Supplementary for Interpretable and Personalized Apprenticeship Scheduling: Learning Interpretable Scheduling Policies from Heterogeneous User Demonstrations

Anonymous Author(s)
Affiliation
Address
email

1 Additional Experiment Domain Details

Synthetic Scheduling Environment The synthetic scheduling environment represents one of the hardest scheduling problems. In this environment, two agents must complete a set of 20 tasks which have upper- and lower-bound temporal constraints (i.e., deadline and wait constraints), proximity constraints (i.e., no two agents can be in the same place at the same time), and travel-time constraints. For the purposes of apprenticeship learning, an action is defined as the assignment of an agent to complete a task presently. The decision-maker must decide the optimal sequence of actions according to the decision-maker's own criteria. For this environment, we construct a set of heterogeneous, mock decision-makers that schedule according to Equation 1.

$$\tau_i^* = \underset{\tau_j \subset \boldsymbol{\tau_S}}{\arg\max}(\rho_1 H_{EDF}(\tau_j) + \rho_2 H_{dist}(\tau_j) + H_{ID}(\tau_j, \rho_3)) \tag{1}$$

In this equation, our decision-maker selects a task τ_i^* from the set of tasks $\boldsymbol{\tau_S}$. The task-prioritization scheme is based on three criteria: H_{EDF} prioritizes tasks according to deadline (i.e., "earliest-deadline first"), H_{dist} prioritizes the closest task, and H_{ID} prioritizes tasks according to a user-specified highest/lowest index or value based on ρ_3 (i.e., $\rho_3(j) + (1 - \rho_3)(-j)$). The heterogeneity in decision-making comes from the latent weighting vector $\vec{\rho}$. Specifically, $\rho_1 \in \mathbb{R}$ and $\rho_2 \in \mathbb{R}$ weight the importance of H_{EDF} and H_{dist}, respectively. $\rho_3 \in \{0, 1\}$ is a mode selector in which the highest/lowest task index is prioritized. By drawing $\vec{\rho}$ from a multivariate random distribution, we can create an infinite number of unique demonstrator types. This adapted environment differs from the synthetic, low-dimensional environment in that there are a rich set of temporal, spatial, and agent-based constraints modeling the job-shop scheduling problem; furthermore, the parameters of the demonstrator's decision-making process is hidden and comprised of one discrete factor and two continuous factors. In this domain, counterfactuals are generated by consider specific task information such as availability, distance from agent, prerequisites satisfied.

Real-world Data: Taxi Domain Our environment has three locations: the village, the airport, and the city. The taxi driver has the objective of picking up a passenger from the city or village. There is always a passenger at the city, but the taxi driver may encounter up to 60 minutes of traffic going into the city. There may be a wait time of up to 60 minutes to pick up a passenger at the village; however, there is no traffic on the way to the village, and the wait time is unknown to the taxi driver unless she is at the village. A dataset of 70 human-collected tree policies to solve this task (given with leaf node actions such as "Drive to the City", "Drive to the Airport", and "Wait for Passenger", and decision node criterion depending on the amount of wait time, traffic time, and current location) are used to generate heterogeneous trajectories. We originally collect 98 tree-based policies through an IRB-approved study. However, 28 of these do not produce successful trajectories. The tree-based policies

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

Table 1: Apprenticeship Performance in Imitating Robot Kinesthetic Ping Pong Demonstrations.

Environment	**Our Method**	Sammut et al.	Nikolaidis et al.	Tamar et al.	Hsiao et. al.	InfoGAIL Li et. al.	Gombolay et. al.
Ping-Pong	**59.60%**	18.14%	31.20%	26.17%	17.96%	36.70%	28.60%

can be found in this GitHub repository https://github.com/Personalized-Neural-Trees/Interpretable-and-Personalized-Apprenticeship-Scheduling-Learning-Interpretable-Scheduling-Policies.

