Task Programming: Learning Data Efficient Behavior Representations

Jennifer J. Sun[1] Ann Kennedy[2] Eric Zhan[1] David J. Anderson[1] Yisong Yue[1] Pietro Perona[1]
[1]Caltech [2]Northwestern University

Code & Project Website: https://sites.google.com/view/task-programming

Abstract

Specialized domain knowledge is often necessary to accurately annotate training sets for in-depth analysis, but can be burdensome and time-consuming to acquire from domain experts. This issue arises prominently in automated behavior analysis, in which agent movements or actions of interest are detected from video tracking data. To reduce annotation effort, we present TREBA: a method to learn annotation-sample efficient trajectory embedding for behavior analysis, based on multi-task self-supervised learning. The tasks in our method can be efficiently engineered by domain experts through a process we call "task programming", which uses programs to explicitly encode structured knowledge from domain experts. Total domain expert effort can be reduced by exchanging data annotation time for the construction of a small number of programmed tasks. We evaluate this trade-off using data from behavioral neuroscience, in which specialized domain knowledge is used to identify behaviors. We present experimental results in three datasets across two domains: mice and fruit flies. Using embeddings from TREBA, we reduce annotation burden by up to a factor of 10 without compromising accuracy compared to state-of-the-art features. Our results thus suggest that task programming and self-supervision can be an effective way to reduce annotation effort for domain experts.

1. Introduction

Behavioral analysis of one or more agents is a core element in diverse fields of research, including biology [36, 26], autonomous driving [6, 39], sports analytics [42, 43], and video games [20, 3]. In a typical experimental workflow, the location and pose of agents is first extracted from each frame of a behavior video, and then labels for experimenter-defined behaviors of interest are applied on a frame-by-frame basis based on the pose and movements of the agents. In addition to reducing human effort, automated quantification of behavior can lead to more objective, precise, and scalable measurements compared to manual annotation [1, 10]. However, training behavior detection models can be data intensive and manual behavior annotation often requires specialized domain knowledge and high-frequency temporal labels. As a result, this process of generating training datasets is time-consuming and effort-intensive for experts. Therefore, methods to reduce annotation effort by domain experts are needed to accelerate behavioral studies.

We study alternative ways for domain experts to improve

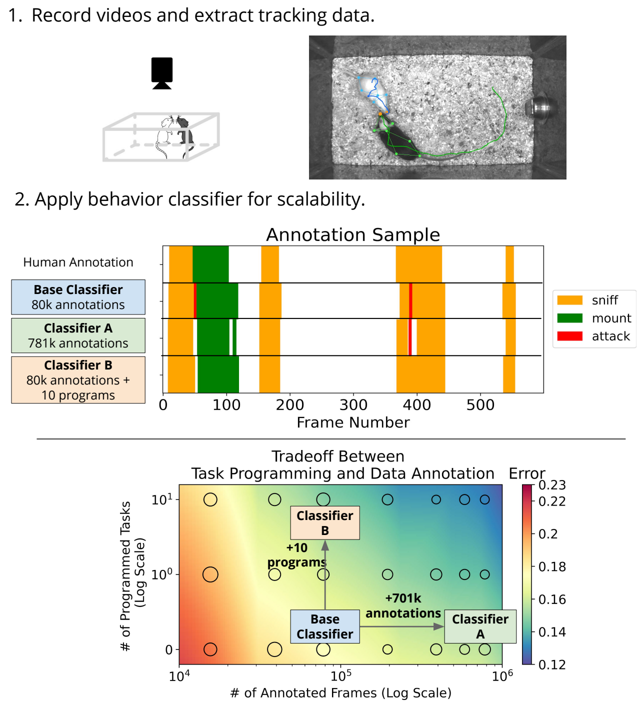

Figure 1. **Overview of our approach.** *Part 1:* A typical behavior study starts with extraction of tracking data from videos. We show 7 keypoints for each mouse, and draw the trajectory of the nose keypoint. *Part 2:* Domain experts can either do data annotation (Classifier A) or task programming (Classifier B) to reduce classifier error. The middle panel shows annotated frames at 30Hz. Colors in the bottom plot represent interpolated performance based on classifier error at the circular markers (full results in Section 4.3). The size of the marker represents the error variance.

Correspondence to jjsun@caltech.edu.

classifier accuracy beyond simply increasing the sheer volume of annotations. In particular, we propose a framework that unifies: (1) self-supervised representation learning, and (2) encoding explicit structured knowledge on trajectory data using expert-defined programs. Domain experts can construct these programs efficiently because keypoint trajectories in each frame are typically low dimensional, and experts can already hand-design effective features for trajectory data [36, 28]. To best leverage this structured expert knowledge, we develop a framework to learn trajectory representations based on multi-task self-supervised learning, which has not been well-explored for trajectory data.

Our Approach. Our framework, **Tr**ajectory **E**mbedding for **B**ehavior **A**nalysis (TREBA), learns trajectory representations through trajectory generation alongside a set of decoder tasks based on expert-engineered programs. These programs are created by domain experts through a process we call task programming, inspired by the data programming paradigm [33]. Task programming is a process by which domain experts identify trajectory attributes relevant to the behaviors of interest under study, write programs, and apply those programs to inform representation learning (Section 3.2). This flexibility in decoder tasks allows our framework to be applicable to a variety of agents and behaviors studied across diverse fields of research.

Expert Effort Tradeoffs. Since task programming will typically require a domain expert's time, we study the tradeoff between doing task programming and data annotation. We compare behavior classification performance with different amounts of annotated training data and programmed tasks. For example, for the domain illustrated in Figure 1, domain experts can reduce error by 13% relative to the base classifier by annotating 701k additional frames, or they can reduce error by 16% by learning a representation using 10 programmed tasks in our framework. Our approach allows experts to trade a large number of annotations for a small number of programmed tasks.

We study our approach across two domains in behavioral neuroscience, namely mouse and fly behavior. We chose this setting because it requires specialized domain knowledge for data annotation, and data efficiency is important for domain experts. Furthermore, decoder tasks in our framework can be efficiently programmed by experts based on simple functions describing trajectory attributes for identifying behaviors of interest. For example, for mouse social behaviors such as attack [36], important behavior attributes include the speed of each mouse and distance between mice. The corresponding task could then be to decode these attributes from the learned representations.

