ by performing behavior cloning on demonstrations $\tilde{\Xi}$
while $\boldsymbol{\theta}$ not converged **do**
  for each $\tilde{\xi}_{i,j} \in \tilde{\Xi}_i$ **do**
    Find support vectors $\tilde{\Xi}_{i,j}^{\text{SV}_k}(\alpha_k)$ given $\tilde{\xi}_{i,j}$
    for each k **do**
      $\alpha_k \leftarrow \alpha_k \exp\{-\eta_t' \tilde{r}_{\theta}^{(i,j)} \sum_{\tilde{\xi}_{i,m} \in \tilde{\Xi}_{i,j}^{\text{SV}_k}(\alpha_k)} (f_k^{(\tilde{\xi}_{i,j})} - f_k^{(\tilde{\xi}_{i,m})}) + \lambda |\tilde{\Xi}| \alpha_k\}$
    end for
    Update $\boldsymbol{\theta}$ according to Equation (11).
  end for
end while

Proof. As a function of the realized features (f_k) of the imitator, $\min_{\alpha_k} \text{subdom}_{\alpha_k,1}^k(f_k, \tilde{\Xi})$ is monotonic (increasing). Thus, $\min_{\alpha} \text{subdom}_{\alpha,\mathbf{1}}(\mathbf{f}, \tilde{\Xi})$ is a quasiconvex function of $\mathbf{f}$ for sum- or max-aggregated subdominance. The intersection of any sublevel set of $\min_{\alpha} \text{subdom}_{\alpha,\mathbf{1}}(\mathbf{f}, \tilde{\Xi})$ with $\mathcal{F}$ is also convex. Therefore, $\min_{\mathbf{f} \in \mathcal{F}} \text{subdom}_{\alpha}(\mathbf{f}, \tilde{\Xi})$ is a quasiconvex minimization problem. □

Proof of Theorem 9. The generalization guarantee is based on leave-one-out cross validation error, which is an almost-unbiased estimate of generalization error under IID assumptions [22]. Removing non-support vectors does not change global optima of subdominance minimization when no distinct local optima exist, which is the case for this quasiconvex optimization problem. □

E Generalization Bound Analysis

We now define the notion of a γ–**satisficing** stochastic policies and present a generalization bound.

Definition 8. A policy is considered γ–**satisficing** (or γ–**acceptable**) for cost features $\mathbf{f}$ and distribution of demonstrated trajectories $P(\tilde{\xi})$, if its trajectories ξ drawn from policy π satisfies with probability at least γ: $P(\xi \in \Omega_{\tilde{\xi}}) \geq \gamma$.

Theorem 9. *The policy minimizing the absolute or relative* $\text{subdom}_{\boldsymbol{\alpha}}\left(\xi^*(\pi_{\boldsymbol{\theta}}), \tilde{\xi}_i\right)$ *(N iid demonstrations) with realizable features that are convex sets has the support vector set* $\left\{\tilde{\Xi}_{SV_k}(\xi^*(\pi_{\boldsymbol{\theta}}), \alpha_k)\right\}$ *and is on average* γ–**satisficing** *on the population distribution with:* $\gamma = 1 - \frac{1}{N}\left\| \bigcup_{k=1}^{K} \tilde{\Xi}_{SV_k}(\xi^*(\pi_{\boldsymbol{\theta}}), \alpha_k) \right\|$.

This bound motivates subdominance minimization for producing demonstrator-acceptable behavior.

F Learning a Cost Feature Representation

Formulation: Out formulation for learning a cost feature representation in §1.3 differs from the formulation of TREX in two key aspects. First, under the exponential preference model [3, 9, 5, 6], we employ subdominance between pairs of demonstrations as a loss function, rather than a linear cost function. The second difference emerges from choosing subdominance as the loss function: our formulation permits learning latent representations of *any* dimensionality, rather than just a scalar cost signal; such a vector representation allows us to recover multiple, competing objectives from preferences, rather than arbitrarily extrapolating over a scalar reward signal.

Architecture: To learn the cost feature representation, we use a multi-layer perceptron network with two hidden layers of width 8 as our cost feature architecture. In contrast with TREX, which employs four different levels of preference (or ranks) to categorize demonstration quality, we consider two preference levels (i.e., *acceptable* or *not acceptable*).

