Automatic Synthesis of Diverse Weak Supervision Sources for Behavior Analysis Supplementary Materials

Albert Tseng
Nuro*

Jennifer J. Sun
Caltech

Yisong Yue
Caltech, Argo AI*

Here, we provide additional information about our datasets and experiments. The Supplementary Materials are organized into three sections: implementation details (Section 1), example learned programs (Section 2), and additional results (Section 3).

1. Implementation Details

1.1. Datasets and Domain-level Labeling Functions

Each of our behavior domains contains a dataset with tracked keypoints and features, domain-level labeling functions (LFs), and a DSL. Below, we provide more details about the datasets and domain-level LFs. Details about the DSL are provided in Section 1.2.

Fly vs. Fly. The Fly dataset [2] consists of videos (144×144 at 30 Hz), extracted behavioral features based on fly trajectories, and frame-level behavior annotations. We use the "Aggression" and "Courtship" videos from this dataset, similar to [5], and use the frame-level behavior annotations to evaluate our framework for behavior classification. The breakdown of behaviors in this dataset is given by: lunge - 1.24%, wing threat - 4.26%, tussle - 0.35%, wing extension - 3.31%, circle - 0.80%, copulation - 59.73%, and no behavior - 30.42%. This dataset is available under the CC0 1.0 Universal license.

The domain-level LFs for the Fly DSL are based on the behavioral features provided with the dataset, and are crafted from the outputs of FlyTracker [2]. They are similar to the fly behavior features used in [5], and we provide visualizations for example LFs in Figure 1. The domain-level LFs included in the Fly DSL, grouped by category, are:

- Angular LFs: angular velocity of each fly, angle between fly trajectories, facing angle of flies.
- Linear (speed) LFs: linear velocity and speed of each fly.
- Positional LFs: position of each fly, distance between flies, distance between the legs of each fly.
- Ratio LFs: ratio of major and minor axis of fly shape for each fly, ratio of major and minor axis of fly body, ratio of major and minor axis of bounding box around wing spread.
- Wing LFs: minimum and maximum wing angle (window), mean wing length (window), wingspan.

where "window" indicates that the LF is computed over a sliding window of the past 20 frames.

CalMS21 (Mouse). CalMS21 [4] consists of trajectory data, frame-level behavior annotations, and video data (1024×570 at 30 Hz) of a pair of interacting mice in a resident-intruder assay. The trajectory data from CalMS21 is from the MARS detector [3], which detects 7 anatomically-defined body parts on each mouse in the form of keypoints. We use the trajectory data and behavior annotations from the Task 1 train/test split to evaluate our framework for mouse behavior classification. The breakdown of behaviors in this dataset is given by: attack - 3.44%, investigation - 27.56%, mount - 7.45%, and other - 61.56% This dataset is available under the CC-BY-NC-SA license.

The domain-level LFs for the Mouse DSL are based on existing behavioral features used by domain experts in [3] and are similar to the features used in [5] (visualized in Figure 2). The LFs included in the Mouse DSL, grouped by category, are:

*Work done while author was affiliated with Caltech.

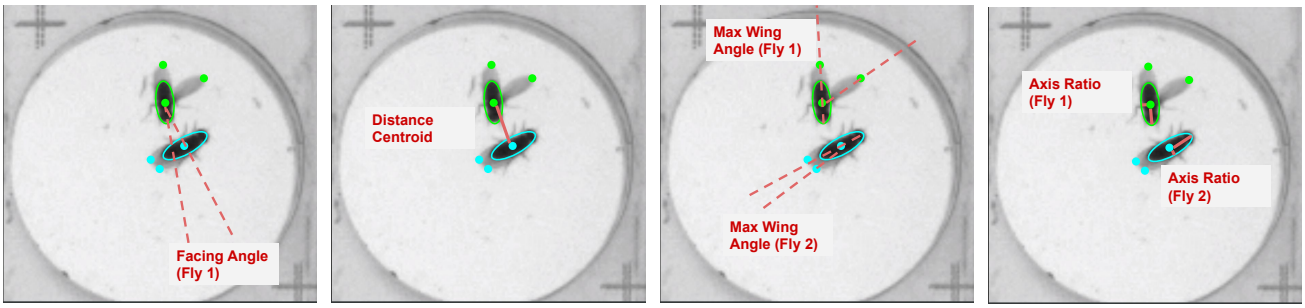

Figure 1. Visualizing a subset of programs computed on Fly vs. Fly. Reproduced with permission from [5].