Kinesthetic Robot Table Tennis We collected a real-world data set consisting of 10 human demonstrators kinesthetically presenting four different table tennis strikes on a Rethink Robotics Sawyer. The table tennis strike variants consisted of push, topspin, slice, and lob and were conducted using a forehand motion, giving four different categories of motion. While our approach is primarily for discrete classification problems, such as decision making, it can naturally be extended to complex continuous domains, such as low-level robot joint control.

To collect data, we first show each demonstrator a sample video of the table tennis strike and allow them to practice until she feels confident that she can return the ping pong ball over the net. Then, we reset the robotic arm to a preset initial position and allow the demonstrator to strike a ping pong ball launched from an automatic ball launcher. Throughout the demonstration, we record the position of the end-effector.

Survey Scheduling Environment This domain describes a **ND[ST-SR-TA]** scheduling domain defined by Korsah [5]. In this domain, synthetic schedulers are given utilities of three tasks where utility $U \in \{1, 2, 3\}$ and must choose the highest or lowest task index based on a pre-specified latent decision-making criteria. We generate a set of 100 schedules (each of length 20) from heterogeneous demonstrators.

2 LfD Performance in Kinesthetic Robot Table Tennis

Here, we show that Personalized Neural Trees can easily be extended to a variety of domains, increasing the data-efficiency, accuracy, and utility of learning-from-demonstration with multiple human demonstrators. We demonstrate this by using a PNT to learn kinesthetic robot table tennis demonstrations in Table 1. We received 40 demonstrations across 10 demonstrators, representing four different table tennis strikes. To clean the trajectory of the end effector prior to learning, we transformed our trajectories into a transformed three-dimensional space using Principal Components Analysis. Our data was then labeled by selecting the principal component in which the end effector moved most at each timestep ($|\mathcal{A}| = 6$). As seen in row 4 of Table 1, our approach outperforms all other benchmarks.

3 Sensitivity Analysis of PNTs

To analyze the sensitivity of our framework, we use our synthetic scheduling environment and perturb the amount of data available to the PNT and the amount of noise (correctness) within the data. To provide a thorough analysis, we validate our approach using k-fold cross-validation. This entails both choosing a different subset of data to learn from and perturbing different truth-values of state-actions pairs each fold.

As shown in Figure 1, our PNT is reasonably robust to noise for 2, 5, and 15 schedules as there is not a steep drop in accuracy. We do not see the typical trend where the effect of noise deteriorates as the amount of data increases. We posit the cause of this deviation as follows: As the number of demonstrators increases, the embedding space Ω of the PNT tends to represent a richer distribution. While the heterogeneity among the demonstrators may remain constant (represent the same number of modes), cases in which the PNT is unable to tease out the demonstrator mode from a single schedule are more likely (due to the increase in the number of schedules), leading to an embedding distribution with higher variance. Without noise, the PNT is able to make sense of the embedding space and learn with high performance; as the amount of noise increases, it is likely more difficult to

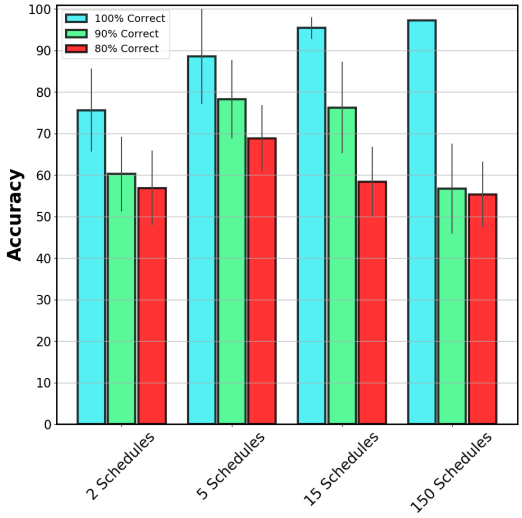


Figure 1: Sensitivity analysis in the synthethic scheduling environment.

represent demonstrators compactly within the embedding space. We posit that this increased variance within the embedding space caused by the combined effect of an increased number of demonstrators and noise leads to a reduction in performance when noise is held constant and the amount of data increases.