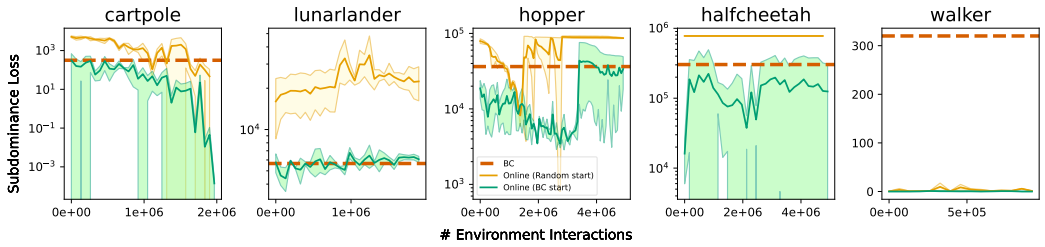

Figure 3: Subdominance loss as a function of online environment interactions for MinSubFI$_{\text{ON}}$ initialized from either: a random policy (yellow), or a behavior cloning policy (green). Subdominance of the behavioral cloning policy (brown) is shown for comparison. Some environments (e.g., `lundarlander`, `halfcheetah`) require non-random initialization for subdominance minimization method to be effective.

G Optimization of α: Numerical and Analytical

The α values of the subdominance define the sensitivity of the learned policy to not performing significantly better than demonstrations. For a given imitator trajectory and set of demonstrations, the α values can be updated numerically via (exponentiated) stochastic gradient optimization (e.g., within Algorithm 1), as shown in Algorithm 3.

Algorithm 3 Online update of α values

1: **for each** k **do**

2: $\alpha_k \leftarrow \alpha_k \exp\{-\eta'_t \sum_{i,\tilde{\xi}_{i,j} \in \tilde{\Xi}^{\text{SV}_k}_{i,m}} (f_k^{(\xi_i)} - f_k^{(\tilde{\xi}_{i,j})}) + \lambda |\tilde{\Xi}| \alpha_k\}$

3: **end for**

Alternatively, the optimal α values for can be computed analytically [13]:

$$\alpha_k^* = \arg\min_{\alpha_k} m \text{ such that: } f_k(\xi) + \lambda \leq \frac{1}{m} \sum_{j=1}^{m} f_k(\xi^{(j)}), . \tag{12}$$

where $\alpha_k^{(j)} = \frac{1}{f_k(\xi) - f_k(\tilde{\xi}^{(j)})}$ is the hinge slope that makes demonstration $\tilde{\xi}^{(j)}$ exactly where the subdominance becomes zero. Further, naively approaching the optimization in Equation 2 can be problematic, since $\boldsymbol{\alpha} = 0$ corresponds to a degenerate local optimum. However, the optimal $\boldsymbol{\alpha}$ values for a policy achieving at least the average feature counts of the demonstrations are not degenerate. This suggests bootstrapping from an initial policy estimate when minimizing $\boldsymbol{\alpha}$ values or restricting $\boldsymbol{\alpha}$ values above zero.

H Non-Random Policy Initialization

Using Algorithm 1 and analytically computed α values in step 4, we train our Online MinSubFI policy with different policy initializations. For all of our experiments throughout the paper, we employ a quadratic expansion of the original cost features, as described in Table 4 in Appendix J.2. Figure 3 shows the resulting training subdominance loss curves for each. While low subdominance is achieved by all initialization methods for some environments, some environments (e.g., `lundarlander`, `halfcheetah`) require non-random initialization for subdominance minimization method to be effective. We adopt behavioral-cloned policy initialization in the remainder of our experiments.

I Demonstration Details

For each environment, we obtain 100 demonstrations using a suboptimal policy learned using PPO. For each timestep in a demonstration, we collect the state observation vector along with the

corresponding action, cost feature vector, and true reward (for evaluation only). Demonstration return statistics for environment-specific demonstration sets of varying quality are provided in Table 3.

Table 3: True return statistics of demonstration sets for each environment (100 demonstrations each).

Environment	Min	Mean	Max
lunarlander	-196	112	284
cartpole	10	76	194
hopper	6	939	3441
halfcheetah	-83	680	1483
walker	18	968	4293

J Implementation Details

J.1 Environments

The environments used for our experiments are famous games re-implemented by OpenAI's gym [4], providing the tools and interface for interacting with reinforcement learning algorithms. We present the specifications and goals of the environments considered in this work. The available environments are separated into two categories: classic control (Cartpole, Lunar Lander) and MuJoCo environments (Hopper, Walker, HalfCheetah). Observations in classic control environments are 1D state vectors.

