KAGGLE MABe – SOCIAL ACTION RECOGNITION IN MICE PRESENTATION

https://www.youtube.com/watch?v=tTyxZPQVIkM

EN.605.742.81: Deep Neural Networks
Professor: Oleg Melnikov
Team Members: XiangFeng Liang, Lijian Gong, Swarup Sahu

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

kaggle

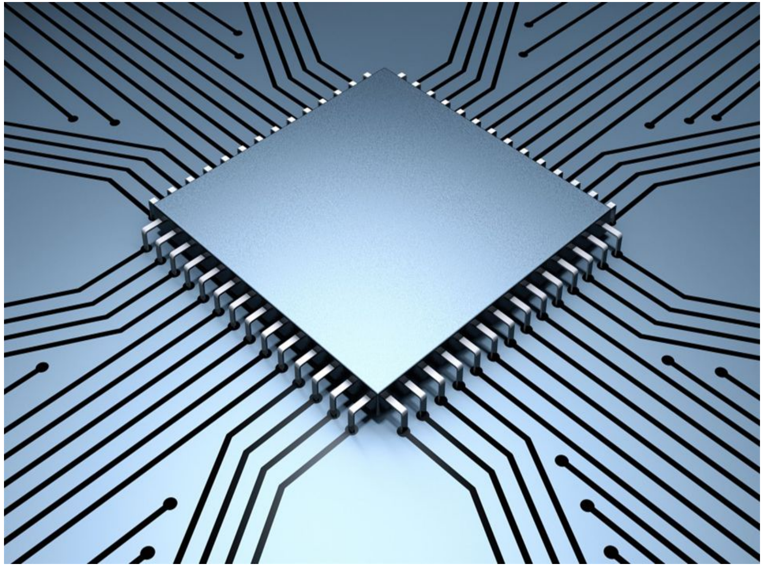
Background

JHU-Team 1 Participation Overview

- Team members: Swarup Sahu, Xiang Feng Liang, Lijian Gong
- Competition started on September and concludes on December 15, 2025
- Prize pool totals $50,000 with 1280 teams on leaderboard
- Consent granted to share presentations for educational use after December 2, 2025
- Project completed as part of Johns Hopkins University DNN 605.742 course

(AI generated image)

Key Acronyms in Our Project



(AI-generated image)

CPU/GPU/TPU stand for central, graphical, and tensor processing units.

FP, FPIP, FPVP: course acronyms for to final, interim, and video project presentations.

RAM: short for random access memory in computing.

TF: Tensorflow, open-sourse machine learning library developed by Google.

URL: represents uniform resource locator, a web address.

MS PPT: Microsoft Powerpoint*

LB: leaderboard tracking within a competition's standings.

F1-Score: competition metric. Calculated by using recall & precision metrics

* Note that some images and slides were created in Microsoft Powerpoint and then ported to Google Slide. Copilot was used to develop complicated visuals; the authors have done their best to label images developed by LLMs where applicable..

Advancing Mouse Behavior Analysis with Machine Learning

- This competition is sponsered by Cornell University
- Develop a model to recognize over 30 mouse behaviors from video footage data
- Leverage a comprehensive dataset with 400+ hours of mice actions & tracked movements using 20+ different recording systems
- Overcome challenges of limited labeled data and rare mouse behaviors (ex: ‘rest’ or ‘freeze’)
- Aim for robust cross-lab generalization to ensure wide applicability
- Aim of competition is to automate large-scale behavior analysis to support neuroscience and ecology

(AI-generated image)

Training Infrastructure & Dataset Overview:

- Kaggle notebooks must complete runs within 9 hours on CPU or GPU
- Kaggle notebooks run in a fully offline environment — no internet access is allowed during execution.
- Typical RAM available: ~13 GB for CPU, ~16 GB for GPU notebooks
- Dataset includes 400+ hours of video from 20+ different recording methods
- Metadata stored as csv files & tracking & annotation data as .parquet files. The final submission data requires 7 fixed columns.
- Competition started in September. Key deadlines:
  - 12/7: Deadline to merge teams
  - 12/15: Final submission

(AI-generated image)

Team's Metric & Performance Overview

- Performance tracked across multiple modules from 5 to 14.
- Latest leaderboard scores range from 0.268 to 0.446.
- Team rank improved steadily, reaching 441 out of 1286 teams.
- Cumulative submissions increased from 5 to 59 over modules.

Module	9	10	11	12	13	14
LB score	0.281	0.362	0.370	0.268	0.378	0.446
rank/#teams	523/857	508/930	492/1005	474/123	456/1207	441/1286
Cumulative # of submissions	5	16	24	33	47	59

- Macro F scores are averaged across each lab & video
- Scores only specific behaviors and the mice that were annotated for a specific video
- Includes a weighting factor to balance recall / precision
- Macro-averaging across actions prevents long, frequent behaviors from overwhelming the metric (ex: 'sniff' vs 'rest'))

$$F_\beta(a) = \frac{(1+\beta^2)\cdot TP_a}{(1+\beta^2)\cdot TP_a + \beta^2 \cdot FN_a + FP_a}$$

$$F_{\beta\,\text{macro}} = \frac{1}{n}\sum_{a=1}^{n} F_\beta(a)$$

β = weighting factor
a = single action (ex: sniff)

```
    for action in distinct_actions:
        if tps[action] + fns[action] + fps[action] == 0:
            action_f1s.append(0)
        else:
            action_f1s.append((1 + beta**2) * tps[action] / ((1
+ beta**2) * tps[action] + beta**2 * fns[action] + fps[action]))
    return sum(action_f1s) / len(action_f1s)
```

Source: MABe F Beta Notebook

LB URL: MABe Challenge - Social Action Recognition in Mice | Kaggle Leaderboard

Team Name: JHU-Team-1

Data’s Key Features

Data’s Key Features:

- **Metadata**: provides general information about each video and the mice within it.
  - Teams received CSV metadata for all videos used for *training* | *test*. Categories of features are listed below:
    - **Identifiers** → the lab providing the data and the video id
    - **Video data** → frames per second, video duration, pixels per cm, etc.
    - **Arena** → the shape, type, dimensions, etc.
    - **Labels** → i.e tracked body parts (tail, ears, etc.), behaviors labeled (chase, attack, etc.), tracking method