As expected, as the number of schedules increase, the PNTs have higher accuracy. However, from 15 to 150 schedules (a 10x magnitude increase in data), for the case of 100% correct data, there is only a $\sim 2\%$ increase in accuracy. This result provides support to the claim of data-efficiency in our apprenticeship scheduling framework.

4 Evidence Lower Bound

Here, we present the full derivation of the evidence lower bound (ELBO) that is used maximize the mutual information between ω and trajectories τ.

$$
\begin{aligned}
G(\omega;\tau) &= H(\omega) - H(\omega|\tau) \\
&= \mathbb{E}_{\omega\sim P(\omega), a_p^t \sim f^{PNT}_{\theta|\omega}}[logP(\omega|s_p^t, a_p^t)] + H(\omega) \\
&= \mathbb{E}_{a\sim f^{PNT}_{\theta|\omega}}[D_{KL}(log(P(\omega_p|s_p^t, a_p^t))||log(q^{\omega}_{\zeta|\theta}(s_p^t, a_p^t))) + \mathbb{E}_{\omega\sim P(\omega)} log(q^{\omega}_{\zeta|\theta}(s_p^t, a_p^t))] + H(\omega) \\
&\geq \mathbb{E}_{\omega_p\sim\mathcal{N}(\vec{\mu}_p,\vec{\sigma}_p^2), a\sim f^{PNT}_{\theta|\omega}}[log(q^{\omega}_{\zeta|\theta}(\omega_p|s_p^t, a_p^t))] + H(\omega) = L_G(f^{PNT}_{\theta|\omega}||q^{\omega}_{\zeta|\theta})
\end{aligned}
\tag{2}
$$

In our approach, we make use of continuous personalized embeddings which allow for greater expressivity in the embedding space, Ω. As such, we utilize a mean-squared error (MSE) loss between a sample from the approximate posterior (modeled as a normal distribution with constant variance) and the current embedding.

We present the approximate normal distribution, $\mathcal{N}_{q^{\omega}_{\zeta|\theta}}$, in Equation 3, where ω is the mean outputted by the posterior network, and σ is the standard deviation.

$$
\mathcal{N}_{q^{\omega}_{\zeta|\theta}} = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\frac{(x-\omega)^2}{\sigma^2}} \tag{3}
$$

Theorem 4.1. *Minimizing the mean-squared error between a sample from the approximate posterior and the current embedding is equivalent to maximizing the log-likelihood and therefore, the evidence lower bound.*

Proof. The mean-squared error (MSE) loss is $(x-\omega)^2$, where ω is the sample from the approximate posterior, and x is the current personalized embedding used to generate the predicted action. This is equivalent to the exponent numerator in $\mathcal{N}_{q^{\omega}_{\zeta|\theta}}$. With constant variance, the exponential function is

monotonic, and thus, minimizing the exponent will maximize the likelihood of the posterior. Thus, minimizing the MSE is equivalent to maximizing the likelihood of the posterior. This naturally extends to the multivariate case. □

5 Interpretability User Study Details

Here, we present the details of our novel user study to assess the interpretability of our discretized PNTs. We design an online questionnaire that asks users to predict a task to schedule given an input using a decision-tree based method and a neural-network-based method. Each user is randomly assigned a reasoning level, standard, pointwise, or counterfactual. Standard and counterfactual reasoning are discussed in the main paper. Pointwise reasoning outputs a probability of taking a certain action given a feature vector describing that action, x_a^t from state s^t, and a contextual feature vector capturing features common to all actions $\bar{x}^t$. We can generate pointwise features through Equation 4.

$$z^{t,p} := [\omega_p, \bar{x}^t, x_a^t], y_a^t = 1 \tag{4}$$

$$z^{t,p} := [\omega_p, \bar{x}^t, x_{a'}^t], y_{a'}^t = 0 \tag{5}$$

The tree and neural network-based models were trained under minimal sizes that were capable of achieving near-perfect accuracy. Tree models are learned PNTs, which are then discretized. The NN models are generated by appending personalized embeddings to a NN and following the training methodology described in Algorithm 1 from the main paper. Then, comparison weights and model weights for the discrete trees and neural networks, respectively, were rounded to the nearest 0.25. Rounding yielded $\sim 2\%$ loss in accuracy but allowed for the survey to be conducted within a reasonable time. For each type of decision-making framework, we provide instructions for how to utilize the framework to make a prediction. The order in which the user completes the neural network portion and decision tree portion is randomized. We explore additional hypothesis: counterfactual tree-based decision-making models are more interpretable **(H4)**, quicker to validate **(H5)**, and are more easily utilized **(H6)** than neural-network based models of any reasoning level. We then provide further comparisons between tree-based methods of different levels of reasoning.