Our contributions are:

- We introduce task programming as an efficient way for domain experts to reduce annotation effort and encode structural knowledge. We develop a novel method to learn an annotation-sample efficient trajectory representation using self-supervision and programmatic supervision.
- We study the effect of task programming, data annotation, and different decoder losses on behavior classifier performance.
- We demonstrate these representations on three datasets in two domains, showing that our method can lead to a $10\times$ annotation reduction for mice, and $2\times$ for flies.

2. Related Work

Behavior Modeling. Behavior modeling using trajectory data is studied across a variety of fields [26, 6, 39, 42, 20, 3]. In particular, there is an increasing effort to automatically detect and classify behavior from trajectory data [23, 1, 14, 27, 13, 36]. Our experiments are based on behavior classification datasets from behavioral neuroscience [15, 4, 36], a field where specialized domain knowledge is important for identifying behaviors of interest.

The behavior analysis pipeline generally consists of the following steps: (1) tracking the pose of agents, (2) computing pose-based features, and (3) training behavior classifiers [4, 21, 36, 28]. To address step 1, there are many existing pose estimation models [15, 27, 18, 36]. In our work, we leverage two existing pose models, [36] for mice and [15] for flies, to produce trajectory data. In steps 2 and 3 of the typical behavior analysis pipeline, hand-designed trajectory features are computed from the animals' pose, and classifiers are trained to predict behaviors of interest in a fully supervised fashion [4, 21, 15, 36]. Training fully supervised behavior classifiers requires time-consuming annotations by domain experts [1]. Instead, our proposed approach enables domain experts to trade time-consuming annotation work for task programming with representation learning.

Another group of work uses unsupervised methods to discover new motifs and behaviors [22, 41, 2, 26, 5]. Our work focuses on the more common case where domain experts already know what types of actions they would like to study in an experiment. We aim to improve the data-efficiency of learning expert-defined behaviors.

Representation Learning. Visual representation learning has made great progress in effective representations for images and videos [17, 16, 7, 29, 25, 19, 38]. Self-supervised signals are often used to train this visual representation, such as learning relative positions of image patches [11], predicting image rotations [16], predicting future patches [29], and constrastive learning on augmented images [7]. Compared to visual data, trajectory data is significantly lower dimensional in each frame, and techniques from visual representation learning often cannot be applied directly. For example, while we can create image patches that represent the same visual class, it is difficult to select

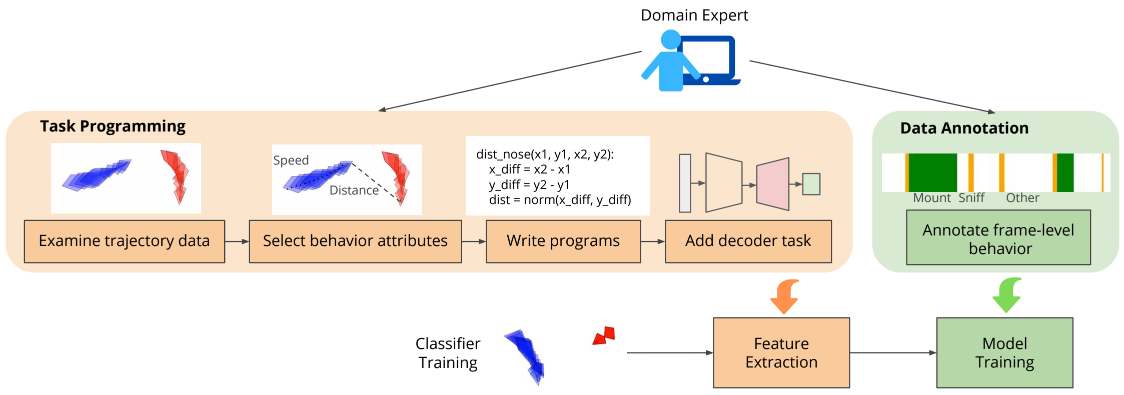

Figure 2. **Task Programming and Data Annotation for Classifier Training.** Domain experts can choose between doing task programming and/or data annotation. Task programming is the process for domain experts to engineer decoder tasks for representation learning. The programs enable learning of annotation-sample efficient trajectory features to improve performance instead of additional annotations.

a partial set of keypoints that represent the same behavior. Our framework builds upon these approaches to learn effective representations for behavioral data.

We investigate different decoder tasks in order to learn an effective behavior representation. One decoder task that we investigate is self-decoding: the reconstruction of input trajectories using generative modeling. Generative modeling has previously been applied to learn representations for visual data [45, 38, 29] and language modeling [31]; for trajectory data, we use imitation learning [40, 44, 43] to train our trajectory representation. The other tasks in our multi-task self-supervised learning framework are created by domain experts using task programming (Section 3.2). This idea of using a human-provided function as part of training has been studied for training set creation [33, 32], and controllable trajectory generation [43]. Our work explores these additional decoder tasks to further improve the learned representation over the generative loss alone.

Multi-Task Self-Supervised Learning. We jointly optimize a family of self-supervised tasks in an encoder-decoder setup, making this work an example of multi-task self-supervised learning. Multi-task self-supervised learning has been applied to other domains such as visual data [12, 25], accelerometer recordings [35], audio [34] and multi-modal inputs [37, 30]. Generally in each of these domains, tasks are defined ahead of time, as is the case for tasks such as frame reconstruction, colorization, finding relative position of image patches, and video-audio alignment. Most of these tasks are designed for image or video data, and cannot be applied directly to trajectory data. To construct tasks for trajectory representation learning, we propose that domain experts can use task programming to engineer decoder tasks and encode structural knowledge.

3. Methods

We introduce **Tra**jectory **E**mbedding for **B**ehavior **A**nalysis (TREBA), a method to learn an annotation-sample efficient trajectory representation using self-supervision and auxiliary decoder tasks engineered by domain experts. Figure 2 provides an overview of the expert's role. In our framework, domain experts replace (a significant amount of) time-consuming manual annotation with the construction of a small number of programmed tasks, reducing total expert effort. Each task places an additional constraint on the learned trajectory embedding.