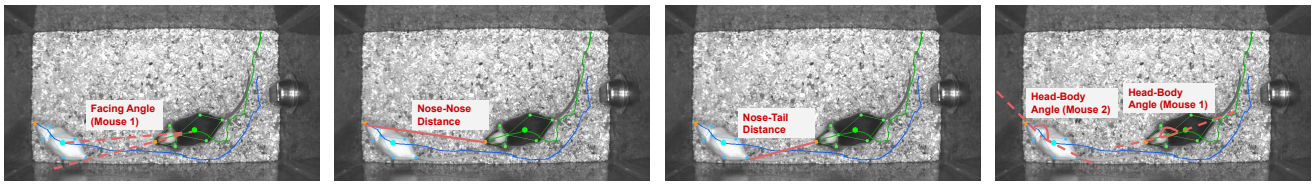

Figure 2. Visualizing a subset of programs computed on CalMS21. Reproduced with permission from [5].

- Positional LFs: nose, right ear, left ear, neck, RHS, LHS, and tail position for each mouse.
- Centroid LFs: centroid position of entire mouse, head, hips, and body for each mouse.
- Angular LFs: orientation of entire mouse, head, body, and angle between head and body (L&R) for each mouse, as well as angle and facing angle between the two mice.
- Shape LFs: ratio of major and minor axis of mouse body and bounding ellipse for each mouse, as well as ratio of bounding ellipse areas between the two mice, bounding ellipse overlap, and edge distance between bounding ellipses.
- Speed LFs: linear, radial, and tangential speed of each mouse, linear acceleration of each mouse, and relative speed of resident vs intruder mouse.
- Relative Distance LFs: relative distance between bodies, noses, heads, and centroids of the two mice, among other relative distances.

Basketball. The StatsPerform Generative Models Basketball [6] dataset consists of basketball player trajectories from NBA games. Each trajectory is located in the left half of the court, and contains 25 frames sampled at 3Hz of 5 defense players, 5 offense players, and 1 basketball. As the Basketball dataset does not have ground truth labels for any tasks, we use a "ballhandler" label computed from the position of the ball relative to each player [1]. As such, we exclude the position of the ball itself from our training data and domain-level LFs. The distribution of ballhandler by player is given by: 1 - 18.50%, 2 - 22.09%, 3 - 22.87 %, 4 - 18.18%, 5 - 18.36%. Since we are interested in the ballhandler, we exclude the offense players from our training data and domain-level LFs.

The dataset itself is available for free on AWS Marketplace. The list of domain-level LFs included in the Basketball DSL, grouped by category, are:

- Player: acceleration, velocity, speed, and position of each defense player.
- Ball: acceleration, velocity, and speed of the ball. Note this does not include the starting position of the ball so the true position of the ball cannot be computed from the velocity.

1.2. Domain Specific Language (DSL)

The DSL we use in our experiments consists of elementary operations, operations on sequential data, and branching structures, among others. We use the same DSL for all domains; only the domain-level LFs differ from domain to domain. In Backus-Naur form, our DSL can be written as:

$$\begin{aligned}\alpha \quad ::= \quad & x \mid c \mid +(\alpha_1, \ldots, \alpha_k) \mid \cdot(\alpha_1, \ldots, \alpha_k) \mid \times(\alpha_1, \ldots, \alpha_k) \mid \frown(\alpha_1, \ldots, \alpha_k) \\ & \oplus_\theta(\alpha_1, \ldots, \alpha_k) \mid \oplus_D x \mid \textbf{if } \alpha_1 \textbf{ then } \alpha_2 \textbf{ else } \alpha_3 \\ & \textbf{map } (\textbf{fun } x_1, \alpha_1) \; x \mid \textbf{fold } (\textbf{fun } x_1, \alpha_1) \; c \; x\end{aligned} \tag{1}$$

Here, x and c represent input features and constants, respectively. $+, \cdot, \times$, and $\frown$ represent the addition, dot product, outer product, and concatenation operators. $\oplus_\theta$ represents domain-expert provided parameterized library functions and $\oplus_D$ represents domain-level labeling functions. $\textbf{fun } x.f$ represents a lambda that evaluates f over x; for example, if x is a sequence $\{x_0, x_1, \ldots, x_k\}$ then $\textbf{fun } x.f$ returns $\{f(x_0), f(x_1), \ldots f(x_k)\}$. $\textbf{map}$ and $\textbf{fold}$ represent operations on vectors and sequences, respectively.