J.1.1 cartpole (`CartPole-v0`)

The task is to keep a rotating pole, attached to a moving cart, vertical for as long as possible under a gravity model. The player can control the angle by moving the cart either left or right, each movement affecting the angular velocity of the pole. An episode terminates when the pole angle θ exceeds $\pm 12^\circ$ (from the vertical y-axis) or, when the cart position x exceeds ± 2.4. Maximum true return is 200.

J.1.2 lunarlander (`LunarLander-v2`)

The task is to land a shuttle that operates under a gravitational model, on the surface of the moon. An initial force is applied to the lander, providing with a starting velocity and angle; the player must then balance the shuttle and land it at the center of the screen, in a location delimited by two yellow flags. The player controls the shuttle by engaging one vertical and two lateral engines; the main vertical engine displaces the lander while the lateral ones pitch the lander. In the MinSubFI implementation, we modified the lander to have fixed starting parameters (starting force and moon layout) which then characterize a single task; the maximum true reward in this case is approximately 310.

J.1.3 hopper (`Hopper-v3`)

The task is to control various joints of a hopping robot (restricted to the vertical xz-plane) to "hop" and make forward progress. Reward at each timestep is a function of the forward velocity of the robot and its "health" (determined by the physics engine based on the joint angles of the robot).

J.1.4 halfcheetah (`HalfCheetah-v3`)

The task is to control various joints of a bipedal robot (restricted to the vertical xz-plane) to "run" and make forward progress. Reward at each timestep is a function of the forward velocity of the robot and its "health" (determined by the physics engine based on the joint angles of the robot).

J.1.5 walker (`Walker2d-v3`)

Adds an additional leg to the hopper environment so that the task is to "walk" forward rather than "hop".

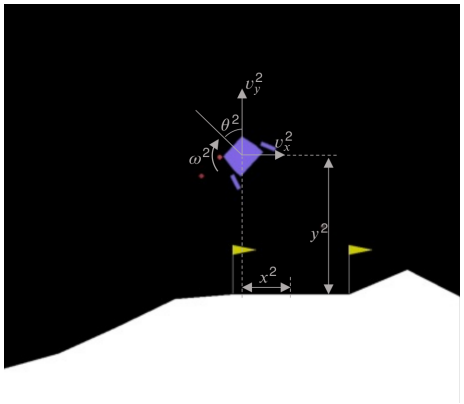

Figure 4: Illustration of cost features for the lunarlander environment. These features can be computed directly from the environment's observation vector.

Table 4: Description of cost features for each environment. $\sigma(x) = (1 + \exp(x))^{-1}$ is the sigmoid function used to scale features to $[-1, +1]$.

Environment	# Cost Features	Cost Feature	Description
`cartpole`	4	x^2	(cart position)^2
		v^2	(cart velocity)^2
		θ^2	(pole angle)^2
		ω^2	(pole angular velocity)^2
`lunarlander`	6	x^2	(lander x-position)^2
		y^2	(lander y-position)^2
		v_x^2	(lander x-velocity)^2
		v_y^2	(lander y-velocity)^2
		θ^2	(lander angle)^2
		ω^2	(lander angular velocity)^2
	3	$\|a_t\|_2^2$	control cost
`hopper`	2	$-\sigma(v_x^{top}) + 1$	cost inversely proportional to x-velocity
		$-\sigma(v_z^{top}) + 1$	cost inversely proportional to z-velocity
	1	$1 - tanh(z)$	cost inversely proportional to z-position
	1	$-\sigma(\theta) + 1$	cost inversely proportional to torso-angle
	1	$\sum \|a_t\|_2^2$	control cost
`halfcheetah`	1	$-\sigma(v_x^{top}) + 1$	cost inversely proportional to x-velocity
	1	$-v_x^{top} + 10$	cost inversely proportional to x-velocity
	1	$-[v_x^{top}]_+ + 10$	cost inversely proportional to x-velocity
	1	$\sum \|a_t\|_2^2$	control cost
`walker`	2	$-\sigma(v_x^{top}) + 1$	cost inversely proportional to x-velocity
		$-\sigma(-z) + 1$	cost inversely proportional to z-position
	1	$\sum \|a_t\|_2^2$	control cost

