ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

BÁO CÁO BÀI TẬP NHÓM MÔN HỌC MÁY

23020037 Lê Minh Đạt
23020079 Bùi An Huy
23020028 Nguyễn Xuân Dũng

HÀ NỘI - 2025

MABe kaggle competition report

Nguyen Xuan Dung
ID: 23020028
VNU University of Eng. & Tech.
Hanoi, Vietnam

Le Minh Dat
ID: 23020037
VNU University of Eng. & Tech.
Hanoi, Vietnam

Bui An Huy
ID: 23020079
VNU University of Eng. & Tech.
Hanoi, Vietnam

***Tóm tắt nội dung*—This report presents our attemps at the Kaggle Mouse Social Action Recognition challenge (MABe). The goal is to label mouse behaviors based on tracking data converted into a tabular format. By analyzing the dataset, we identified significant challenges regarding data sparsity and heterogeneity. Our attempts include improvements upon the baseline XGBoost method by creating a Soft-voting model combining XGBoost, CatBoost, and LightGBM. Key technical contributions include a Stratified Subsampling strategy to handle class imbalance, advanced feature engineering (Egocentric coordinates, Contextual lag/lead), Linear Scan threshold tuning, and a robust feature mapping mechanism. Another side attempt is to train a LSTM model from scratch; this attempt is incomplete due to inexperience and time constraints.**

***Index Terms*—Mouse Behavior, Ensemble Learning, Stratified Subsampling, Feature Engineering, MABe Challenge.**

TÀI LIỆU

[1] MABe Challenge Dataset, [Online]. Available: Kaggle.
[2] "Social Action Recognition in Mice | XGBoost", Kaggle Notebook.
[3] "MABe Nearest Neighbors: The Original", https://www.kaggle.com/code/ambrosm/mabe-nearest-neighbors-the-original
[4] "MABe EDA which makes sense", https://www.kaggle.com/code/ambrosm/mabe-eda-which-makes-sense
[5] "MABe: An Exploratory Data Safari", https://www.kaggle.com/code/antoninadolgorukova/mabe-an-exploratory-data-safari

I. Problem Understanding and Data Analysis

A. Problem Statement

The core problem is to train a machine learning model capable of automatically labeling mouse behaviors (e.g., attack, investigation, rear) based on video tracking data that has been converted into a tabular format (keypoint coordinates over time).

B. Data Insights

Through the analysis of the problem statement, reading public Exploratory Data Analysis (EDA) works and self EDA, we extracted the following insights:

1) On Data Structure: The dataset is composed of three main parts:

- Video metadata (video ID, lab ID, labeled behaviors).
- Tracking data (frame-by-frame coordinates of body parts).
- Annotation data (start and end frames for specific behaviors).

2) On Behavior: Mouse behaviors into two types to optimize feature engineering: Self behaviors (actions on oneself) and Pair behaviors (actions from an agent mouse to a target mouse). Each lab has their own set of behaviors. The metadata file (`train.csv`) lists the tracked behaviors in the form of (Agent, Behavior, Target). This means the annotation data may not cover all behaviors that actually happened in the video.

3) Data Sparsity and Validity: A significant observation is that even though the training metadata file contains a large number of rows, most of them have no tracked behaviors (as in, Behaviors_labeled = (`None`), implying no ground truth labels for training. By applying a simple filter, we isolated about **800 valid videos** out of over 8000 entries.

4) Heterogeneity and Quality Issues: The data varies significantly across different laboratories:

- Folders like *MABe22_keypoints* and *MABe22_movies* lack corresponding tracking data.
- Different labs track different body parts and different sets of behaviors
- There are missing values (e.g. missing coordinates for certain body parts, missing mouses), which necessitated a robust pre-processing pipeline.

II. Notebook Implementation Summary

This section outlines the technical implementation details from our provided notebook (`version-7 (2).ipynb`).

A. Configuration and Initialization

- **Environment Setup:** Imported essential libraries including `xgboost`, `catboost`, `lightgbm`, `polars`, and `sklearn`.
- **Configuration (CFG Class):** Defined data paths, execution modes (`Debug` vs. `Submit`), and model hyperparameters (e.g., `n_estimators=300`).

B. Modeling Strategy

- **Ensemble Learning:** Implemented `get_model_zoo()` to instantiate three SOTA models: **XGBoost**, **CatBoost**, and **LightGBM**. The final prediction utilizes Soft Voting (probability averaging).
- **Stratified Subsampling:** Defined the `StratifiedSubsetClassifier` class to handle class imbalance by downsampling the majority class (no-action) while preserving minority classes, optimizing both speed and accuracy.