$\textbf{if} - \textbf{then} - \textbf{else}$ represents a differentiable ITE construct, which can be written with our program notation $[\![\alpha]\!](x, \theta)$ as

$$y = \sigma(\beta [\![\alpha]\!](x, \theta_1)) \tag{2}$$

$$[\![\textbf{if } \alpha_1 \textbf{ then } \alpha_2 \textbf{ else } \alpha_3]\!](x, (\theta_1, \theta_2, \theta_3)) = y[\![\alpha_2]\!](x, \theta_2) + (1 - y)[\![\alpha_3]\!](x, \theta_3) \tag{3}$$

where σ is the sigmoid function and β serves as a temperature parameter. As $\beta \to \infty$, Equation 3 becomes a regular ITE.

1.3. Training Details

Student Networks. We use neural networks (NNs) for nonsequential data and LSTMs for sequential data. For NN based student networks, we use 3 hidden layers with layer sizes of $l_1 \sim \mathcal{U}\{50, 90\}, l_2 \sim \mathcal{U}\{50, 80\}, l_3 \sim \mathcal{U}\{20, 30\}$. Each hidden layer is followed by a dropout layer with dropout probability $d_1 \sim \mathcal{U}_{[0,0.2]}, d_2 \sim \mathcal{U}_{[0,0.2]}, d_3 \sim \mathcal{U}_{[0,0.1]}$. All hidden layers use the ReLU activation function. For LSTM based student networks, we use a 1 or 2 layer LSTM with equal probability and $h \sim \mathcal{U}\{64, 128\}$ hidden units. If we use a 2 layer LSTM, we also set the LSTM dropout probability to $d \sim \mathcal{U}_{[0,0.2]}$. The final hidden state of the LSTM is classified with a neural network with two hidden layers of sizes $l_1 \sim \mathcal{U}\{50, 80\}, l_2 \sim \mathcal{U}\{20, 30\}$. Both layers are each followed by a dropout layer $\sim_{i.i.d} \mathcal{U}_{[0,0.2]}$ and use ReLU activation. All student networks are trained with the Adam optimizer, using a learning rate of 10^{-4} for NNs and 3×10^{-4} for LSTMs.

AutoSWAP Programs. For the Fly and Mouse dataset, neural completions were trained for 6 epochs, and symbolic components were trained for 15 epochs. For the Basketball dataset, neural completions were trained for 4 epochs, and symbolic components were trained for 6 epochs. For all datasets, parameters were optimized with the Adam optimizer. A learning rate of 10^{-3} was used for the Fly and Mouse datasets, and 0.02 was used for the Basketball dataset. β was set to 1 for the differentiable ITE construct.

Downstream Classifier. Again, we use NNs for nonsequential data and LSTMs for sequential data. The downstream classifier NN is a 3 layer network with $(128, 64, 32)$ units in each hidden layer, respectively. Each layer is followed by a dropout layer with dropout probability $(0.5, 0.4, 0.3)$, respectively, and uses ReLU activation. For active learning experiments, the weak label outputs are also included via skip connections to each layer. For LSTM based downstream classifiers, we use a two layer LSTM with 128 hidden units and a dropout probability of 0.2. The final hidden state of the LSTM is classified by a two layer neural network, with the first layer having 128 units with dropout probability 0.3, and the second layer having 48 units with dropout probability 0.2. Again, both layers use ReLU and weak label outputs are included via skip connections for active learning experiments. All downstream classifiers are trained with the Adam optimizer, using a learning rate of 10^{-4} for NNs and 3×10^{-4} for LSTMs.

1.4. Reproducibility

The code for our experiments is available at https://github.com/autoswap/autoswap_cvpr_2022. All experiments were run on two different virtual machines with 12 Broadwell cores clocked at 2.6GHz and two Nvidia Tesla P40 GPUs each. Some experiment sets were run across different machines due to resource constraints, thus some results that should otherwise be identical have slight differences.

2. Learned Labeling Functions

Here, we present some example AutoSWAP LFs and our interpretations of them. While domain experts may have different interpretations from us, the purpose of this section is to demonstrate the relative interpretability of AutoSWAP LFs.

Example 1: AutoSWAP labeling function for the Ballhandler task. The program can be interpreted as using the sum of the frame level probabilities ($\textbf{fold}$), with each frame level probability being determined from player velocities or player coordinates depending on the velocity of the ball. Summing over frame-level probabilities corresponds well with the "majority" part of the ballhandler task (find the player that was the ballhandler for the majority of the sequence). The ITE construct can

be thought of as detecting passes, as the ball moves the fastest when it is being passed between players. Each **Affine* labeling function contains a set of parameters describing a linear transformation on the base LF. We include the parameters for scalar affine LFs as $x_{[a,b]}$, which indicates the transformation $ax + b$ is applied. We do not include the parameters for vector LFs, as they are too large to list here, but examples can be found by running the code.