J.2 Computing Cost Features

For all environments employed in our experiments, we compute trajectory cost features $\mathbf{f} : \Xi \to \mathbb{R}^K_{\geq 0}$ that characterize the trajectory. Trajectory cost features are additive over the states of the trajectory, which are in turn characterized by state cost features $\mathbf{f} : \mathcal{S} \to \mathbb{R}^K_{\geq 0}$ or state-action cost features $\mathbf{f} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^K_{\geq 0}$. The specific cost features employed differ between environments, but are chosen such that they may be readily computed from each environment's respective observation and/or action vector. For example, in case of cartpole and lunarlander environments, for a given state s_t, the cost

feature vector may be computed as either

$$\mathbf{f}(s_t) = \{\phi_k(s_t)^2\}_{k=1}^K$$

where $\boldsymbol{\phi}(s) : \mathcal{S} \to \mathbb{R}^D$ is simply the D-dimensional observation vector returned by the respective environments. Alternatively, $\boldsymbol{\phi}$ can be chosen to also be a function of the actions a_t, $\boldsymbol{\phi} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^D$; this is useful in integrating control costs in the subdominance minimization problem. The cost feature set can be expanded to include any linear or non-linear, monotonic transformations of the entire cost feature vector $\mathbf{f}$ or a subset of its components $\mathbf{f}_k$. We can expand this cost feature set by computing the outer product of the original cost feature vector, $\mathbf{f}_{\text{expanded}} = \mathbf{f} \cdot \mathbf{f}^\top$.

In essence, cost features are easily characterizable properties of each environment which, when minimized over a trajectory, allow an agent to successfully complete a task. Note, however, that these features are chosen carefully so as to not leak the true cost signal for an environment. For example, for the cartpole problem, the cost features comprise the pole angle θ^2, pole angular velocity ω^2, the cart position x^2, and the cart velocity v^2. While these cost features are pertinent to task-completion, they are unrelated to the true reward signal for the cartpole environment i.e., the number of timesteps elpased before the pole tips over. The number of cost features defined is environment-specific; the complete list of cost features defined for each environment is provided in Table 4.

J.3 Trajectory Padding

It follows from the definition of subdominance (Eq. (5)) that minimizing $f_k(\xi)$ naturally minimizes subdominance. Based on the specific definition of cost features employed, this sometimes results in degenerate policies. This degenerate behavior most commonly manifests as the trained policy learning to terminate episodes early to achieve lower subdominance via encountering fewer states in the trajectory. This phenomenon is best illustrated using the following example with a single, simple cost feature.

J.3.1 Example

Consider a problem setting where we employ a single cost feature f. An agent incurs cost features $f(s) = 0$ upon reaching the terminal state s_{success} and $f(s) = 10$ in all other states (including s_{fail}.) Now, consider three trajectories for this task – a human demonstration $\tilde{\xi} = \{s_1, s_2, s_3, s_4, s_{\text{success}}\}$, and two trajectories $\xi_1 = \{s_1, s_2, s_3, s_4, s_5, s_6, s_{\text{success}}\}$ and $\xi_2 = \{s_1, s_2, s_{\text{fail}}\}$, sampled from policies π_1 and π_2 respectively. In choosing between the candidate policies π_1 and π_2, an agent opts for π_2, since, given any α, π_2 results in lower subdominance despite not completing the task successfully.

$$\begin{aligned}
f(\tilde{\xi}) &= \sum_{s_t \in \tilde{\xi}} f(s_t) = (4 \times 10) + 0 = 40 \\
f(\xi_1) &= \sum_{s_t \in \xi_1} f(s_t) = (6 \times 10) + 0 = 60 \\
f(\xi_2) &= \sum_{s_t \in \xi_2} f(s_t) = (3 \times 10) = 30 \\
\implies \text{[rel]subdom}_{\alpha,\beta}^{[\Sigma]}(\xi_1, \tilde{\xi}) &> \text{[rel]subdom}_{\alpha,\beta}^{[\Sigma]}(\xi_1, \tilde{\xi}) \\
&\implies \xi_1 \prec \xi_2.
\end{aligned}$$

This toy examples gives us a peek into the source of this degeneracy. This phenomenon is very similar in nature to 'reward gaming' often encountered in other reinforcement learning settings [2]. In our problem setting, this typically results from misalignment between the defined cost features and task objective, and is encountered experimentally when training is initialized from a random policy in such cases. For environments with such misaligned cost features and when starting subdominance minimization from a random policy, we employ the trajectory padding scheme described next.