TREBA uses the expert-programmed tasks based on a multi-task self-supervised learning approach, outlined in Figure 3. To learn task-relevant low-dimensional representations of pose trajectories, we train a network jointly on (1) reconstruction of the input trajectory (Section 3.1) and (2) expert-programmed decoder tasks (Section 3.3). The learned representation can then be used as input to behavior modeling tasks, such as behavior classification.

3.1. Trajectory Representations

Let $\mathcal{D}$ be a set of N unlabelled trajectories. Each trajectory τ is a sequence of states $\tau = \{(s_t)\}_{t=1}^{T}$, where the state s_i at timestep i corresponds to the location or pose of the agents at that timestep. In this study, we divide trajectories from longer recordings into segments of length T, but in general trajectory length can vary. For multiple agents, the keypoints of each agent is stacked at each timestep.

Before we introduce our expert-programmed tasks, we will use trajectory reconstruction as an initial self-supervised task. Given a history of agent states, we would like our model to predict the next state. This task is usually studied with sequential generative models. We used trajectory variational autoencoders (TVAEs) [9, 43] to embed the input trajectory using an RNN encoder, q_ϕ, and an RNN decoder, p_θ, to predict the next state. The TVAE loss is:

$$\mathcal{L}^{\text{tvae}} = \mathbb{E}_{q_\phi}\left[\sum_{t=1}^{T} - \log(p_\theta(s_{t+1}|s_t, \mathbf{z}))\right] + D_{KL}(q_\phi(\mathbf{z}|\tau)||p_\theta(\mathbf{z})). \tag{1}$$

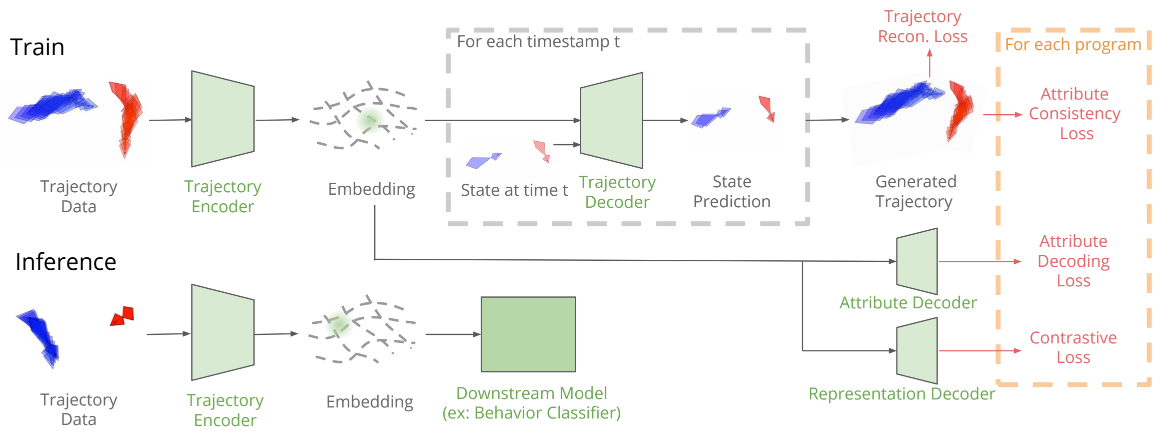

Figure 3. **TREBA Training and Inference Pipelines.** During training, we use trajectory self-decoding and the programmed decoder tasks to train the trajectory encoder. The learned representation is used for downstream tasks such as behavior classification.

We use a prior distribution $p_\theta(\mathbf{z})$ on $\mathbf{z}$ to regularize the learned embeddings; in this study, our prior is the unit Gaussian. By optimizing for the TVAE loss only, we learn an unsupervised version of TREBA. When performing subsequent behavior modeling tasks such as classification, we use the embedding mean, $\mathbf{z}_\mu$.

3.2. Task Programming

Task programming is the process by which domain experts create decoder tasks for trajectory self-supervised learning. This process consists of selecting attributes from trajectory data, writing programs, and creating decoder tasks based on the programs (Figure 2). Here, domain experts are people with specialized knowledge for studying behavior, such as neuroscientists or sports analysts.

To start, domain experts identify attributes from trajectory data relevant to the behaviors of interest under study. Behavior attributes capture information that is likely relevant to agent behavior, but is not explicitly included in the trajectory states $\{(s_t)\}_{t=1}^T$. These attributes represent structured knowledge that domain experts are implicitly or explicitly considering for behavior analysis, such as the distance between two agents, agent velocity, or the relative positioning of agent body parts.

Next, domain experts write a program to compute these attributes on trajectory data, which can be done with existing tools such as MARS [36] or SimBA [28]. Algorithm 1 shows a sample program from the mouse social behavior domain, for measuring the "facing angle" between a pair of interacting mice. Each program can be used to construct decoder tasks for self-supervised learning (Section 3.3).

Our framework is inspired by the data programming paradigm [33], which applies programs to training set creation. In comparison, our framework uses task programming to unify expert-engineered programs, which encode structured expert knowledge, with representation learning.

Algorithm 1: Sample Program for Facing Angle

```
Input: centroid of mouse 1 $(x_1, y_1)$, centroid of
  mouse 2 $(x_2, y_2)$, heading of mouse 1 $(\phi_1)$
$x_{\text{diff}} = x_2 - x_1$
$y_{\text{diff}} = y_2 - y_1$
$\theta = \arctan(y_{\text{diff}}, x_{\text{diff}})$
Return $\theta - \phi_1$
```

Domain	Behavior Attributes
Mouse	Facing Angle Mouse 1 and 2, Speed Mouse 1 and 2 Nose-Nose Distance, Nose-Tail Distance, Head-Body Angle Mouse 1 and 2 Nose Movement Mouse 1 and 2
Fly	Speed Fly 1 and 2, Fly-Fly Distance Angular Speed Fly 1 and 2, Facing Angle Fly 1 and 2 Min and Max Wing Angles Fly 1 and 2 Major/Minor Axis Ratio Fly 1 and 2

Table 1. **Behavior Attributes used in Task Programming.** We base our programmed tasks in our experiments on these behavior attributes from domain experts in each domain.