$$
\begin{aligned}
&\textbf{fold}(\textbf{fun}\ x_t\ .\ [\\
&\quad \textbf{if}\ \textit{BallVelocityAffine}(x_t)_{[0.73,-0.26]}\\
&\quad \textbf{then}\ \textit{PlayerVelocitiesAffine}(x_t)\\
&\quad \textbf{else}\ \textit{PlayerCoordinatesAffine}(x_t)\\
&])\ x_t
\end{aligned}
$$

Example 2: AutoSWAP labeling function for the Wing Threat task (fly domain). The program can be interpreted as using the product of the resident fly's wing ratio and the sum of the resident fly's body ratio and the distance between the two flies to derive a probability for the resident fly displaying a wing threat towards the intruder fly. The program learns a positive a for the wing ratio domain-level LF, which is reasonable as wing threats are correlated with wing spreading.

$$
\begin{aligned}
&\textbf{map}(\textbf{fun}\ x\ .\ [\\
&\quad \times\ (\textit{WingRatioAffine}(x)_{[0.96,-0.71]},\\
&\qquad +\ (\textit{BodyRatioAffine}(x)_{[0.35,-0.20]}, \textit{FlyDistanceAffine}(x)_{[0.67,0.13]}))\\
&])\ x
\end{aligned}
$$

Example 3: AutoSWAP labeling function for the mouse behavior classification task. Since the mouse task is a multi-class classification task (for the "attack", "investigate", and "mount" behaviors), this program is not as trivially interpretable as those for the Fly and Basketball domains. Nevertheless, we can still deduce that the program classifies the angular domain-level LFs if the distance is small and the speed domain-level LFs otherwise. This is reasonable, as if the mice are far apart, the speed of the mice is probably sufficient for determining which behavior they are engaging in, and if they are close to each other, more detailed data (such as the angular LFs) may be needed.

$$
\begin{aligned}
&\textbf{map}(\textbf{fun}\ x_t\ .\ [\\
&\quad \textbf{if}\ \textit{MouseDistanceAffine}(x)_{[1.02,-0.38]}\\
&\quad \textbf{then}\ \textit{MouseAngularAffine}(x)\\
&\quad \textbf{else}\ \textit{MouseSpeedAffine}(x)\\
&])\ x
\end{aligned}
$$

3. Additional Results

Using More Labeling Functions. In our main experiments, we used 3 labeling functions. The plots below (Figures 3, 4, 5) show the effect of using more labeling functions (5, 7). In general, active learning and random sampling performance stays relatively stable, but some improvement can be observed in weak supervision settings. This can most likely be attributed to the generative weak label model performing better with more labeling functions, as well as improved coverage from the diversity measures.