J.3.2 Padding Scheme

For an environment with misaligned cost features $\mathbf{f}^{(\text{mis})} \in \mathbb{R}^K_{\geq 0}$, we fix a time horizon $h < H$ where H is number of steps deemed sufficient to complete the task objective for that environment. The cost

features of any short trajectory $\xi = (\mathbf{f}_1^{(\text{mis})}, \ldots, \mathbf{f}_T^{(\text{mis})})$ where $T < h$ is padded with a fixed padding cost vector $\mathbf{f}_{\text{pad}} \in \mathbb{R}^K_{\geq 0}$ up to the horizon h. The padded/augmented trajectory ξ' then becomes:

$$\xi' = (\mathbf{f}_1^{(\text{mis})}, \ldots, \mathbf{f}_T^{(\text{mis})}, \mathbf{f}_{T+1}^{(\text{pad})}, \ldots, \mathbf{f}_h^{(\text{pad})}).$$

Intuitively, this enables a random policy to avoid degenerate solutions by augmenting the cost of such solutions.

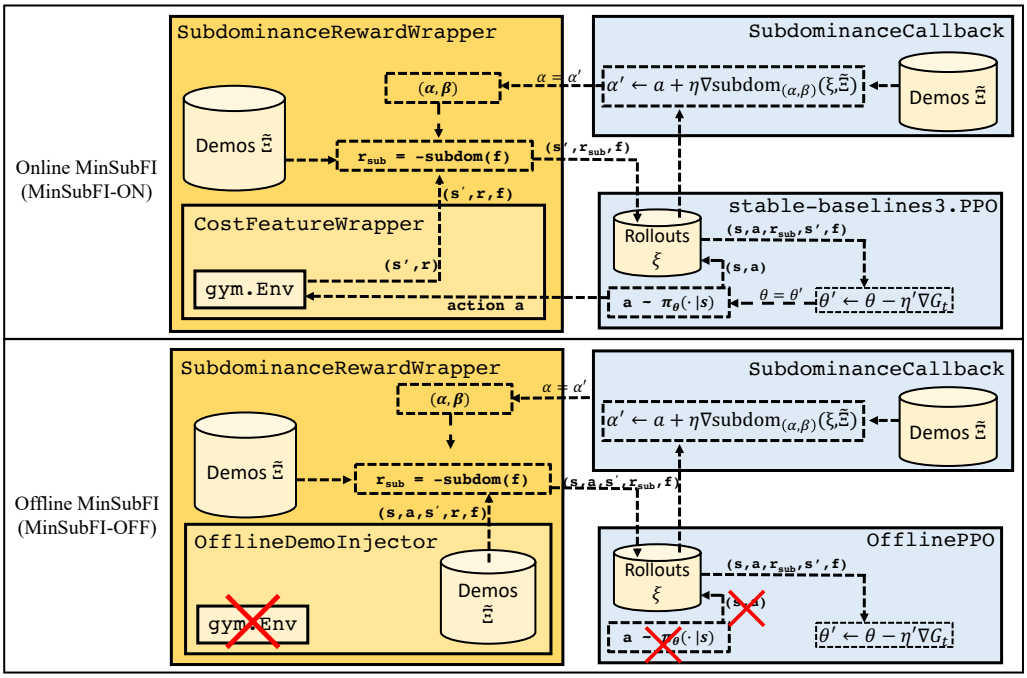

Figure 5: Implemented architecture of Online MinSubFI (top) and Offline MinSubFI (bottom) using `gym` and `stable-baselines3`. The primary functionality of cost feature and subdominance computation is tackled via two environment wrappers `CostFeatureWrapper` and `SubdominanceRewardWrapper`. The former computes cost features from observations and the latter computes subdominance relative to demonstrations using cost features. `SubdominanceCallback` is called periodically to update α. Offline MinSubFI uses the `OfflineDemoInjector` wrapper around an environment to pass (s, a, r, s', f) tuples from demonstrations as rollout data instead, and there is no action returned from the policy to the environment.

J.4 Reinforcement Learning

For our experiments, we utilize a modified Proximal Policy Optimization (PPO) approach for our policy gradient updates. We implement the policy optimization of MinSubFI using Stable Baselines3's [17] implementation of the PPO algorithm [18] employing the same base policy model across all experiments and baseline methods, and minimal hyperparameter tuning (hyperparameters provided in the Appendix section J.5). We build our core subdominance minimization functionalities via wrapper classes for `gym` environments and callback classes for `stable-baselines3` models. The MinSubFI architectures for online and offline training are shown in Figure 5. Specifically, for online MinSubFI the `CostFeatureWrapper` computes cost features `f` for observation `s` received from the environment. The `SubdominanceRewardWrapper` contains the demonstrator's cost features which it uses to compute the subdominance; environment reward `r` is replaced with negative subdominance $\texttt{r}_{\text{sub}}$ and returned to the `PPO` agent. We find that equivalently returning the *total* negative subdominance as a sparse cost in the terminal state of the rollout (i.e., avoiding the per-step cost decomposition from Corollary 5) works equally well in practice. The subdominance slopes α are updated via a periodic call to `SubdominanceCallback`. For offline MinSubFI, we create a dummy environment wrapper `OfflineDemoInjector` which returns (s, a, r) tuples sequentially from demonstrations *c*oncealed as rollouts. We build `OfflinePPO` and modify it's rollout collection to *n*ot sample the policy, and