Working with domain experts in behavioral neuroscience, we created a set of programs to use in studying our approach. The selected programs are a subset of behavior attributes in [36] (for mouse datasets) and a subset of behavior attributes in [15] (for fly datasets). We list the programs used in Table 1, and provide more details about the programs in the Supplementary Material.

3.3. Learning Algorithm

We develop a method to incorporate the programs from domain experts as additional learning signals for TREBA. We consider the following three approaches: (1) enforcing attribute consistency in generated trajectories (Section 3.3.1), (2) performing attribute decoding directly (Section 3.3.2), (3) applying contrastive loss based on program supervision (Section 3.3.3). Each of these methods applies a different loss on the low-dimensional representation $\mathbf{z}$ of trajectory τ. Any combinations of these decoding tasks can be combined with self-decoding from Section 3.1 to inform the trajectory embedding $\mathbf{z}$.

3.3.1 Attribute Consistency

Let λ be a set of M domain-expert-designed functions measuring agent behavior attributes, such as agent velocity or facing angle. Recall that each $\lambda_j, j = 1...M$ takes as input a trajectory τ, and returns some expert-designed attribute $\lambda_j(\tau)$ computed from that trajectory. For λ_j designed for a single frame, we apply the function to the center frame of τ. Attribute consistency aims to maintain the same behavior attribute labels for the generated trajectory as the original. Let $\tilde{\tau}$ be the trajectory generated by the TVAE given the same initial condition as τ and encoding $\mathbf{z}$.The attribute consistency loss is:

$$\mathcal{L}^{\text{attr}} = \mathbb{E}_{\tau\sim\mathcal{D}}\left[\sum_{j=1}^{M} \mathbb{1}(\lambda_j(\tilde{\tau}) \neq \lambda_j(\tau))\right]. \qquad (2)$$

Here, we show the loss for categorical λ_j, but in general, λ_j can be continuous and any loss measuring differences between $\lambda_j(\tilde{\tau})$ and $\lambda_j(\tau)$ applies, such as mean squared error. We do not require λ to always be differentiable, and we use the differentiable approximation introduced in [43] to handle non-differentiable λ.

3.3.2 Attribute Decoding

Another option is to decode each attribute $\lambda_j(\tau)$ directly from the learned representation $\mathbf{z}$. Here we apply a shallow decoder f to the learned representation, with decoding loss:

$$\mathcal{L}^{\text{decode}} = \mathbb{E}_{\tau\sim\mathcal{D}}\left[\sum_{j=1}^{M} \mathbb{1}(f(q_\phi(\mathbf{z}_\mu|\tau)) \neq \lambda_j(\tau))\right]. \qquad (3)$$

Similar to Eq. (2), we show the loss for categorical λ_j, however any type of λ may be used.

3.3.3 Contrastive Loss

Lastly, the programmed tasks can be used to supervise contrastive learning of our representation. For a trajectory τ_i, and for each λ_j, positive examples are those trajectories with the same attribute class under λ_j. For λ_j with continuous outputs, we create a discretized $\hat{\lambda}_j$ in which we apply fixed thresholds to divide the output space into classes. For our work, we apply two thresholds for each program such that our classes are approximately equal in size.

We apply a shallow decoder g to the learned representation, and let $\mathbf{g} = g(q_\phi(\mathbf{z}_\mu|\tau))$ represent the decoded representation. We then apply the contrastive loss:

$$\mathcal{L}^{\text{cntr.}} = \sum_{i=1}^{B}\sum_{j=1}^{M}\left[\frac{-1}{N_{pos(i,j)}}\sum_{k=1}^{B} \mathbb{1}_{i\neq k}\cdot \mathbb{1}_{\lambda_j(\tau_i)=\lambda_j(\tau_k)} \cdot \log\frac{\exp(\mathbf{g}_i\cdot\mathbf{g}_k/t)}{\sum_{l=1}^{N}\mathbb{1}_{i\neq l}\cdot\exp(\mathbf{g}_i\cdot\mathbf{g}_l/t)}\right], \qquad (4)$$

where B is the batch size, $N_{pos(i,j)}$ is the number of positive matches for τ_i with λ_j, and $t > 0$ is a scalar temperature parameter. Our form of contrastive loss supervised by task programming is similar to the contrastive loss in [24] supervised by human annotations. A benefit of task programming is that the supervision from programs can be quickly and scalably applied to unlabelled datasets, as compared to expert supervision which can be time-consuming. We note that the unsupervised version of this contrastive loss is studied in [7], based on previous works such as [29].

3.3.4 Data Augmentation

We can perform data augmentation on trajectory data based on our expert-provided programs. Given the set of all possible augmentations, we define Λ to be the subset of augmentations that are *attribute-preserving*: that is, for all λ_j in the set of programs, $\lambda_j(\tau) = \lambda_j(\Lambda_m(\tau))$ for some augmentation $\Lambda_m \in \Lambda$. An example of a valid augmentation in the mouse domain is reflection of the trajectory data.

All losses presented above can be extended with data augmentation, by replacing τ with $\Lambda_m(\tau)$ in losses. For contrastive loss, adding data augmentation corresponds to extending the batch size to $2B$, with B samples from the original and augmented trajectories.

The augmentations we use in our experiments are reflections, rotations, translations, and a small Gaussian noise on the keypoints (mouse data only). In practice, we add the loss for each decoder with and without data augmentation.

4. Experiments

4.1. Datasets

We work with datasets from behavioral neuroscience, where there are large-scale, expert-annotated datasets from scientific experiments. We study behavior for the laboratory mouse and the fruit fly, two of the most common model organisms in behavioral neuroscience. For each organism, we first train TREBA using large unannotated datasets: for the mouse domain we use an in-house dataset comprised of approximately 100 hours of recorded diadic social interactions (**Mouse100**), while for the fly domain we use the **Fly vs. Fly** dataset [15] without annotations.