- **Tracking**: "pose data" or where the mouse's body parts are within each frame.
  - All data for both *training* | *test* was in .parquet files *(more on this later)*. Below are all of the features:
    - **video_frame** → Frame index
    - **mouse_id** → Mouse ID
    - **bodypart** → Body part tracked
    - **x, y** → X/Y pixel coordinates

- **Annotation**: behavior labels (i.e. what one mouse may be doing at a given time to itself or another mouse).
  - Same as the tracking data above, all annotation data was in .parquet format. Only training data was available:
    - **agent_id** → the mouse performing the behavior
    - **target_id** → the mouse receiving the behavior (or same mouse for self-directed actions like grooming)
    - **action** → the behavior type (e.g., grooming, chasing, sniffing)
    - **start_frame / stop_frame** → when the behavior occurs in the video

Data’s Key Features:

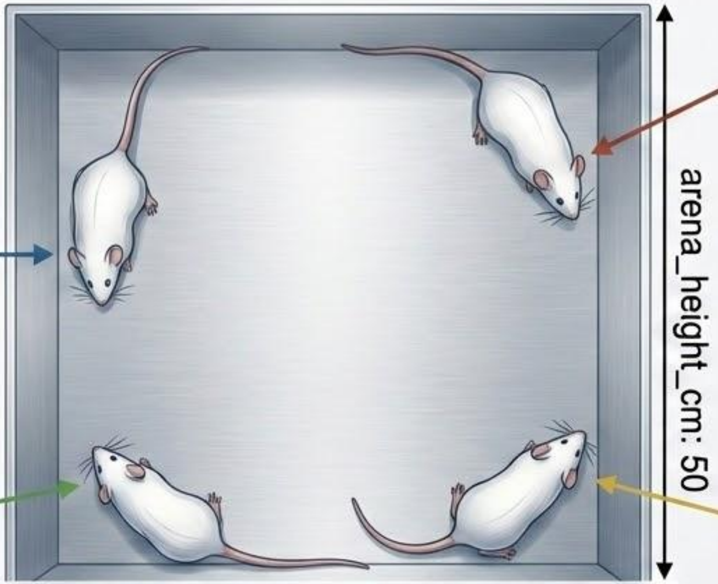
Metadata Structure

video_id: 44566106
video_id: 44566106
frames_per_second: 30
video_duration_sec: 60
arena_shape: square

Mouse 1: **Strain** CD-1 (ICR)
Color: white
Sex: male
ID: 10.0, 8-12 weeks
Condition: wireless device

Mouse 3: **Strain** CD-1 (ICR)
Color: white
Sex: male
Age: 8-12 weeks
Condition: wireless device

Diagram and Raw Data



body_parts_tracked: nose, ears, body_center, tail, behaviors
_labeled: locomotion, pause, sniff_body, **tracking_method**

Raw Data

Mouse 2:
Strain: CD-1 (ICR)
Color: white
Sex: male
ID: 38.0
Age: 8-12 weeks
Condition: wireless device

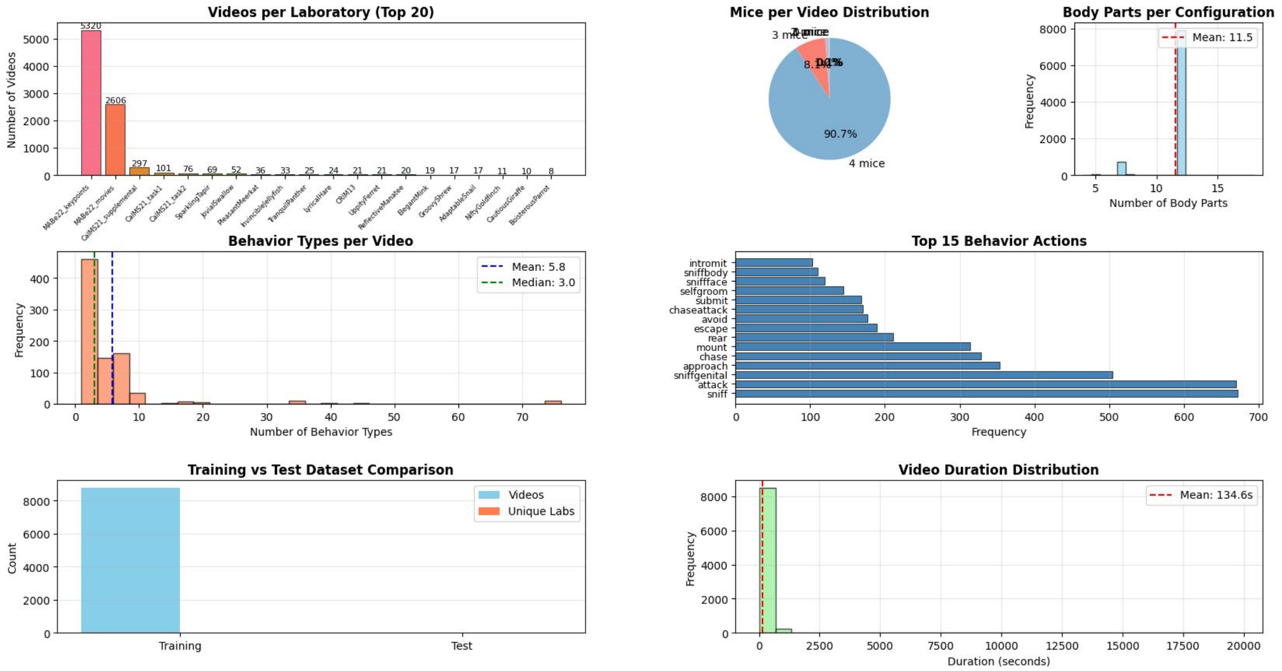
Mouse 4:
Strain: CD-1 (ICR)
Color: white
Sex: male
ID: 51.0
Age: 8-12 weeks
Condition: wireless device

Raw Data

video_id	lab_id	arena_type	mouse1_strain	mouse1_age	mouse2_strain	mouse2_age	mouse3_strain	mouse4_strain	mouse4_age
44566106	40602	open_field	CD-1 (ICR)	8-12 weeks	male | male	8-12 weeks	CD-1 (ICR)	male | male	8-12 weeks