instead treat the demonstrator's action as the one taken. Finally, the offline MinSubFI architecture in Figure 5 shows three separate demonstration buffers only for sake of clarity.

J.5 Hardware Details and Hyperparameters Used

Details of the hardware employed are provided in Table 5. Training the online version of MinSubFI end-to-end requires approximately 2 hours for $9e6$ environment interactions, on lunarlander, utilizing approximately 15% of the GPU's memory; measured on a tabletop computer equipped with an NVIDIA GeForce RTX 3080 GPU and Ryzen 5600X CPU. The other two more powerful machines were used for concurrent experimentation. The values of (relevant) hyperparameters used for each environment are provided in Table 5 and Table 6 respectively.

Table 5: Hardware used for experiments.

Machine Tag	OS	GPU (VRAM)	CPU	Memory
Tabletop PC	Ubuntu 20.04	GeForce RTX 3080 (10GB)	AMD Ryzen 5 5600X	64 GB
Lab Server	Ubuntu 20.04	2 x GeForce GTX 1080 Ti (12GB)	Intel Xeon E5-2697	180 GB
Shared Cluster	Ubuntu 20.04	2 x Tesla V100 (32 GB)	Intel Xeon Silver 4114	380 GB

Table 6: Values of relevant PPO hyperparameters for each environment

Environment	learning rate	entropy coefficient	minibatch size	horizon	epochs	clip range	total steps
cartpole	1e−4	0	512	2048	10	0.2	2e6
lunarlander	1e−4	1e−6	2048	2048	10	0.2	2e6
hopper	9.8e−5	1e−2	512	2048	5	0.2	5e6
halfcheetah	9.8e−5	1e−4	256	2048	5	0.2	5e6
walker	2e−5	6e−4	32	512	20	0.1	1e6

K Impact of Demonstration Quality on Performance

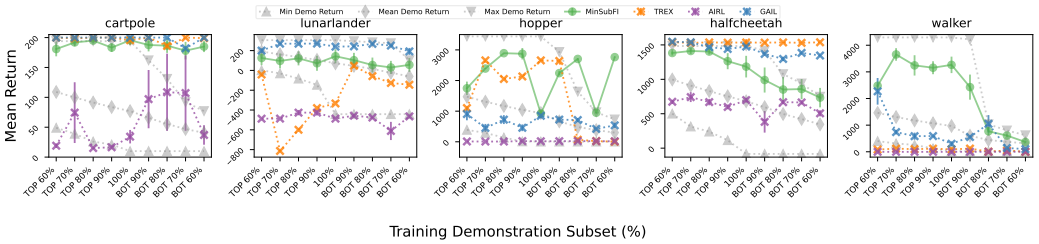

Figure 6: Mean true returns of 100 trajectories rolled out from the learned policies and the minimum, average, and maximum reward of the training set trajectories. Each policy was trained on a subset of demonstrations obtained by removing the best or worst 10%, 20%, 30%, or 40% of the demonstrations. Compared to T-REX (orange), GAIL (blue), and AIRL (purple), the performance of MinSubFI (green) is more robust to increases in the proportion of suboptimal demonstrations in the dataset.

Demonstrated behavior is often noisy and suboptimal, making learning from such data a desirable capability. In this section, we control the quality of demonstrations used for imitation. We sort all demonstrations by their total (true) return and then choose a subset by retaining the best or worst 90%, 80%, 70%, or 60% of the original set. We use this demonstration subset to train T-REX and Online MinSubFI. The performance is shown in Figure 6.

For the simple `cartpole` environment, TREX, GAIL, and MinSubFI are all able to continue outperforming the best demonstrations even when they become worse in quality. For the remaining environments, except for `halfcheetah` in which TREX performs exceptionally well, MinSubFI tends to provide better true returns as the quality of demonstrations becomes worse. The performance of AIRL, which assumes demonstrations are optimal, is poor across all environments often failing to match even the mean returns of demonstrations.