After pre-training TREBA, we evaluate the suitability of our trajectory representation for supervised behavior classification (classifying frame-level behaviors on continuous trajectory data), on three additional datasets:

MARS. The MARS dataset [36] is a recently released mouse social behavior dataset collected in the same conditions as Mouse100. The dataset is annotated by neurobiologists on a frame-by-frame basis for three behaviors: sniff, attack, and mount. We use the provided train, validation, and test split (781k, 352k, and 184k frames respectively). Trajectories are extracted by the MARS tracker [36].

CRIM13. CRIM13 [4] is a second mouse social behavior dataset manually annotated on a frame-by-frame basis by experts. To extract trajectories, we use a version of the the MARS tracker [36] fine-tuned on pose annotations on CRIM13. We select a subset of videos from which trajectories can be reliably detected for a train, validation and test split of 407k, 96k, and 142k frames respectively. We evaluated classifier performance on the same three behaviors studied in MARS (sniff, attack, mount).

CRIM13 is a useful test of the robustness of TREBA trained on Mouse100, as the recording conditions in CRIM13 (image resolution 640×480, frame rate 25Hz, and non-centered cage location) are different from those of Mouse100 (image resolution 1024×570, frame rate 30Hz, and centered cage location).

Fly vs. Fly (Fly). We use the Aggression and Courtship videos from the Fly dataset [15]. These videos record interactions between a pair of flies annotated on a frame-by-frame basis for social behaviors by domain experts. Our train, validation and test split has 1067k, 162k, 322k frames respectively. We use the trajectories tracked by [15] and evaluate on all behaviors with more than 1000 frames of annotations in the full training set (lunge, wing threat, tussle, wing extension, circle, copulation).

4.2. Training and Evaluation Procedure

We use the attribute consistency loss (Section 3.3.1) and contrastive loss (Section 3.3.3) to train TREBA using programs. With the same programs, we find that different loss combinations result in similar performance, and that the combination of consistency and contrastive losses performs the best overall. The results for all loss combinations are provided in the Supplementary Material.

For the datasets in the mouse domain (MARS and CRIM13) we train TREBA on Mouse100, with 10 programs provided by mouse behavior domain experts. For the Fly dataset, we train TREBA on the training split of Fly without annotations, with 13 programs provided by fly behavior domain experts. The full list is in Table 1. We then use the trained encoder, with pre-trained frozen weights, as a trajectory feature extractor over $T = 21$ frames, where the representation for each frame is computed using ten frames before and after the current frame.

We evaluate our classifiers, with and without TREBA features, using Mean Average Precision (MAP). We compute the mean over behaviors of interest with equal weighting. Our classifiers are shallow fully-connected neural networks on the input features. To determine the relationship between classifier performance and training set size, we sub-sample the training data by randomly sampling trajectories (with lengths of 100 frames) to achieve a desired fraction of the training set size. Sampling was performed to achieve a similar class distribution as the full training set. We train each classifier nine times over three different random selections of the training data for each training fraction (1%, 2%, 5%, 10%, 25%, 50%, 75%, 100%). Additional implementation details are in the Supplementary Material.

4.3. Main Results

We evaluate the data efficiency of our representation for supervised behavior classification, by training a classifier to predict behavior labels given both our learned representation and one of either (1) raw keypoints or (2) domain-specific features designed by experts. The TREBA+keypoints evaluation allows us to test the effectiveness of our representation without other hand-designed features, while the TREBA+features evaluation is closer to most potential use cases. The domain-specific features for mice are the trajectory features from [36] and features for flies are the trajectory features from [4]. The input features are a superset of the programs we use in Table 1.

Our representation is able to improve the data efficiency for both keypoints and domain-specific features, over all evaluated amounts of training data availability (Figure 4). We discuss each dataset below:

MARS. Our representation significantly improves classification performance over keypoints alone (Figure 4 A1). We achieve the same performance as the full baseline training using only between 1% and 2% of the data. While this result is partially because our representation contains temporal information, we can also observe a significant increase in data efficiency in A2 compared to domain-specific features, which also contains temporal features. Classifiers using TREBA has the same performance as the full baseline training set with around 5% $\sim$ 10% of data (i.e., $10\times \sim 20\times$ improved annotation efficiency).

CRIM13. We test the transfer learning ability of our representation on CRIM13, a dataset with different image properties than Mouse100, the training set of TREBA. Our representation achieves the same performance as the baseline training with keypoints using around 5% to 10% of the training data (Figure 4 B1). With domain-specific features, TREBA uses 50% of the data annotation to have the same performance as the full training baseline (i.e., $2\times$ improved annotation efficiency). Our representation is able to generalize to a different dataset of the same organism.