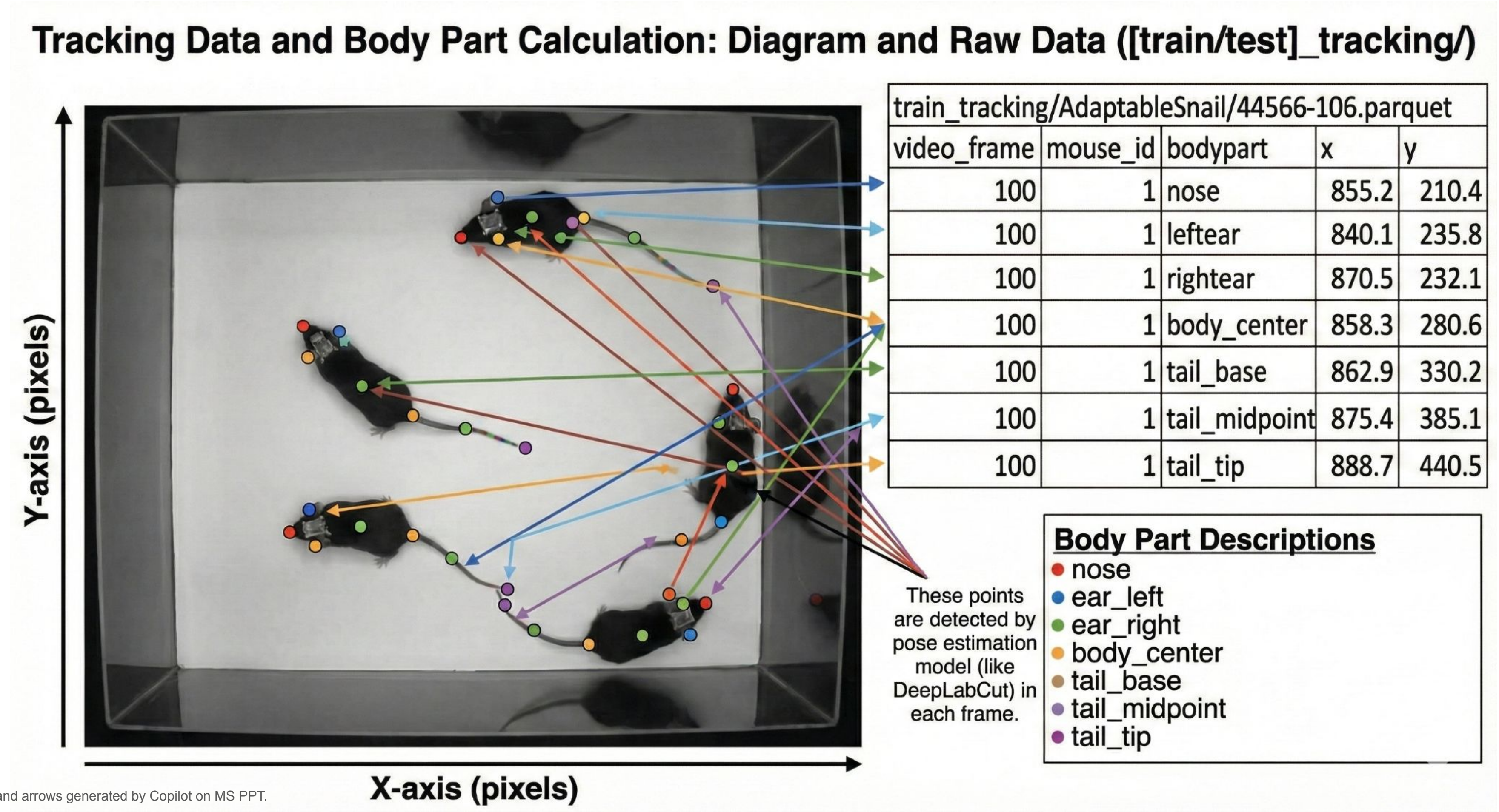
Center image and background generated by Copilot on MS PPT.

MABe Dataset Comprehensive Overview




```
DATASET STATISTICS SUMMARY
==================================================================

TRAINING DATA:
- Total Videos: 8,789
- Videos with Missing Behaviors: 7941 (90.4%)
- Unique Laboratories: 21
- Average Mice per Video: 3.89
- Total Unique Body Part Configurations: 10

TEST DATA:
- Total Videos: 1
- Unique Laboratories: 1

BEHAVIOR ANALYSIS:
- Total Unique Behaviors: 37
- Most Common Behavior: sniff (672 occurrences)
- Average Behaviors per Video: 5.8
- Max Behaviors in Single Video: 76
```