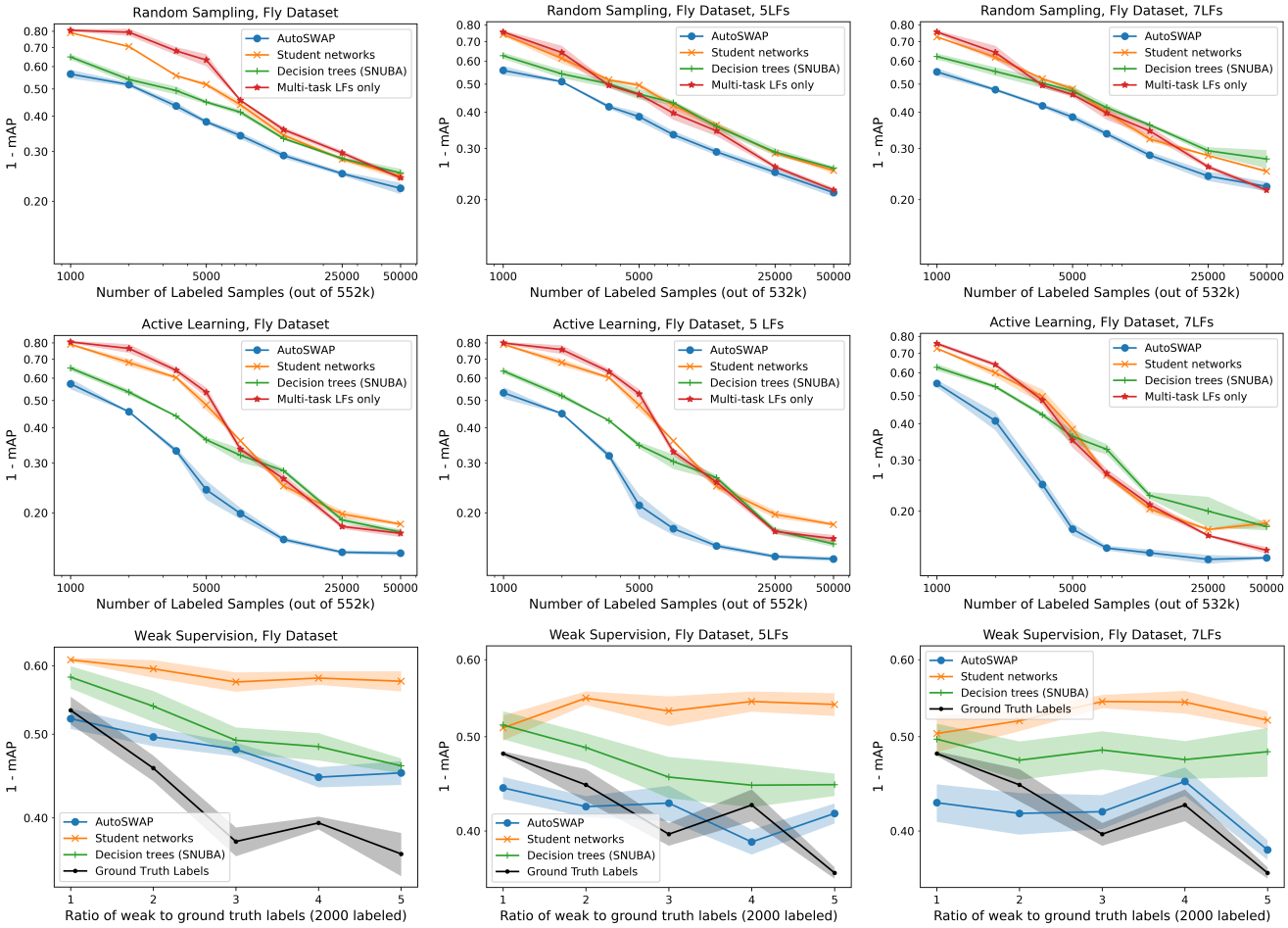

Figure 3. Experiments with 3 (left-most column), 5, and 7 LFs for the Fly dataset. All plots are on a log-log scale. Due to resource constraints, some experiments were run on different machines, effectively giving different seeds. Results within each plot were run on the same machine and are directly comparable, but some results that should otherwise be the same across plots (e.g. ground truth labels) may have slight differences. We note that in all plots, regardless of seed, AutoSWAP outperforms the baselines.