Fly. When using keypoints only, our representation re-

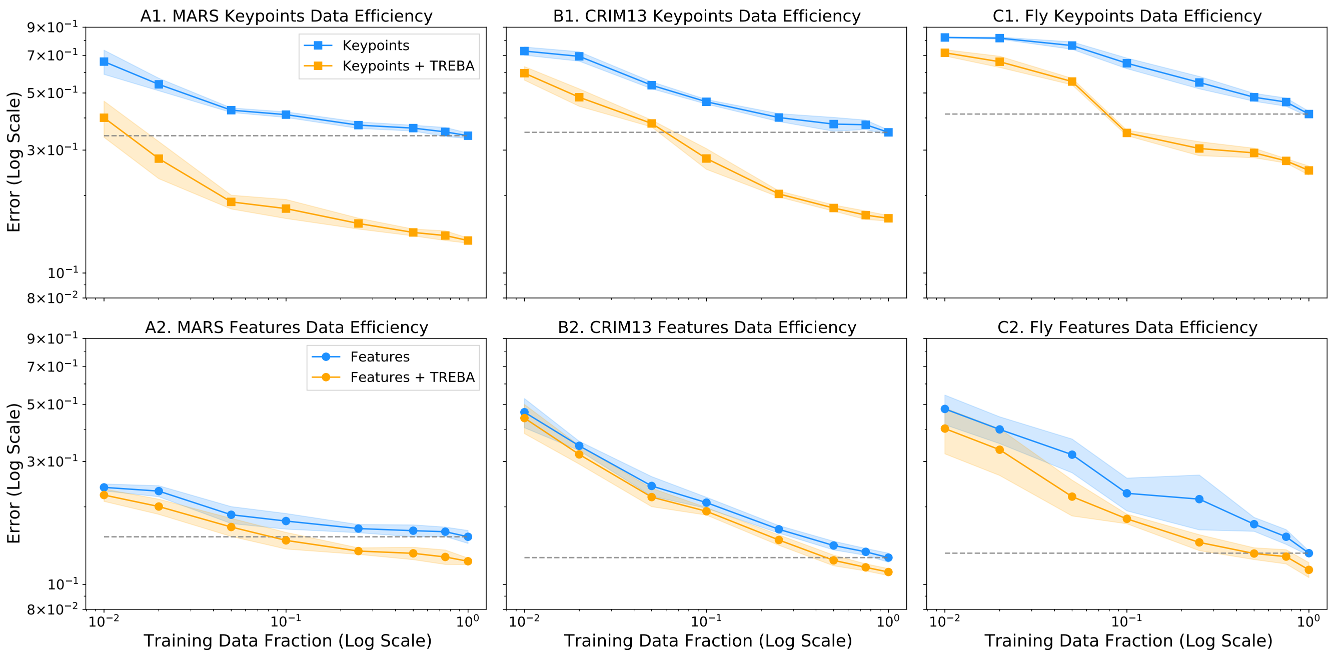

Figure 4. **Data Efficiency for Supervised Classification.** Training data fraction vs. classifier error on MARS (left), CRIM13 (middle) and fly (right). The blue lines represent performance with baseline keypoints and features, and the orange lines are with TREBA. The shaded regions correspond to the classifier standard deviation over nine repeats. The gray dotted line marks the best observed classifier performance when trained on the baseline features (using the full training set). Note the log scale on both the x and y axes.

quires 10% of the data (Figure 4 C1) and for features, our representation requires 50% of the data (Figure 4 C2) to achieve the same performance as full baseline training. This corresponds to 2× improved annotation efficiency.

4.4. Model Ablations

We perform the following model ablations to better characterize our approach. In this section, percentage error reduction relative to baseline is averaged over all training fractions. Additional results are in the Supplementary Material.

Varying Programmed Tasks. We test the performance of TREBA trained with each single program provided by the domain experts in Table 1, and the average, best, and worst performance is visualized in Figure 5. On average, representations learned from a single program is better than using features alone, but using all provided programs further improves performance.

For a single program, there could be a large variation in performance depending on the selected program (Figure 5). While the best performing single program is close in classifier MAP to using all programs, the worst performing program may increase error, as in MARS and CRIM13. We further tested the performance using more programs.

In the mouse domain, we found that with three randomly selected programs, the variation between runs is much smaller compared to single programs (Supplementary Material). With three programs, we achieve comparable average error reduction from baseline features to using all programs (MARS: 14.6% error reduction for 3 programs vs. 15.3% for all, CRIM13: 9.2% for 3 programs vs. 9.5% for all). For the fly domain, we found that we needed seven programs to achieve comparable performance (20.7% for 7 programs vs. 21.2% for all).

Varying Decoder Losses. When the programmed tasks are fixed, decoder losses with different combinations of consistency (Section 3.3.1), decoding (Section 3.3.2), and contrastive (Section 3.3.3) loss are similar in performance (Supplementary Material). Additionally, we evaluate the TREBA framework without programmed tasks, with decoder tasks using trajectory generation and unsupervised contrastive loss. While self-supervised representations are also effective at reducing baseline error, we achieve the best classifier performance using TREBA with programmed tasks (Table 2). Furthermore, we found that training trajectory representations without self-decoding, using the contrastive loss from [7, 8], resulted in less effective representations for classification (Supplementary Material).

Data Augmentation. We removed the losses using the data augmentation described in Section 3.3.4, and found that performance was slightly lower for all datasets than with augmentation. In particular, adding data augmentation decreases error by 1.2% on MARS, 2.5% on CRIM13, and 5.3% on Fly compared to without data augmentation.

Pre-Training Variations The results shown for MARS was obtained with pre-training TREBA on Mouse100, a large in-house mouse dataset with the same image prop-

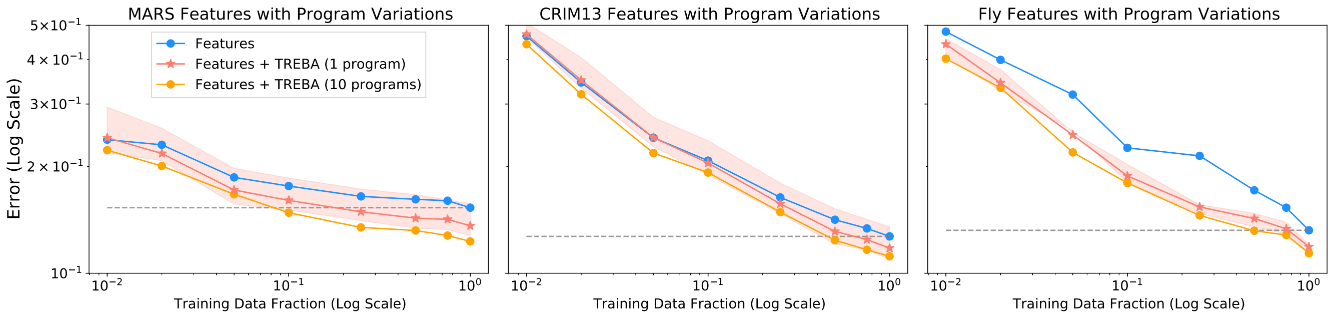

Figure 5. **Varying Programmed Tasks.** Effect of varying number of programmed tasks on classifier data efficiency. The shaded region corresponds to the best and worst classifiers trained using a single programmed task from Table 1. The grey dotted line corresponds to the value where the baseline features achieve the best performance (using the full training set).