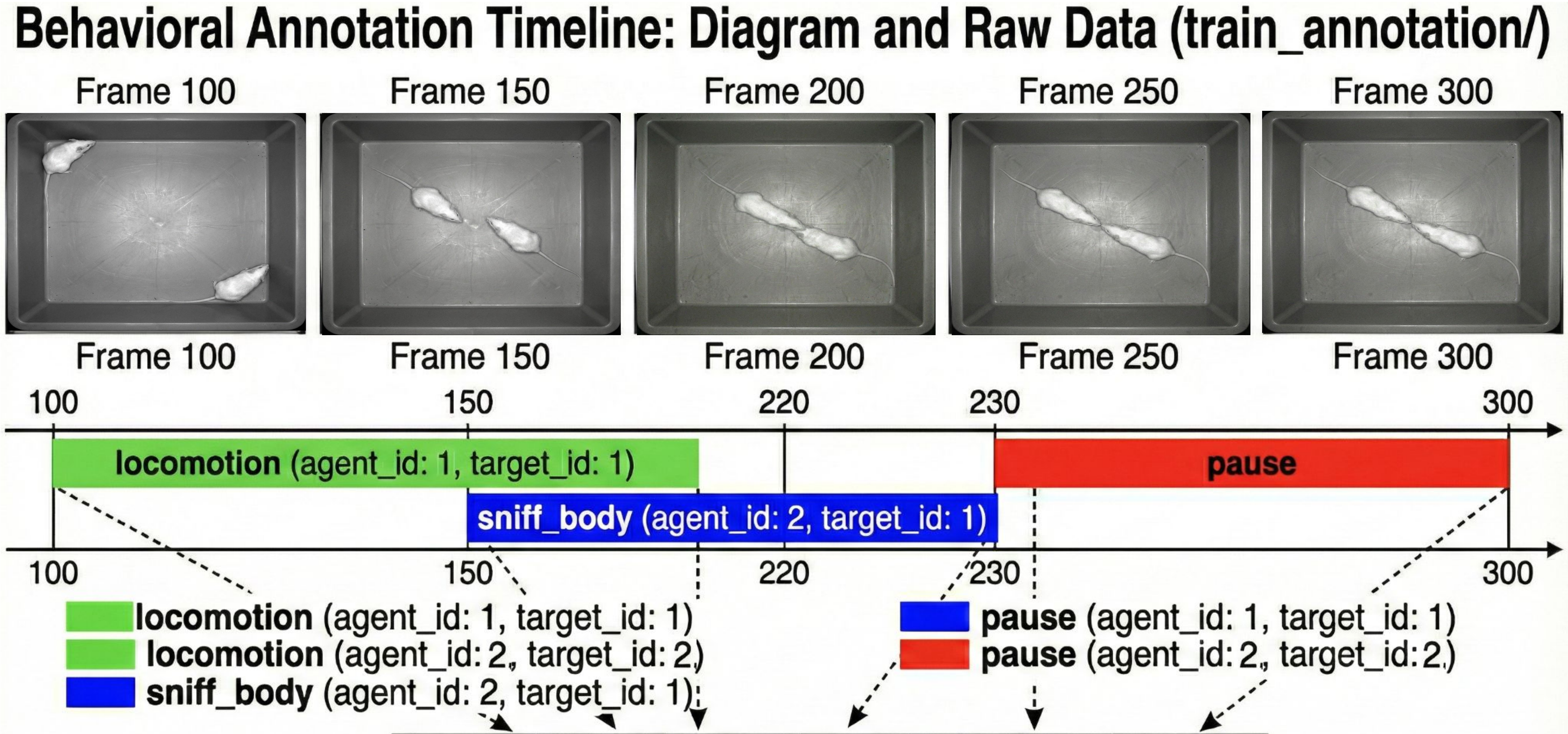
Data’s Key Features:



train_tracking/AdaptableSnail/44566-106.parquet				
video_frame	mouse_id	bodypart	x	y
100	1	nose	855.2	210.4
100	1	leftear	840.1	235.8
100	1	rightear	870.5	232.1
100	1	body_center	858.3	280.6
100	1	tail_base	862.9	330.2
100	1	tail_midpoint	875.4	385.1
100	1	tail_tip	888.7	440.5

Image and arrows generated by Copilot on MS PPT.

Data’s Key Features—Target Variable



video_id	agent_id	target_id	action	start_frame	stop_frame
44566106	1	1	locomotion	100	180
44566106	2	1	sniff_body	150	220

Image generated by Copilot on MS PPT (not including bottom table)

Submission Format: Raw Data (sample_submission.csv)

row_id	video_id	agent_id	target_id	action	start_frame	stop_frame
1	44566106	1	1	locomotion	100	180
2	44566106	2	1	sniff_body	150	220
3	44566106	1	1	pause	230	300
4	44566106	3	3	grooming	350	400
5	44566106	4	3	chasing	420	480

Preprocessing Pipeline

Preprocessing and Organizing Mouse Behavior Data

- Organize tracking data capturing mouse positions per video frame
- Manage annotation data with detailed behavior labels for each frame
- Index data using video ID, agent and target mouse IDs, and frame number
- Track multiple body parts including ears, nose, neck, and tail bases
- Differentiate behaviors into self-actions and pairwise interactions
- Preprocess labels by extracting agents and behaviors from raw CSV files

(AI generated image)

Advanced Feature Engineering for Mouse Behavior Analysis

- Quantify self features such as distances, speeds, and body orientation.
- Analyze curvature, turning rates, and multiscale speed statistics.
- Capture movement states and transitions with smoothed position data.
- Extract pair features including distances and speeds between two mice.
- Aggregate interaction metrics like rolling distances for agent-target pairs.
- Implement a pipeline to extract and save features per video for training.

(AI generated image)

Preprocessing Code Snippets

```python
def generate_speed_features(mouse_A, mouse_B, body_parts_tracked, interval=10):
    important_parts = ['tail_base', 'nose', 'ear_left', 'body_center']
    parts = list(set(important_parts) & set(body_parts_tracked))

    # Schema enforcement: fill missing parts with NaN
    for part in parts:
        if part not in mouse_A.columns:
            mouse_A[part] = np.nan
        if part not in mouse_B.columns:
            mouse_B[part] = np.nan

    features = {}

    # Precompute shifted positions
    shifted_A = {part: mouse_A[part].shift(interval) for part in parts}
    shifted_B = {part: mouse_B[part].shift(interval) for part in parts}

    # Precompute speeds
    speed_A = {part: np.square(mouse_A[part] - shifted_A[part]).sum(axis=1, skipna=False) for part in parts}
    speed_B = {part: np.square(mouse_B[part] - shifted_B[part]).sum(axis=1, skipna=False) for part in parts}

    for partA, partB in itertools.product(parts, repeat=2):
        # Speed features
        features[f'speed_{partA}'] = speed_A[partA]
        features[f'speed_{partB}'] = speed_B[partB]
        features[f'speed_{partA}{partB}'] = np.square(mouse_A[partA] - shifted_B[partB]).sum(axis=1, skipna=False)
        features[f'speed_{partB}{partA}'] = np.square(mouse_B[partB] - shifted_A[partA]).sum(axis=1, skipna=False)

    return pd.DataFrame(features)
```