We use four metrics throughout our user study: interpretability of the decision-making model, interpretability of the overall decision-making process, time to compute an output, and correctness. To verify **H4-H6**, we must compare the counterfactual discretized PNT to a standard neural network, pointwise neural network, and pairwise neural network. As the first case is shown in the paper (standard neural network vs. discretized PNT), we provide the results for the other two scenarios here.

6 Survey Results

Our IRB-approved anonymous survey was sent out to adult university students. We collected 35 responses, 14 of standard, 11 of pointwise, and 15 of counterfactual. We filter out responses that put in nonsensical answers (i.e., letters where numbers should be and repeated answers).

H4: In comparing a NN with pointwise reasoning to a discretized PNT, we test for normality and homoscedasticity and do not reject the null hypothesis in either case, using Shapiro-Wilk ($p > 0.9$ and $p > 0.3$) and Levene's Test ($p > 0.2$ and $p > 0.3$). We perform a paired t-test and find that counterfactual tree-based models were rated statistically significantly higher than pointwise neural networks on users' Likert scale ratings for model interpretability and overall process interpretability ($p < 0.05$ and $p < 0.01$). In comparing a NN with pairwise reasoning to a discretized PNT, we test for normality and homoscedasticity and do not reject the null hypothesis in either case, using Shapiro-Wilk ($p > 0.1$ and $p > 0.1$) and Levene's Test ($p > 0.4$ and $p > 0.4$). We perform a paired t-test and find that counterfactual tree-based models were rated statistically significantly higher than pointwise neural networks on users' Likert scale ratings for model interpretability and overall process interpretability ($p < 0.01$ and $p < .05$). These results support **H4**.

H5: In comparing a NN with pointwise reasoning to a discretized PNT, we perform a Wilcoxon signed-rank test on the per-model time to determine an output and find that tree-based models were not statistically significantly quicker to validate than neural networks ($p = 0.37$). In comparing a

NN with pairwise reasoning to a discretized PNT, we perform a Wilcoxon signed-rank test on the per-model time to determine an output and find that tree-based models were statistically significantly quicker to validate than neural networks ($p = 0.001$). This result provides partial support **H5**.

H6: In comparing a NN with pointwise reasoning to a discretized PNT, we perform a Wilcoxon signed-rank test on the per-model time to determine an output and find that tree-based models were statistically significantly achieved higher overall correctness scores compared to NN based models ($p < 0.05$), supporting **H6**. In comparing a NN with pairwise reasoning to a discretized PNT, we test for normality and homoscedasticity and do not reject the null hypothesis in either case, using Shapiro-Wilk ($p > 0.05$) and Levene's Test ($p > 0.2$). We perform a paired t-test and find that users using tree-based models statistically significantly achieved higher overall correctness scores compared to NN based models ($p < 0.001$), supporting **H6**.

7 Hyperparameters and Architecture Details

We compare our personalized apprenticeship scheduling approach to several baselines [2, 4, 6, 7, 10, 11]. Throughout this section, we will discuss the architecture, implementation details, and learning rates for all baselines and our algorithm in each domain. The runtime mentioned is in respect to a desktop with a NVIDIA RTX 2080Ti GPU and an Intel i7 processor.

7.1 Synthetic Low-Dimensional Environment

Each apprenticeship learning algorithm below is given 50 schedules to learn from and tests on a set of 50 unseen demonstrations.