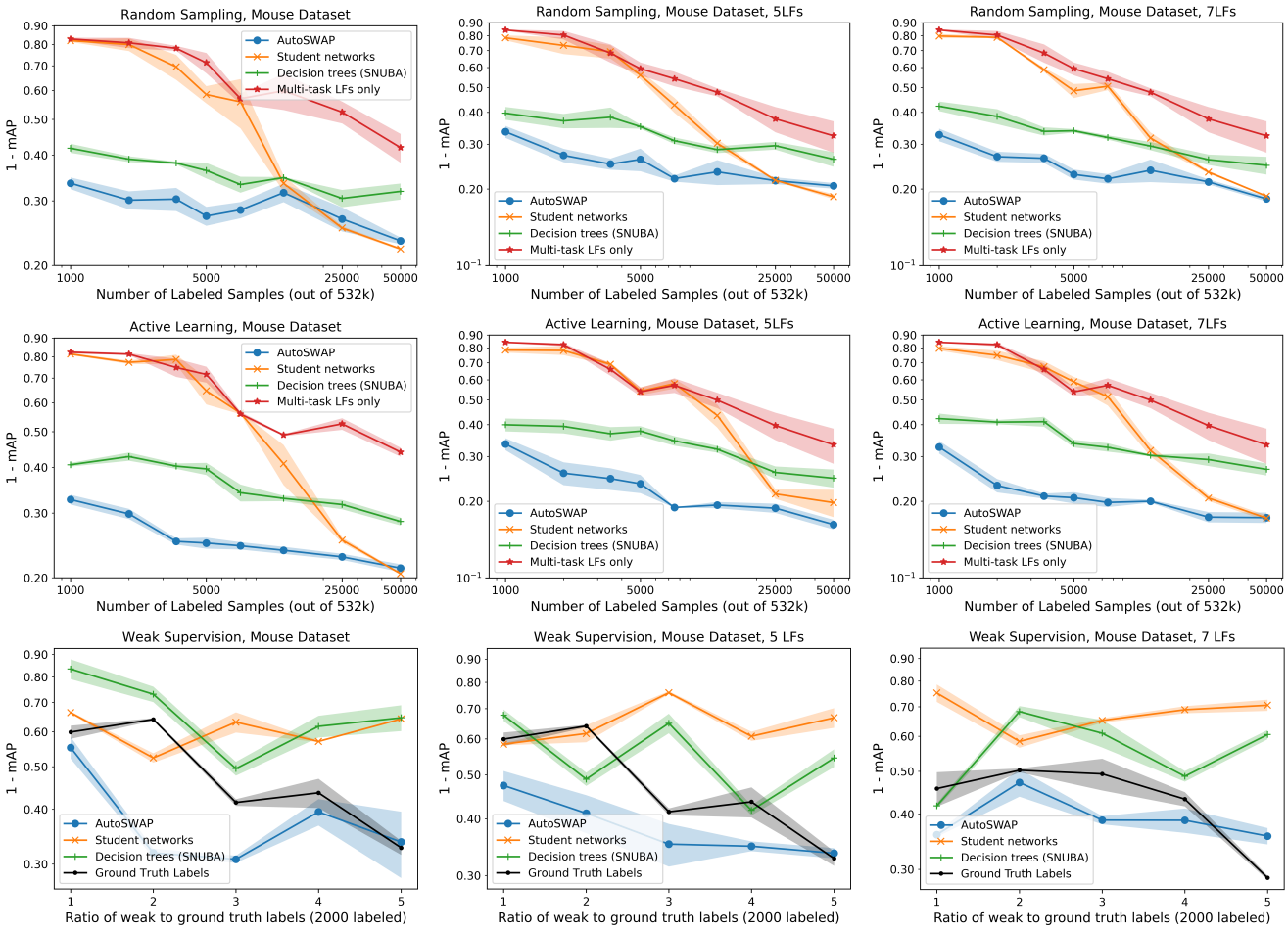

Figure 4. Experiments with 3 (left-most column), 5, and 7 LFs for the Mouse dataset. All plots are on a log-log scale. Due to resource constraints, some experiments were run on different machines, effectively giving different seeds. Results within each plot were run on the same machine and are directly comparable, but some results that should otherwise be the same across plots (e.g. ground truth labels) may have slight differences. We note that in all plots, regardless of seed, AutoSWAP outperforms the baselines.