Quantifies speed of both mice - this can help determine actions such as when mice chase or play with each other

```python
# Process single mouse data
if len(single_mouse_list) > 0:
    #single_mouse, single_mouse_label, single_mouse_meta = down_sampling(single_mou
    print(f"\n  🐭 Processing {len(single_mouse_list)} single mouse batches...")
    performance_metrics['single_mouse_batches'] += len(single_mouse_list)

    single_mouse = pd.concat(single_mouse_list)
    single_mouse_label = pd.concat(single_mouse_label_list)
    single_mouse_meta = pd.concat(single_mouse_meta_list)

    del single_mouse_list, single_mouse_label_list, single_mouse_meta_list

    X_tr = transform_single(single_mouse, body_parts_tracked)
    X_tr,single_mouse_label,single_mouse_meta = down_sampling(X_tr,single_mouse_lab
    print(f"\n  🐭 downsampling: {len(X_tr)} single mouse batches...")
    del single_mouse

    print(f"    Features shape: {X_tr.shape}")

    # Create feature analysis for first 2 configs
    #if create_visualizations and section <= 2:
    #    create_feature_analysis_plots(X_tr, 'single', section)
    if validate_or_submit == 'validate':
        thresholds = cross_validate_classifier_enhanced(
                binary_classifier, X_tr, single_mouse_label,
                single_mouse_meta, body_parts_tracked_str
            )
    else:
        submit(body_parts_tracked_str, 'single', binary_classifier, X_tr,
                    single_mouse_label, single_mouse_meta)

    del X_tr
    gc.collect()
```

```python
# Process pair data
if len(mouse_pair_list) > 0:
    #mouse_pair, mouse_pair_label, mouse_pair_meta = down_sampling(mouse_pair_list, mouse_pair_lab
    print(f"\n  🐭🐭 Processing {len(mouse_pair_list)} mouse pair batches...")
    performance_metrics['pair_batches'] += len(mouse_pair_list)

    mouse_pair = pd.concat(mouse_pair_list)
    mouse_pair_label = pd.concat(mouse_pair_label_list)
    mouse_pair_meta = pd.concat(mouse_pair_meta_list)

    del mouse_pair_list, mouse_pair_label_list, mouse_pair_meta_list

    X_tr = transform_pair(mouse_pair, body_parts_tracked)
    X_tr,mouse_pair_label,mouse_pair_meta = down_sampling(X_tr,mouse_pair_label,mouse_pair_meta)
    print(f"\n  🐭 downsampling: {len(X_tr)} mouse pair batches...")
    del mouse_pair

    print(f"    Features shape: {X_tr.shape}")

    # Create feature analysis for first 2 configs
    #if create_visualizations and section <= 2:
    #    create_feature_analysis_plots(X_tr, 'pair', section)
    if validate_or_submit == 'validate':
        thresholds = cross_validate_classifier_enhanced(
                binary_classifier, X_tr, mouse_pair_label,
                mouse_pair_meta, body_parts_tracked_str
            )
    else:
        submit(body_parts_tracked_str, 'pair', binary_classifier, X_tr,
                mouse_pair_label, mouse_pair_meta)

    del X_tr
    gc.collect()
```

We wrote custom code to track 1 mice (sometimes mice act on themselves) or pair of mice (ex: chase). Google Gemini was used to check our code & offer improvements.

```python
def estimate_ear_position(X, available_body_parts):

    center_parts = ['nose', 'head','neck', 'body_center', 'spine_1', 'spine_2', 'tail_base']
    ear_pairs = [('ear_left', 'ear_right'), ('ear_right', 'ear_left')]

    for ear_unknown, ear_known in ear_pairs:
        for head, tail in itertools.combinations(center_parts, 2):
            if head not in available_body_parts or tail not in available_body_parts:
                continue

            # Build mask for valid frames
            mask = (
                X[(ear_unknown, 'x')].isna() | X[(ear_unknown, 'y')].isna()
            )
            if not mask.any():
                return X

            mask &= (
                X[(ear_known, 'x')].notna() & X[(ear_known, 'y')].notna()
            ) & (
                X[(tail, 'x')].notna() & X[(tail, 'y')].notna()
            ) & (
                X[(head, 'x')].notna() & X[(head, 'y')].notna()
            )

            if not mask.any():
                continue

            # Midline vector
            dx = X.loc[mask, (head, 'x')] - X.loc[mask, (tail, 'x')]
            dy = X.loc[mask, (head, 'y')] - X.loc[mask, (tail, 'y')]
            norm = np.sqrt(dx**2 + dy**2) + 1e-6
            spine_dx = dx / norm
            spine_dy = dy / norm

            # Vector from tail to known ear
            ear_dx = X.loc[mask, (ear_known, 'x')] - X.loc[mask, (tail, 'x')]
            ear_dy = X.loc[mask, (ear_known, 'y')] - X.loc[mask, (tail, 'y')]

            # Project ear vector onto spine
            proj = ear_dx * spine_dx + ear_dy * spine_dy
            proj_x = proj * spine_dx
            proj_y = proj * spine_dy

            # Perpendicular component
            perp_x = ear_dx - proj_x
            perp_y = ear_dy - proj_y

            # Reflect across spine
            X.loc[mask, (ear_unknown, 'x')] = X.loc[mask, (tail, 'x')] + proj_x - perp_x
            X.loc[mask, (ear_unknown, 'y')] = X.loc[mask, (tail, 'y')] + proj_y - perp_y
```

```python
def estimate_nose_position(mouse_df, scale=0.75):
    # Compute ear center
    ear_left_x = mouse_df[('ear_left', 'x')]
    ear_left_y = mouse_df[('ear_left', 'y')]
    ear_right_x = mouse_df[('ear_right', 'x')]
    ear_right_y = mouse_df[('ear_right', 'y')]

    center_x = (ear_left_x + ear_right_x) / 2
    center_y = (ear_left_y + ear_right_y) / 2

    # Ear-to-ear vector
    dx = -ear_right_x + ear_left_x
    dy = -ear_right_y + ear_left_y

    # Perpendicular direction (forward-facing)
    perp_x = -dy
    perp_y = dx
    norm = np.sqrt(perp_x**2 + perp_y**2) + 1e-6
    perp_x /= norm
    perp_y /= norm

    # Estimate nose position
    ear_dist = np.sqrt(dx**2 + dy**2)
    est_x = center_x + scale * ear_dist * perp_x
    est_y = center_y + scale * ear_dist * perp_y

    # Fill only missing values
    mask_x = mouse_df[('nose', 'x')].isna()
    mask_y = mouse_df[('nose', 'y')].isna()
    mouse_df.loc[mask_x, ('nose', 'x')] = est_x[mask_x]
    mouse_df.loc[mask_y, ('nose', 'y')] = est_y[mask_y]

    return mouse_df
```

Nose / Ear position can help determine mice's location within the arena as well as helping identifying "sniff"-related actions or whether they are turning towards or away from another mice.