- For the method of Sammut et al. [10], we utilize an multi-layer perceptron (MLP) with 3 linear layers connected by ReLU activation functions. After the last linear layer, we utilize a log softmax function to compute the log probability of which task to schedule. Each linear layer has 10 hidden units. We utilize the Adam optimizer with a learning rate of $1e^{-}3$. The runtime for training and verifying this model is under 30 minutes.
- For the method of Nikolaidis et al. [7], we utilize k-means clustering to separate the data into two clusters. Two neural networks (one for each cluster) are trained to imitate demonstrator data within the cluster. Each network utilizes the same architecture and learning rate used in the baseline of Sammut et al. [10]. The runtime for training and verifying this model is under 30 minutes.
- For the method of Li et al. [6], we utilize an simulator-free version of infoGAIL. The policy, discriminator, and approximate posterior are modeled by MLPs with 2 linear layers (32 hidden units) connected by a ReLU activation function, and an output activation function of a softmax, sigmoid, and softmax respectively. We initialize the number of discrete modes to 2. We utilize learning rates of $1e^{-}4, 1e^{-}3, 1e^{-}4$ respectively. For the hyperparameters of infoGAIL, we initialize λ_1 to 1, γ to 0.95, and λ_2 to 0. The runtime for training and verifying this model is under 30 minutes.
- For the method of Tamar et al. [11], we utilize a neural network with 3 linear layers (10, 2, 2 hidden units, respectively) connected by ReLU activation functions. We use N=5 samples as our hyperparameter to estimate the intention probability distribution $\mathcal{P}(z)$. We utilize a learning rate of $1e^{-}3$ alongside Stochastic Gradient Descent (SGD). The runtime for training and verifying this model is under 30 minutes.
- For the method of Hsiao et al. [4], we utilize a bidirectional LSTM with attention followed by a linear layer as specified in their paper. For the decoder, we utilize three linear layers connected by ReLU activation functions. We utilize a learning rate of $1e^{-}3$ alongside Stochastic Gradient Descent (SGD). The runtime for training and verifying this model is under 30 minutes.
- For the method of Gombolay et al. [2], we utilize a standard decision tree (counterfactuals are not possible when $|A| \leq 2$) of depth 10. The runtime for training and verifying this model is under 30 minutes.
- For our Personalized Neural Trees, we utilize a max depth of 6 (32 leaves) and embedding dimension of 2 ($d = 2$). We set learning rates of θ to $1e^{-}3$, ω to $1e^{-}2$, and ζ to $1e^{-}3$.

We find empirically that setting the learning rate of ω slightly higher allows for better LfD accuracy. For our approximate posterior, $q^{\omega}_{\zeta|\theta}$, we set the value of σ_p to zero. The runtime for training and verifying this model is under 30 minutes.

7.2 Synthetic Scheduling Environment

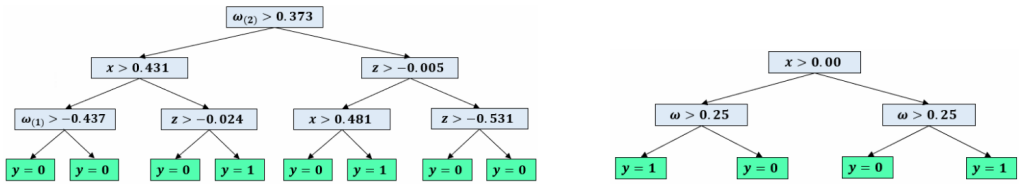
Each apprenticeship learning algorithm below is given 150 schedules to learn from and tests on a set of 100 unseen demonstrators.