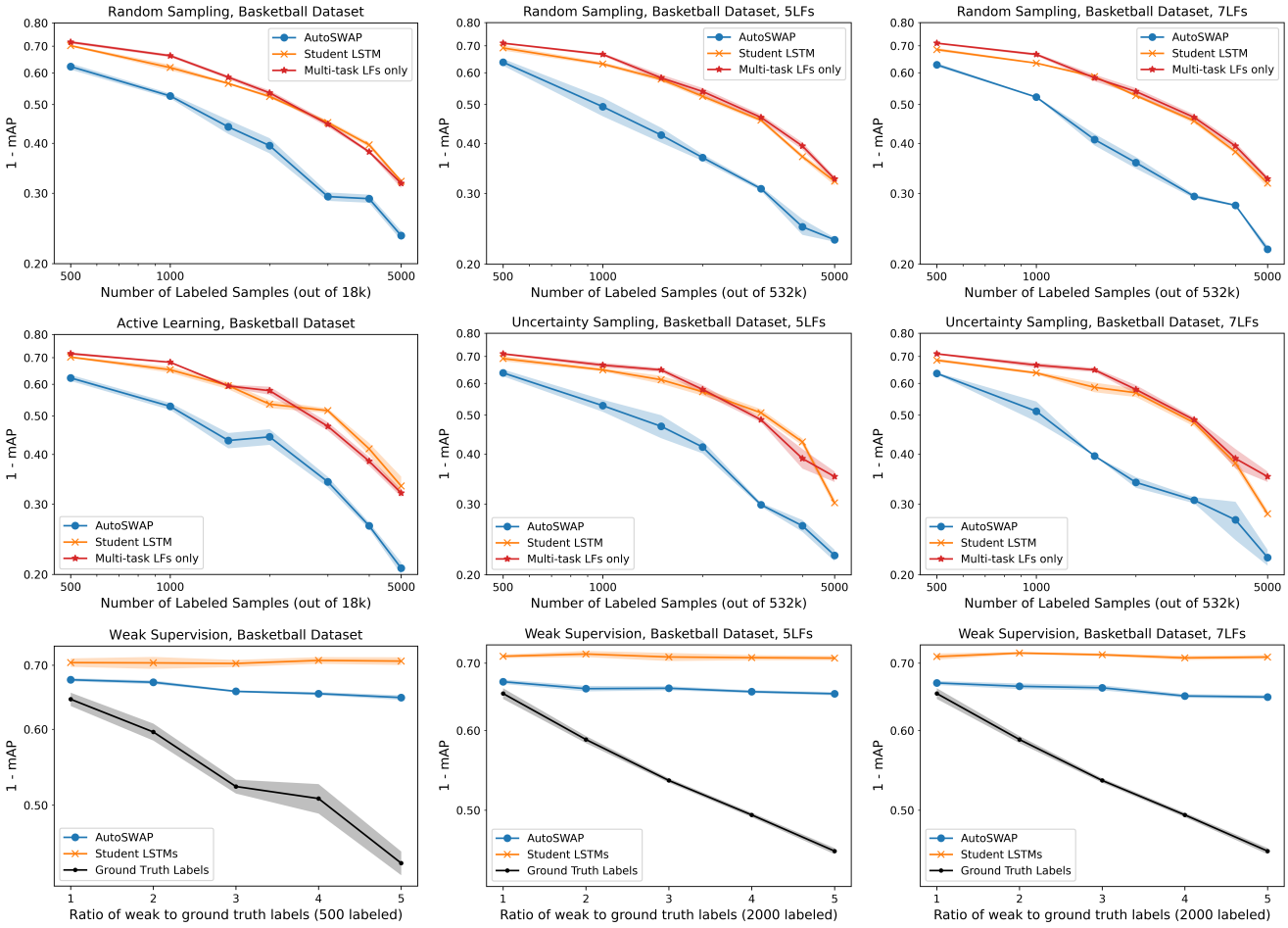

Figure 5. Experiments with 3 (left-most column), 5, and 7 LFs for the Basketball dataset. All plots are on a log-log scale. Due to resource constraints, some experiments were run on different machines, effectively giving different seeds. Results within each plot were run on the same machine and are directly comparable, but some results that should otherwise be the same across plots (e.g. ground truth labels) may have slight differences. We note that in all plots, regardless of seed, AutoSWAP outperforms the baselines.

Isolated Labeling Function Performance. Here, we present the performance of the generated task-level labeling functions themselves, outside of any downstream tasks. Since task-level LFs give labels for the task at hand, they can be evaluated in the same context as the downstream tasks. As can be seen in Figure 6, AutoSWAP LFs do not always outperform LFs from student networks and decision trees. This indicates that the data efficiency benefits of AutoSWAP come from the higher quality learning signals AutoSWAP LFs give in downstream tasks, rather than their raw performance in a vacuum. Furthermore, AutoSWAP LFs generated without the diversity cost do not perform significantly differently than those generated with the diversity cost, indicating that the diversity cost further improves the quality of the LFs' learning signal.

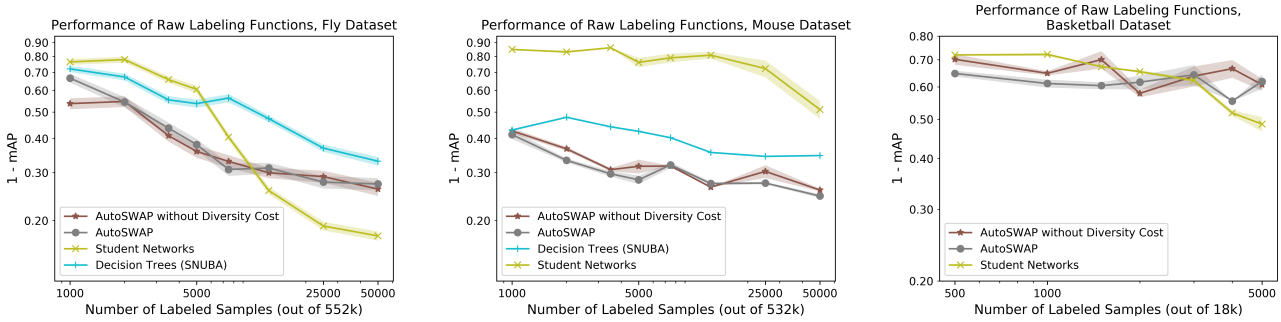

Figure 6. Performance of generated task-level labeling functions outside of downstream tasks. As can be seen, AutoSWAP does not always outperform the baseline LF generation methods. This indicates that the data efficiency benefits of AutoSWAP come from the improved learning signal of AutoSWAP LFs relative to baseline LFs in downstream tasks. Furthermore, AutoSWAP LFs generated without the diversity cost do not perform significantly differently than those generated with the diversity cost, indicating that the diversity cost further improves the quality of the LFs' learning signal.