	Keypoint Error Reduction (%)		
Decoder Loss	MARS	CRIM13	Fly
TVAE	52.2 ± 4.0	34.7 ± 1.5	15.4 ± 2.1
TVAE+ Unsup. Contrast	52.6 ± 3.9	37.4 ± 2.4	20.9 ± 1.7
TVAE+ Contrast+Consist	$\mathbf{55.1 \pm 3.0}$	$\mathbf{41.1 \pm 2.1}$	$\mathbf{33.7 \pm 1.2}$

	Features Error Reduction (%)		
Decoder Loss	MARS	CRIM13	Fly
TVAE	13.7 ± 1.8	8.2 ± 4.6	11.7 ± 4.7
TVAE+ Unsup. Contrast	14.3 ± 2.2	8.9 ± 4.1	16.1 ± 1.7
TVAE+ Contrast+Consist	$\mathbf{15.3 \pm 2.1}$	$\mathbf{9.5 \pm 3.8}$	$\mathbf{21.2 \pm 4.5}$

Table 2. **Decoder Error Reductions.** Percentage error reduction relative to baseline keypoints and domain-specific features for training with different decoder losses for TREBA. The average is taken over all evaluated training fractions.

erties as MARS. Figure 6 demonstrates the effect of varying TREBA training data amount with TVAE only and with programs. For both keypoints and features, we observe that TVAE (MARS) has the largest error. We see that error can be decreased by either adding more data (features + TVAE (Mouse100) with 3.9% decrease) or adding task programming (features + Programs (MARS) with 4.4% decrease). Adding both more data and task programming results in an average decrease of 5.7% error relative to TVAE (MARS) and the lowest average error.

5. Conclusion

We introduce a method to learn an annotation-sample efficient Trajectory Embedding for Behavior Analysis (TREBA). To train this representation, we study self-supervised decoder tasks as well as decoder tasks with programmatic supervision, the latter created using task programming. Our results show that TREBA can reduce annotation requirements by a factor of 10 for mice and 2 for flies. Our experiments on three datasets (two in mice and one in fruit flies) suggest that our approach is effective across different domains. TREBA is not restricted to animal behavior

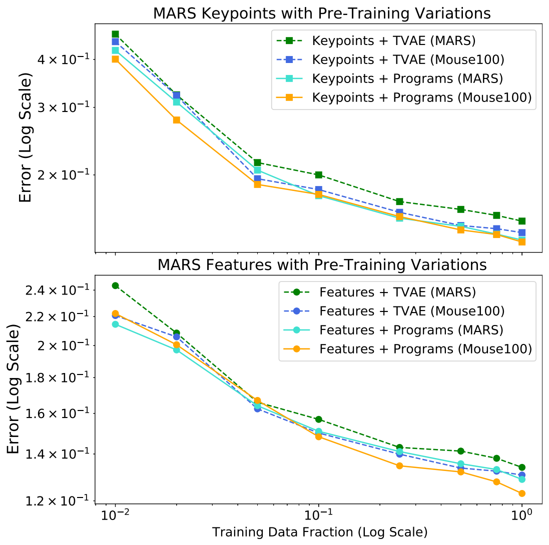

Figure 6. **Pre-Training Data Variations.** Effect of varying pre-training data on classifier data efficiency for the MARS dataset. "TVAE" corresponds to training TREBA with TVAE losses only, and "Programs" corresponds to training with all programs.

and may be applied to other domains where tracking data is expensive to annotate, such as in sports analytics.

Our experiments highlight, and quantify, the tradeoff between task programming and data annotation. The choice of which is more effective will depend on the cost of annotation and the level of expert understanding in identifying behavior attributes. Directions in creating tools to facilitate program creation and data annotation will help further accelerate behavioral studies.

6. Acknowledgements

We would like to thank Tomomi Karigo at Caltech for providing the mouse dataset. The Simons Foundation (Global Brain grant 543025 to PP) generously supported this work, and this work is partially supported by NIH Award #K99MH117264 (to AK), NSF Award #1918839 (to YY), and NSERC Award #PGSD3-532647-2019 (to JJS).

References

[1] David J Anderson and Pietro Perona. Toward a science of computational ethology. *Neuron*, 84(1):18–31, 2014.

[2] Gordon J Berman, Daniel M Choi, William Bialek, and Joshua W Shaevitz. Mapping the stereotyped behaviour of freely moving fruit flies. *Journal of The Royal Society Interface*, 11(99):20140672, 2014.

[3] Brian Broll, Matthew Hausknecht, Dave Bignell, and Adith Swaminathan. Customizing scripted bots: Sample efficient imitation learning for human-like behavior in minecraft.

[4] Xavier P Burgos-Artizzu, Piotr Dollár, Dayu Lin, David J Anderson, and Pietro Perona. Social behavior recognition in continuous video. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1322–1329. IEEE, 2012.

[5] Adam J Calhoun, Jonathan W Pillow, and Mala Murthy. Unsupervised identification of the internal states that shape natural behavior. *Nature neuroscience*, 22(12):2040–2049, 2019.

[6] Ming-Fang Chang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, et al. Argoverse: 3d tracking and forecasting with rich maps. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8748–8757, 2019.

[7] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *ICML*, 2020.

[8] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey Hinton. Big self-supervised models are strong semi-supervised learners. *arXiv preprint arXiv:2006.10029*, 2020.

[9] John D Co-Reyes, YuXuan Liu, Abhishek Gupta, Benjamin Eysenbach, Pieter Abbeel, and Sergey Levine. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. *arXiv preprint arXiv:1806.02813*, 2018.

[10] Anthony I Dell, John A Bender, Kristin Branson, Iain D Couzin, Gonzalo G de Polavieja, Lucas PJJ Noldus, Alfonso Pérez-Escudero, Pietro Perona, Andrew D Straw, Martin Wikelski, et al. Automated image-based tracking and its application in ecology. *Trends in ecology & evolution*, 29(7):417–428, 2014.

[11] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE international conference on computer vision*, pages 1422–1430, 2015.

[12] Carl Doersch and Andrew Zisserman. Multi-task self-supervised visual learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2051–2060, 2017.