- For the method of Sammut et al. [10], we utilize an multi-layer perceptron (MLP) with six linear layers connected by ReLU activation functions. After the last linear layer, we utilize a log softmax function to compute the log probability of which task to schedule. Each linear layers have 128, 128, 32, 32, 32, and 20 hidden units, respectively. We utilize the Adam optimizer with a learning rate of $1e^{-}4$. The runtime for training and verifying this model is approximately 30 minutes.
- For the method of Nikolaidis et al. [7], we utilize k-means clustering to separate the data into three clusters. Three neural networks (one for each cluster) are trained to imitate demonstrator data within the cluster. Each network utilizes the same architecture and learning rate used in the baseline of Sammut et al. [10]. The runtime for training and verifying this model is approximately 30 minutes.
- For the method of Li et al. [6], we again utilize a simulator-free version of infoGAIL. The policy follows the same network structure used in the Sammut et al. [10] baseline. The discriminator and approximate posterior are modeled by MLPs with six linear layers (128, 128, 128, 32, 32, 32 hidden units, respecitively) connected by a ReLU activation function, and an output activation function of a sigmoid, and softmax respectively. We initialize the number of discrete modes to 3. We utilize learning rates of $1e^{-}4, 1e^{-}3, 1e^{-}4$ respectively. For the hyperparameters of infoGAIL, we initialize λ_1 to 1, γ to 0.95, and λ_2 to 0. The runtime for training and verifying this model is approximately 24-48 hours.
- For the method of Tamar et al. [11], we utilize a neural network with 5 linear layers (128, 32, 32, 32, 32, 20, 2, 2 hidden units, respectively) connected by ReLU activation functions. We use N=5 samples as our hyperparameter to estimate the intention probability distribution $\mathcal{P}(z)$. We utilize a learning rate of $1e^{-}3$ alongside Stochastic Gradient Descent (SGD). The runtime for training and verifying this model is approximately 3 hours.
- For the method of Hsiao et al. [4], we utilize a bidirectional LSTM with attention followed by a linear layer as specified in their paper. For the decoder, we utilize six linear layers connected by ReLU activation functions. We utilize a learning rate of $1e^{-}3$ alongside Stochastic Gradient Descent (SGD). The runtime for training and verifying this model is approximately 3 hours.
- For the method of Gombolay et al. [2], we utilize a pairwise decision tree of depth 10. The counterfactuals are set to one-hot encodings of each action, as done in the original paper. The runtime for generating and verifying this model is approximately 5 minutes.
- For our Personalized Neural Trees, we utilize a max depth of six (32 leaves) and embedding dimension of 3 ($d = 3$). We set learning rates of θ to $1e^{-}2$, ω to $1e^{-}2$, and ζ to $1e^{-}2$. We find empirically that pretraining the policy network first and then adding in the posterior at a later epoch results in both good performance and mutual information maximization. This is opposed to training both models at once from scratch. For our approximate posterior, $q^{\omega}_{\zeta|\theta}$, we set the value of σ_p to zero. The runtime for training and verifying this model is approximately 24 hours.

7.3 Taxi Domain

Each apprenticeship learning algorithm below is given 25 successful trajectories from each user and tested on a set of 25 unseen trajectories from each demonstrator.

- For the method of Sammut et al. [10], we utilize the same architecture and learning rate as that of the synthetic scheduling environment. The runtime for training and verifying this model is approximately 30 minutes.



(a) Low-dim Environment (b) Survey Environment (counterfactual)

Figure 2: This figure depicts the learned PNT model after translation to an interpretable form.

- For the method of Nikolaidis et al. [7], we utilize k-means clustering to separate the data into three clusters. Three neural networks (one for each cluster) are trained to imitate demonstrator data within the cluster. Each network utilizes the same architecture and learning rate used in the baseline of Sammut et al. [10]. The runtime for training and verifying this model is approximately 30 minutes.
- For the method of Li et al. [6], we utilize the same architecture and learning rate as that of the synthetic scheduling environment. The runtime for training and verifying this model is approximately 24-48 hours.
- For the method of Tamar et al. [11], we utilize the same architecture and learning rate as that of the synthetic scheduling environment. The runtime for training and verifying this model is approximately 3 hours.
- For the method of Hsiao et al. [4], we utilize the same architecture and learning rate as that of the synthetic scheduling environment. The runtime for training and verifying this model is approximately 3 hours.
- For the method of Gombolay et al. [2], we utilize a pairwise decision tree of depth 13. The counterfactuals are set to one-hot encodings of each action, as done in the original paper. The runtime for generating and verifying this model is approximately 5 minutes.
- For our Personalized Neural Trees, we utilize a max depth of 8 (128 leaves) and embedding dimension of 3 ($d = 3$). As counterfactual task information is not readily available, we utilize one-hot encodings for each action. We set learning rates of θ to $1e^{-}2$, ω to $1e^{-}1$, and ζ to $1e^{-}2$. We find empirically that pretraining the policy network first and then adding in the posterior at a later epoch results in both good performance and mutual information maximization. For our approximate posterior, $q^{\omega}_{\zeta|\theta}$, we set the value of σ_p to zero. The runtime for training and verifying this model is approximately 12 hours.