Code such as generating nose positions and speed features are from the Kaggle community and have been cited under References.

LightGBM Model & Final Results

Our Most Invested Model

- Idea to use LightGBM based on upon a Kaggle community post with a 26% leaderboard score (Kongyanmiao)
- Implemented advanced feature engineering and preprocessing steps as discussed in previous sections
- Focused on hyperparameter tuning to enhance model F1-score and robustness
- Achieved an improved leaderboard score of of 38% (see below)

Custom LightGBM implementation of code w/ hyperparameters

```
# Create classifier
binary_classifier = make_pipeline(
    SimpleImputer(),
    TrainOnSubsetClassifier(
        lgb.LGBMClassifier(
            n_estimators=200,
            learning_rate=0.03,
            min_child_samples=40,
            num_leaves=31,
            max_depth=-1,
            subsample=0.8,
            colsample_bytree=0.8,
            verbose=-1,
            random_state=0
        )
        ,100000
    )
)
```

Competition Notebook
MABe Challenge - Social Action Recogni...

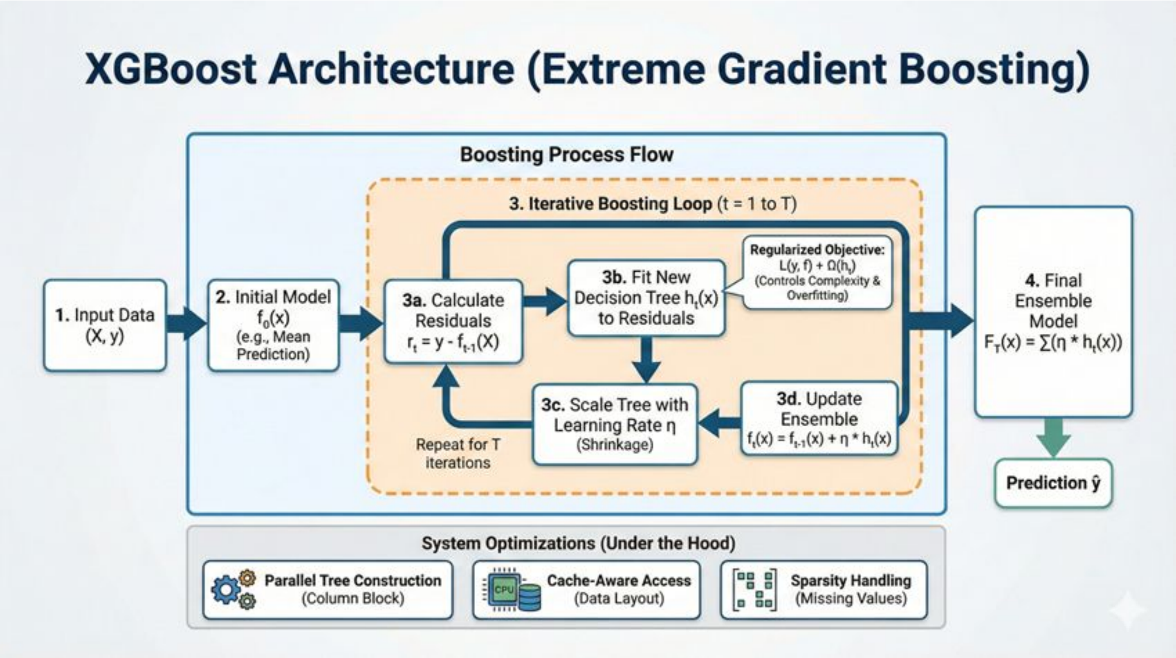
Public Score
0.376

Best Score
0.380 V31

Version 33 of 33

Runtime
55m 30s

Training Model Setup and Optimization

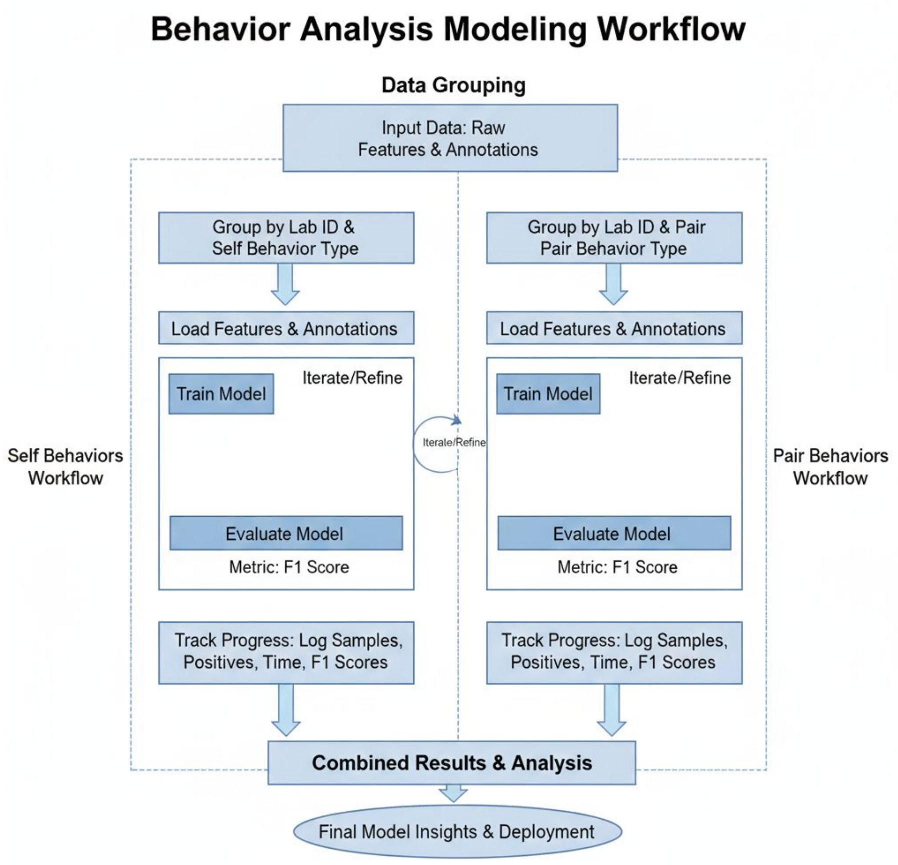


(AI generated image using MS PPT Copilot)

- Use XGBoost with binary logistic objective for classification
- Employ StratifiedGroupKFold cross-validation grouped by video ID.
- Address class imbalance using scale_pos_weight parameter.
- Hyperparameters:
  - Learning Rate = 0.05
  - Max depth = 6
  - Subsample | colsample = 0.8.
- Apply early stopping after 10 rounds without improvement to prevent overfitting.
- Generate outputs per fold including model files, thresholds, and performance plots.