Effect of Diversity Cost Table 1 shows a numerical comparison of the mean pairwise edit distance of 5 AutoSWAP LFs, taken over 5 seeds. Adding the diversity cost results in increased pairwise edit distance over all three behavior domains and increased performance in our experiments.

Frames	Fly	No Diversity	Mouse	No Diversity	Basketball	No Diversity
2000	**5.06** (0.08)	4.08 (0.24)	**4.62** (0.11)	3.63 (0.29)	**3.36** (0.08)	3.06 (0.13)
5000	**5.00** (0.08)	3.86 (0.14)	**4.40** (0.22)	4.17 (0.13)	**3.32** (0.09)	3.08 (0.12)
12500	**4.76** (0.04)	3.50 (0.21)	**4.58** (0.19)	3.94 (0.14)	**3.24** (0.05)	3.06 (0.06)
50000	**4.90** (0.06)	3.34 (0.39)	**4.88** (0.12)	3.77 (0.35)	**3.16** (0.04)	2.86 (0.07)

Table 1. Mean pairwise edit distance of 5 AutoSWAP LFs with & without diversity cost (5 seeds). Standard error in parentheses. Using the diversity cost results in programs that are "farther apart" and thus more structurally diverse.

Compared to Self-Supervised Methods Self-supervised methods such as TREBA [5] have shown promise in reducing the number of labeled data points needed to achieve performance parity in behavior analysis domains. Furthermore, such methods are generally complimentary to LF generation methods, as learned representations can be used as additional input features. We provide an empirical analysis of combining AutoSWAP and TREBA in the Mouse domain active learning task in Figure 7, as TREBA features for CalMS21 are readily available. We do not provide an analysis of TREBA for weak supervision task since TREBA features are *features* and not *labels*, and thus any comparisons between TREBA and AutoSWAP there would be indirect. As can be seen in Figure 7, AutoSWAP alone outperforms TREBA, especially at lower levels of annotated samples, and using both gives slight performance benefits over using AutoSWAP alone.

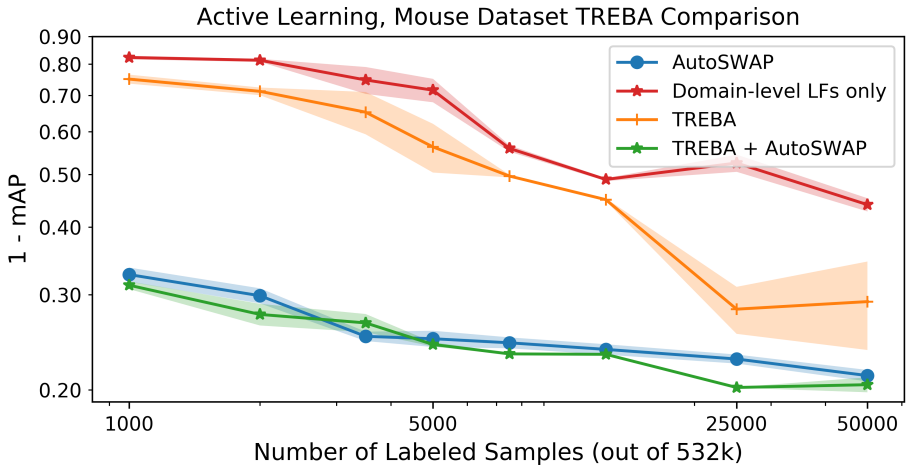

Figure 7. Using self-supervised TREBA [5] features improves performance compared to only using domain-level LFs.

References

[1] Jenna Wiens Armand McQueen and John Guttag. Automatically recognizing on-ball screens. In *MIT Sloan Sports Analytics Conference*, 2014. 2

[2] 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. 1

[3] 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. 1

[4] Jennifer J. Sun, Tomomi Karigo, Dipam Chakraborty, Sharada Mohanty, Benjamin Wild, Quan Sun, Chen Chen, David Anderson, Pietro Perona, Yisong Yue, and Ann Kennedy. The multi-agent behavior dataset: Mouse dyadic social interactions. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. 1

[5] Jennifer J Sun, Ann Kennedy, Eric Zhan, David J Anderson, Yisong Yue, and Pietro Perona. Task programming: Learning data efficient behavior representations. In *Conference on Computer Vision and Pattern Recognition*, 2021. 1, 2, 8, 9

[6] Yisong Yue, Patrick Lucey, Peter Carr, Alina Bialkowski, and Iain Matthews. Learning fine-grained spatial models for dynamic sports play prediction. In *2014 IEEE international conference on data mining*, pages 670–679. IEEE, 2014. 2