8 Interpretable Models

As machine learning is being increasingly deployed into the real world, interpretability is required for these systems to gain human trust [1, 3, 8]. Interpretability refers to attempts that help the user understand why a machine learning model behaves the way it does. A clear visualization of a policy is one way to help a human form an accurate representation of its capabilities [9]. Furthermore, an interpretable model of resource allocation or planning tasks would be highly useful for a variety of reasons, from decision explanations to training purposes. In Figure 2, we display interpretable models generated through discretization for the low-dimensional environment and survey scheduling environment.

9 Future Work

During the deployment of a discretized PNT, we required pre-inferred embeddings to understand decision-maker behavior. As this involves a sample of the decision-maker's data and the use of backpropagation with a pre-discretized PNT to infer demonstrator style, we feel this can be improved by producing the demonstrator embedding through the means of our approximate posterior $q^{\omega}_{\zeta|\theta}$, modeled as a $PNT \setminus \omega$. This can be discretized following the framework of Section 4.3 of our paper, producing an interpretable model that predicts a mean and covariance of an embedding given a single state-action pair. This discretized posterior then takes in a state-action pair and produce the latent

embedding that generated this action. In this way, the interpretable discretized PNT has a method to naturally infer the demonstrator's embedding.

References

[1] Lisa Anne Hendricks, Ronghang Hu, Trevor Darrell, and Zeynep Akata. Grounding visual explanations. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 264–279, 2018.

[2] Matthew Gombolay, Reed Jensen, Jessica Stigile, Sung-Hyun Son, and Julie Shah. Decision-making authority, team efficiency and human worker satisfaction in mixed human-robot teams. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, New York City, NY, U.S.A., July 9-15 2016.

[3] Lisa Anne Hendricks, Ronghang Hu, Trevor Darrell, and Zeynep Akata. Generating counterfactual explanations with natural language. *arXiv preprint arXiv:1806.09809*, 2018.

[4] Fang-I Hsiao, Jui-Hsuan Kuo, and Min Sun. Learning a multi-modal policy via imitating demonstrations with mixed behaviors. *ArXiv*, abs/1903.10304, 2019.

[5] G. Ayorkor Korsah. *Exploring bounded optimal coordination for heterogeneous teams with cross-schedule dependencies*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, January 2011.

[6] Yunzhu Li, Jiaming Song, and Stefano Ermon. Infogail: Interpretable imitation learning from visual demonstrations. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3812–3822. Curran Associates, Inc., 2017.

[7] Stefanos Nikolaidis, Ramya Ramakrishnan, Keren Gu, and Julie Shah. Efficient model learning from joint-action demonstrations for human-robot collaborative tasks. In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*, HRI '15, pages 189–196, New York, NY, USA, 2015. ACM.

[8] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 3(3):e10, 2018.

[9] Steffi Paepcke and Leila Takayama. Judging a bot by its cover: An experiment on expectation setting for personal robots. In *Proceedings of the 5th ACM/IEEE International Conference on Human-robot Interaction*, HRI '10, pages 45–52, Piscataway, NJ, USA, 2010. IEEE Press. ISBN 978-1-4244-4893-7.

[10] Claude Sammut, Scott Hurst, Dana Kedzier, and Donald Michie. *Learning to Fly*, page 171–189. MIT Press, Cambridge, MA, USA, 2002.

[11] Aviv Tamar, Khashayar Rohanimanesh, Yinlam Chow, Chris Vigorito, Ben Goodrich, Michael Kahane, and Derik Pridmore. Imitation learning from visual data with multiple intentions. In *International Conference on Learning Representations*, 2018.