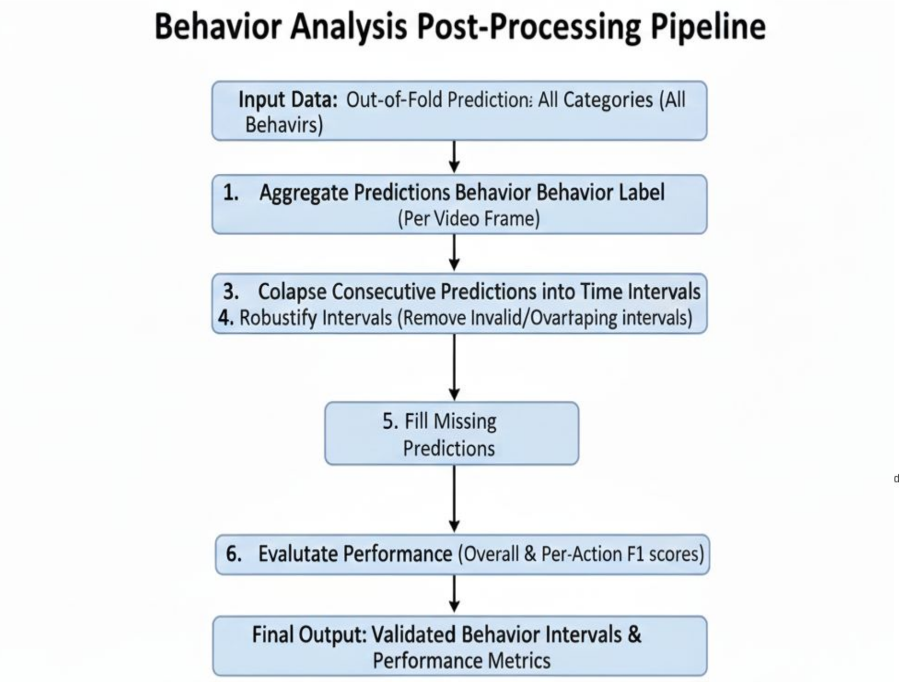
Training Execution Workflow for Mouse Behaviors

- Group data by lab ID and behavior type for self and pair behaviors.
- Load respective features and annotations for each behavior group.
- Train and evaluate models per behavior with F1 scores logged.
- Track progress with detailed tables showing samples, positives, and elapsed time.
- Separate workflows for self behaviors and pairwise interactions ensure accuracy.



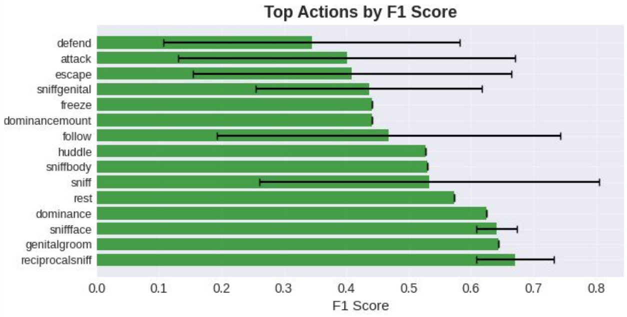
(AI generated flow chart)

Validation & Post-Processing for Reliable Predictions



(AI generated image)

- Apply methods to remove invalid and overlapping intervals.
- Aggregate out-of-fold predictions across all behavior categories, and select the most confident behavior label for each video frame.
- Collapse consecutive identical predictions into defined intervals.
- Fill missing predictions for videos lacking output data.
- Evaluate performance using overall and per-action F1 scores.



Challenges in Mouse Behavior Analysis

(AI generated image)

- Inconsistent frame labeling by lab annotators and diverse mouse body part definitions complicate data normalization across labs
- System memory (even on Colab Pro) limits restrict features to under 300, demanding careful selection and fine tuning
- Missing key body part positions limits model choices to tree-based methods (excl. deep neural networks)
- Rare positive action cases (~10%) make differentiating behaviors very difficult
- Cross-validation results often do not correlate with leaderboard performance

#	Team	Members	Score	Entries	Last	Join
507	JHU-Team 1		0.446	65	2d	

References

- Géron, A. (2022). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems (3rd ed.). O’Reilly Media. https://www.oreilly.com/library/view/hands-on-machine-learning/9781098125967/
- Sun, J. J., Marks, M., Ulmer, A. W., Chakraborty, D., Geuther, B., Hayes, E., Jia, H., Kumar, V., Oleszko, S., Partridge, Z., Peelman, M., Robie, A., Schretter, C. E., Sheppard, K., Sun, C., Uttarwar, P., Wagner, J. M., Werner, E., Parker, J., Perona, P., Yue, Y., Branson, K., & Kennedy, A. (2023). MABe22: A multi-species multi-task benchmark for learned representations of behavior. arXiv. https://doi.org/10.48550/arXiv.2207.10553
- Sun, J. J., Karigo, T., Wild, B., Chakraborty, D., Mohanty, S. P., Sun, Q., Chen, C., Anderson, D. J., Perona, P., Yue, Y., & Kennedy, A. (2021). The multi-agent behavior dataset: Mouse dyadic social interactions. arXiv. https://doi.org/10.48550/arXiv.2104.02710.