C. Feature Engineering Pipeline

- **Preprocessing:** Applied **Savitzky-Golay filters** to smooth tracking jitter.
- **Egocentric Coordinates:** Used `add_egocentric_features` to rotate coordinates, ensuring Rotation Invariance.
- **Physics-based Features:**
  - *Kinematics:* Velocity, acceleration, curvature, angular velocity.
  - *Social Physics:* Time-to-Collision (TTC) and Chase Score.
  - *Morphology:* Head triangle area (for Rearing) and body elongation.
- **Contextual Features:** Used `add_context` to append Lag/Lead features (past/future frames).

D. Training and Validation Process

- **Training Loop:** Iterated through data sections grouped by tracked body parts.
- **Validation Strategy:** Employed **Stratified Group K-Fold** to prevent data leakage.
- **Feature Mapping:** Saved the feature schema during training to resolve column mismatches during inference.
- **Threshold Tuning:** Utilized a **Linear Scan** algorithm to find the optimal probability threshold for maximizing the F1-Score.

E. Inference and Post-processing

- **Prediction:** Loaded trained models and artifacts for inference on the Test set.
- **Smoothing:** Implemented `smooth_predictions` to filter short noise (< 10 frames) and bridge small gaps (< 5 frames).
- **Robustify:** Cleaned predictions to resolve logical errors (e.g., $Start > End$) and remove overlapping segments.

III. Data Preprocessing and Feature Engineering

Raw tracking data from pose estimation systems often contains noise, missing values, and high-frequency jitter. To transform this raw input into a meaningful state space for machine learning, we implemented a rigorous preprocessing pipeline followed by extraction of kinematic and morphological features.

A. Data Preprocessing

1) Imputation and Normalization: The tracking data contains missing frames due to occlusion. We applied linear interpolation to fill gaps shorter than 5 frames. Subsequently, we normalized all spatial coordinates from pixels to centimeters using the `pix_per_cm_approx` factor provided in the metadata. This ensures that the model learns physical distances rather than resolution-dependent values.

2) Noise Reduction via Savitzky-Golay Filter: Raw keypoint trajectories exhibit high-frequency jitter which can artificially inflate velocity and acceleration calculations. To mitigate this, we applied a **Savitzky-Golay filter**. Unlike simple moving averages, this filter preserves the high-frequency components of the signal (such as rapid attack lunges) while attenuating noise. For a sequence of coordinates $x[n]$, the smoothed value $y[n]$ is given by:

$$y[n] = \sum_{i=-(M-1)/2}^{(M-1)/2} C_i x[n+i] \tag{1}$$

We utilized a window length $M = 7$ and a polynomial order $k = 2$. **Benefit:** This step significantly improved the signal-to-noise ratio for derivative-based features (velocity, acceleration).

B. Feature Extraction

We moved beyond simple coordinates to engineer features that capture the physics, morphology, and social dynamics of the mice.

1) Kinematic Features (Motion Dynamics): We computed the first and second derivatives of position to obtain velocity (v) and acceleration (a). Additionally, we derived:

- **Angular Velocity (ω):** Quantifies how fast the mouse changes its heading direction. High ω combined with low linear velocity is indicative of *Tussling* or *Grooming*.

$$\omega_t = \frac{\theta_t - \theta_{t-1}}{\Delta t}, \quad \text{where } \theta_t = \text{atan2}(v_y, v_x) \tag{2}$$

- **Tortuosity (Path Efficiency):** To distinguish between directed locomotion (Chase) and meandering (Explore), we calculated the ratio of displacement to total path length over a window W.

2) Morphological Features (Shape Analysis): Certain behaviors are defined by posture rather than movement. We engineered features to capture body deformation:

- **Body Elongation:** Defined as the ratio between the body length (Nose to Tail-base) and body width (Left Ear to Right Ear).

$$R_{elongation} = \frac{\|\mathbf{p}_{nose} - \mathbf{p}_{tail}\|}{\|\mathbf{p}_{earL} - \mathbf{p}_{earR}\| + \epsilon} \tag{3}$$

  A high ratio indicates running/stretching, while a low ratio indicates huddling or rearing.
- **Head Triangle Area:** To detect *Rearing* (standing up), where the 2D projection of the head shrinks, we computed the area of the triangle formed by the nose and ears using the Shoelace formula.

3) Social Physics (Interaction Features): For pairwise behaviors, relative dynamics are crucial. We modeled the interaction using "Social Physics"concepts:

- **Time-to-Collision (TTC):** A critical feature for aggressive behaviors. It estimates the time until impact based on current closing speed v_{close}.

$$TTC = \begin{cases} \frac{d_{AB}}{v_{close}} & \text{if } v_{close} > 0 \text{ and } d_{AB} < 30cm \\ T_{max} & \text{otherwise} \end{cases} \tag{4}$$

where d_{AB} is the Euclidean distance between mice. A low TTC triggers an "Imminent Collision"flag, strongly correlated with *Attack*.