[13] SE Roian Egnor and Kristin Branson. Computational analysis of behavior. *Annual review of neuroscience*, 39:217–236, 2016.

[14] Eyrun Eyjolfsdottir, Kristin Branson, Yisong Yue, and Pietro Perona. Learning recurrent representations for hierarchical behavior modeling. *ICLR*, 2017.

[15] Eyrun Eyjolfsdottir, Steve Branson, Xavier P Burgos-Artizzu, Eric D Hoopfer, Jonathan Schor, David J Anderson, and Pietro Perona. Detecting social actions of fruit flies. In *European Conference on Computer Vision*, pages 772–787. Springer, 2014.

[16] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. *ICLR*, 2018.

[17] Priya Goyal, Dhruv Mahajan, Abhinav Gupta, and Ishan Misra. Scaling and benchmarking self-supervised visual representation learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6391–6400, 2019.

[18] Jacob M Graving, Daniel Chae, Hemal Naik, Liang Li, Benjamin Koger, Blair R Costelloe, and Iain D Couzin. Deepposekit, a software toolkit for fast and robust animal pose estimation using deep learning. *Elife*, 8:e47994, 2019.

[19] Tengda Han, Weidi Xie, and Andrew Zisserman. Video representation learning by dense predictive coding. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[20] Katja Hofmann. Minecraft as ai playground and laboratory. In *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, pages 1–1, 2019.

[21] Weizhe Hong, Ann Kennedy, Xavier P Burgos-Artizzu, Moriel Zelikowsky, Santiago G Navonne, Pietro Perona, and David J Anderson. Automated measurement of mouse social behaviors using depth sensing, video tracking, and machine learning. *Proceedings of the National Academy of Sciences*, 112(38):E5351–E5360, 2015.

[22] Alexander I Hsu and Eric A Yttri. B-soid: An open source unsupervised algorithm for discovery of spontaneous behaviors. *bioRxiv*, page 770271, 2020.

[23] Mayank Kabra, Alice A Robie, Marta Rivera-Alba, Steven Branson, and Kristin Branson. Jaaba: interactive machine learning for automatic annotation of animal behavior. *Nature methods*, 10(1):64, 2013.

[24] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *arXiv preprint arXiv:2004.11362*, 2020.

[25] Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. Revisiting self-supervised visual representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1920–1929, 2019.

[26] Kevin Luxem, Falko Fuhrmann, Johannes Kürsch, Stefan Remy, and Pavol Bauer. Identifying behavioral structure from deep variational embeddings of animal motion. *bioRxiv*, 2020.

[27] Alexander Mathis, Pranav Mamidanna, Kevin M Cury, Taiga Abe, Venkatesh N Murthy, Mackenzie Weygandt Mathis, and Matthias Bethge. Deeplabcut: markerless pose estimation of user-defined body parts with deep learning. *Nature neuroscience*, 21(9):1281–1289, 2018.

[28] Simon RO Nilsson, Nastacia L Goodwin, Jia J Choong, Sophia Hwang, Hayden R Wright, Zane Norville, Xiaoyu Tong, Dayu Lin, Brandon S Bentzley, Neir Eshel, et al. Simple behavioral analysis (simba): an open source toolkit for

computer classification of complex social behaviors in experimental animals. *BioRxiv*, 2020.

[29] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[30] AJ Piergiovanni, Anelia Angelova, and Michael S Ryoo. Evolving losses for unsupervised video representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 133–142, 2020.

[31] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training, 2018.

[32] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid training data creation with weak supervision. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, volume 11, page 269. NIH Public Access, 2017.

[33] Alexander J Ratner, Christopher M De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. Data programming: Creating large training sets, quickly. In *Advances in neural information processing systems*, pages 3567–3575, 2016.

[34] Mirco Ravanelli, Jianyuan Zhong, Santiago Pascual, Pawel Swietojanski, Joao Monteiro, Jan Trmal, and Yoshua Bengio. Multi-task self-supervised learning for robust speech recognition. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6989–6993. IEEE, 2020.

[35] Aaqib Saeed, Tanir Ozcelebi, and Johan Lukkien. Multi-task self-supervised learning for human activity detection. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 3(2):1–30, 2019.

[36] Cristina Segalin, Jalani Williams, Tomomi Karigo, May Hui, Moriel Zelikowsky, Jennifer J. Sun, Pietro Perona, David J. Anderson, and Ann Kennedy. The mouse action recognition system (mars): a software pipeline for automated analysis of social behaviors in mice. *bioRxiv https://doi.org/10.1101/2020.07.26.222299*, 2020.

[37] Abhinav Shukla, Stavros Petridis, and Maja Pantic. Does visual self-supervision improve learning of speech representations? *arXiv preprint arXiv:2005.01400*, 2020.

[38] Chen Sun, Austin Myers, Carl Vondrick, Kevin Murphy, and Cordelia Schmid. Videobert: A joint model for video and language representation learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 7464–7473, 2019.

[39] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2446–2454, 2020.

[40] Ziyu Wang, Josh S Merel, Scott E Reed, Nando de Freitas, Gregory Wayne, and Nicolas Heess. Robust imitation of diverse behaviors. In *Advances in Neural Information Processing Systems*, pages 5320–5329, 2017.

[41] Alexander B Wiltschko, Matthew J Johnson, Giuliano Iurilli, Ralph E Peterson, Jesse M Katon, Stan L Pashkovski, Victoria E Abraira, Ryan P Adams, and Sandeep Robert Datta. Mapping sub-second structure in mouse behavior. *Neuron*, 88(6):1121–1135, 2015.

[42] Raymond A Yeh, Alexander G Schwing, Jonathan Huang, and Kevin Murphy. Diverse generation for multi-agent sports games. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4610–4619, 2019.

[43] Eric Zhan, Albert Tseng, Yisong Yue, Adith Swaminathan, and Matthew Hausknecht. Learning calibratable policies using programmatic style-consistency. *ICML*, 2020.

[44] Eric Zhan, Stephan Zheng, Yisong Yue, Long Sha, and Patrick Lucey. Generating multi-agent trajectories using programmatic weak supervision. *ICLR*, 2019.

[45] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networkss. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.