References (Kaggle

- AmbrosM. (n.d.). *MABe Validated baseline without machine learning*. Kaggle. Retrieved December 4, 2025, from https://www.kaggle.com/code/ambrosm/mabe-validated-baseline-without-machine-learning
- Cody11null. (n.d.). *Squeeze GBT*. Kaggle. Retrieved December 4, 2025, from https://www.kaggle.com/code/cody11null/squeeze-gbt
- Hutch1221. (n.d.). *MABe-Starter-Train (ja)*. Kaggle. Retrieved December 4, 2025, from https://www.kaggle.com/code/hutch1221/mabe-starter-train-ja
- Hutch1221. (n.d.). *MABe-Starter-Inference (ja)*. Kaggle. Retrieved December 4, 2025, from https://www.kaggle.com/code/hutch1221/mabe-starter-inference-ja
- Kongyanmiao. (n.d.). *Social Action Recognition in Mice | LightGBM*. Kaggle. Retrieved December 4, 2025, from https://www.kaggle.com/code/kongyanmiao/social-action-recognition-in-mice-lightgbm
- Author unknown. (n.d.). *Lim from lambda to complexity: Organized book edition (Google v1 alpha, LightDM)*. Retrieved from https://example.com/lim-from-lambda-to-complexity-organized-book-edition-google-v1-alpha-lightdm
- AmbrosM. (n.d.). MABe EDA which makes sense. Kaggle. Retrieved December 4, 2025, from https://www.kaggle.com/code/ambrosm/mabe-eda-which-makes-sense
- AmbrosM. (n.d.). *MABe Nearest Neighbors: The Original*. Kaggle. Retrieved December 4, 2025, from https://www.kaggle.com/code/ambrosm/mabe-nearest-neighbors-the-original

Appendix

This slide is equivalent to “Plan for Next Week” for class presentation.

TensorFlow Preprocessing Pipeline

- Major bottleneck - 8789 video tracking datasets available in .parquet format
- Max RAM available on Google Colab Pro is <13 GB
- Team’s solution was to create a TF pipeline that loads directly from .parquet files. Main advantage:
    1) Compatibility with LightGBM package (our current model)
    2) Native-support with TF’s deep learning models (next model for the team to try)
    3) Training data in batches in parallel that can be integrated with GPU | TPUs in Google Colab
-

Custom-built TF Flow for MABe Tracking Data

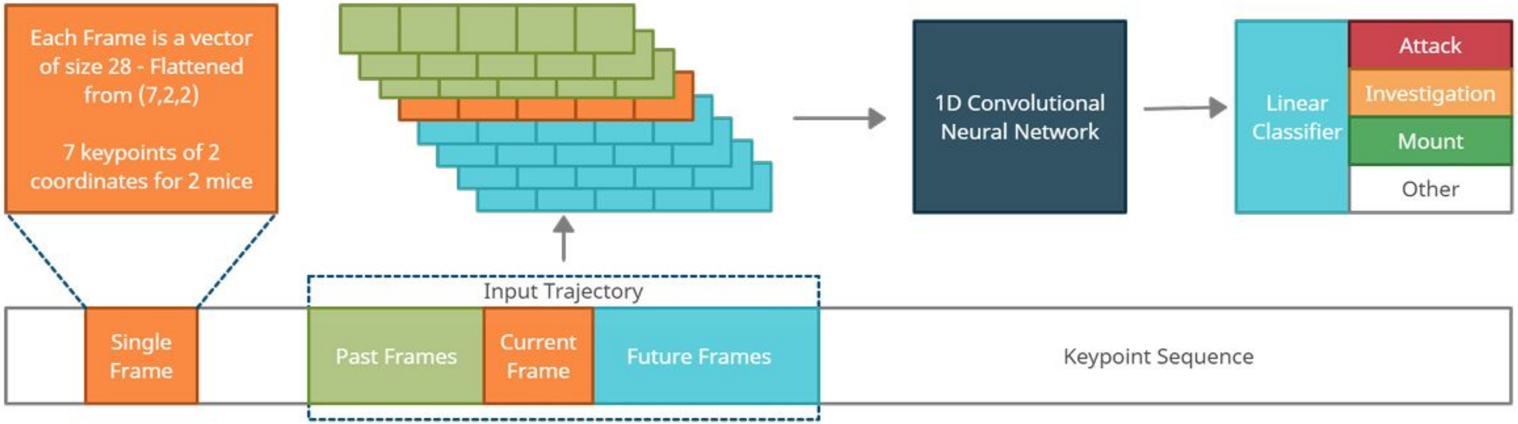
```python
import pyarrow.parquet as pq
import tensorflow as tf

def parquet_generator(files, regex_start):
    print(f"Concatenating {len(files)} files...")

    for file in files:
      print(f"Reading file: {file}")
      parquet_file = pq.ParquetFile(file)

      for batch in parquet_file.iter_batches(batch_size=1024, use_threads=True):
          df = batch.to_pandas().assign(
            lab_id=re.search(regex_start + "/([^/]+)/", file).group(1),
            video_id=re.search(r"(\d+)(?=\.parquet)", file).group(1))

          for row in df.itertuples(index=False):
              yield dict(row._asdict())

dataset_tf = tf.data.Dataset.from_generator(
    lambda: parquet_generator(files=train_tracking, regex_start="train_tracking"),
    output_signature={
        "video_frame": tf.TensorSpec(shape=(), dtype=tf.int32),
        "mouse_id": tf.TensorSpec(shape=(), dtype=tf.int32),
        "bodypart": tf.TensorSpec(shape=(), dtype=tf.string),
        "x": tf.TensorSpec(shape=(), dtype=tf.float32),
        "y": tf.TensorSpec(shape=(), dtype=tf.float32),
        "lab_id": tf.TensorSpec(shape=(), dtype=tf.string),
        "video_id": tf.TensorSpec(shape=(), dtype=tf.int32)
    }).batch(128).prefetch(tf.data.AUTOTUNE)
```

DNN Architecture

- Using neural net models tend to outperform our LightGBM solution
- 1D CNN performs the best based on authors of the original study
- Features the authors considered: distances, velocities, and angles between coordinates relative to the agents’ orientations.
- Hyperparameters considered includes number of input frames, frame skips, units per layer, and learning rate.

Method	Average F1	MAP
Fully Connected	$.659 \pm .005$	$.726 \pm .004$
LSTM	$.675 \pm .011$	$.712 \pm .013$
Self-Attention	$.610 \pm .028$	$.644 \pm .018$
1D Conv Net	$.793 \pm .011$	$.856 \pm .010$

Table 1: Class-averaged results on Task 1 (attack, investigation, mount) for different baseline model architectures. The value is shown of the mean and standard deviation over 5 runs.



All images are from Sun et. al from their paper: MABe22: A multi-species multi-task benchmark for learned representations of behavior.
Note: MAP - mean average precisions results - averages the number of