- **Chase Score:** We refined the chase detection by requiring both high speed and directional alignment.

$$S_{chase} = \|\mathbf{v}_A\| \times \text{Cosine}(\mathbf{v}_A, \mathbf{p}_B - \mathbf{p}_A) \quad (5)$$

This filters out situations where mice are simply moving parallel to each other without interaction.

4) Contextual Features: Recognizing that behavior is a temporal sequence, we augmented the feature space with Lag/Lead information. For every feature f_t, we appended $[f_{t-10}, f_{t+10}, \Delta f]$ to the input vector. This allows the classifier to utilize "future"information (available in offline processing) to disambiguate the onset and offset of actions.

IV. Methodology and Modeling Strategy

After transforming the raw tracking data into a rich feature space (as detailed in Section II), our focus shifted to constructing a robust classification pipeline. This section details our modeling architecture, training strategy to handle class imbalance, and optimization techniques.

A. Coordinate Transformation: Egocentric Frame

Before feeding data into the models, a critical step was establishing **Rotation Invariance**. We mapped global coordinates (x, y) into a local egocentric frame attached to the agent mouse. Let θ be the mouse's orientation angle derived from the tail-to-nose vector. The transformation of any target point (x_t, y_t) is given by the rotation matrix $R(\theta)$:

$$\begin{bmatrix} x_{ego} \\ y_{ego} \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_t - x_{center} \\ y_t - y_{center} \end{bmatrix} \quad (6)$$

This allows the models to learn spatial logic (e.g., "target is directly in front") regardless of the mouse's absolute position or direction in the arena.

B. Ensemble Learning Architecture

Instead of relying on a single model, we established a heterogeneous **Ensemble Learning** framework employing a Soft Voting mechanism. We combined three state-of-the-art Gradient Boosting Decision Tree (GBDT) libraries, each bringing unique strengths:

- **XGBoost:** Utilized for its robust regularization (Ω) to reduce variance and prevent overfitting.
- **CatBoost:** Selected for its superior handling of noisy data and categorical features via Ordered Boosting.
- **LightGBM:** Incorporated for its Gradient-based One-Side Sampling (GOSS) to optimize training speed on large datasets.

The final prediction probability $P_{ensemble}$ for a class c is the arithmetic mean of the individual calibrated probabilities P_k:

$$P_{ensemble}(x) = \frac{1}{3} \sum_{k \in \{XGB, CAT, LGBM\}} P_k(y = c|x) \quad (7)$$

This averaging technique effectively cancels out the biases of individual models, leading to a more stable decision boundary on the test set.

C. Stratified Subsampling Strategy

A major challenge in the MABe dataset is extreme class imbalance. The "no-action"class constitutes over 99% of the frames, while behaviors like *Attack* or *Mount* are rare. Training on the full dataset would bias the model towards the majority class and is computationally wasteful.

We implemented a custom `StratifiedSubsetClassifier`. This wrapper dynamically undersamples the majority class while preserving 100% of the minority classes. Let N_{pos} be the number of positive samples. We sample a subset of the negative class N'_{neg} such that:

$$N'_{neg} \approx \alpha \cdot N_{pos} \quad (\text{where } \alpha \text{ is the sampling ratio}) \quad (8)$$

This strategy drastically reduced training time (enabling the use of computationally expensive features) while maintaining high sensitivity for rare behaviors.

D. Optimization and Post-Processing

1) Threshold Tuning via Linear Scan: GBDT models output continuous probabilities. The standard threshold of 0.5 is suboptimal for maximizing the F1-score on imbalanced data. Instead of complex Bayesian optimization, we employed a efficient **Linear Scan Algorithm**. We discretized the threshold space $T = \{0.1, 0.12, \ldots, 0.9\}$ and selected the optimal τ^* for each specific behavior class by maximizing the F1-score on the validation set:

$$\tau_c^* = \underset{\tau \in T}{\text{argmax}} \, \text{F1}(\text{ValidSet}, \tau) \quad (9)$$

2) Temporal Smoothing (Morphological Operation): Biological behaviors possess temporal continuity; an action like "Attack"does not occur for a single millisecond. However, raw model predictions often contain high-frequency noise (1-frame spikes) or micro-gaps. To address this, we applied a morphological closing-like operation (‘smooth_predictions‘):

- **Filtering (Erosion):** Discard action segments with duration $L < 10$ frames ($\approx 0.3s$) to remove noise.
- **Bridging (Dilation):** Merge two segments of the same action if the gap between them is < 5 frames.

This post-processing step aligns the predictions with biological constraints, significantly boosting the final leaderboard score.

3) Robustness: Feature Mapping: A common technical issue in tracking-based tasks is feature mismatch between training and testing sets (e.g., a specific body part tracking is missing in a test video). We implemented a **Feature Mapping** mechanism where the training schema $\mathcal{S}$ is serialized. During inference, the test data matrix $\mathcal{X}_{test}$ is strictly aligned to $\mathcal{S}$ by padding missing columns with zeros, ensuring runtime stability.

V. Methodology and Modeling Strategy

A. Ensemble Learning Architecture

Motivation: The baseline solution utilized a single XGBoost model, which is susceptible to high variance and may overfit to specific noise patterns in the tracking data. To address this, we developed a heterogeneous **Ensemble Learning** framework.

Model Components: We combined three state-of-the-art Gradient Boosting Decision Tree (GBDT) libraries, leveraging their unique strengths:

- **XGBoost:** Selected for its robust regularization parameters (λ, α) which effectively penalize complex models and reduce variance.
- **CatBoost:** Utilized for its "Ordered Boosting"technique, which is particularly effective at handling noisy data and preventing target leakage in time-series data.
- **LightGBM:** Incorporated for its Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB), optimizing training speed and memory efficiency on large datasets.

Soft Voting Mechanism: Instead of hard voting (class labels), we employed Soft Voting to aggregate the predicted probabilities. Let $P_k(y = c|x)$ be the probability output of model k for class c. The final ensemble probability P_{final} is computed as the arithmetic mean:

$$P_{final}(c) = \frac{1}{3}\left(P_{XGB}(c) + P_{Cat}(c) + P_{LGBM}(c)\right) \quad (10)$$

This averaging technique smooths out the decision boundary, significantly improving the model's generalization capability on the unseen Test set.

B. Stratified Subsampling Strategy

The MABe dataset exhibits extreme class imbalance, where the "no-action"class dominates ($> 99\%$). Training on the full dataset biases the model towards the majority class. We implemented a custom `StratifiedSubsetClassifier` that dynamically downsamples the majority class while preserving 100% of the minority classes (e.g., *Attack*, *Mount*). Let N_{pos} be the count of positive samples. We randomly sample the negative class N'_{neg} such that:

$$N'_{neg} \approx \alpha \cdot N_{pos} \quad (\text{where } \alpha \text{ is the sampling ratio}) \quad (11)$$

This strategy drastically reduces training time while maintaining high sensitivity for rare behaviors.

C. Advanced Feature Engineering

To feed the ensemble models with high-quality data, we engineered physics-based features:

1) Egocentric Coordinates: We transformed global coordinates (x, y) into a local egocentric frame attached to the agent mouse using the rotation matrix $R(\theta)$. This ensures **Rotation Invariance**, allowing the model to learn spatial logic (e.g., "target is directly in front") regardless of the mouse's absolute orientation.

2) Contextual Features: Recognizing that behavior is temporal, we added Lag/Lead features. We constructed the input vector X'_t by concatenating the current state with states from k frames in the past and future (X_{t-k}, X_{t+k}). This provides crucial temporal context (e.g., high velocity at $t - 10$ suggests an imminent attack).

D. Optimization and Post-Processing

1) Threshold Tuning via Linear Scan: Standard decision thresholds ($\tau = 0.5$) are suboptimal for F1-score maximization on imbalanced data. We employed a **Linear Scan Algorithm**, iterating $\tau \in [0.1, 0.9]$ with a step of 0.02 to find the optimal τ^* for each specific behavior class on the validation set.

2) Temporal Smoothing: Raw predictions often contain high-frequency noise. We applied a morphological closing-like operation (`smooth_predictions`):

- **Filtering:** Discard action segments shorter than 10 frames ($\approx 0.3s$).
- **Bridging:** Merge two segments of the same action if the gap is < 5 frames.

This aligns predictions with the biological reality of continuous behavior.

E. Robustness: Feature Mapping

A common technical issue in tracking-based tasks is feature mismatch between training and testing sets (e.g., a specific body part tracking is missing in a test video). We implemented a **Feature Mapping** mechanism where the training schema $\mathcal{S}$ is serialized. During inference, the test data matrix $\mathcal{X}_{test}$ is strictly aligned to $\mathcal{S}$ by padding missing columns with zeros, ensuring runtime stability.

F. Inference and Production Pipeline

The final execution phase of our solution involves a streamlined pipeline designed to generate submission files efficiently and reliably.

1) Artifact Loading: During the training phase, we serialize critical artifacts: the trained model weights (for XGBoost, CatBoost, LightGBM), the optimized decision thresholds τ^* for each behavior, and the feature schema $\mathcal{S}$. In the inference phase, these artifacts are loaded to ensure the test environment typically mirrors the training configuration.

2) Test Data Processing: The test video data is processed in chunks (sections) to manage memory constraints. For each video:

1) **Feature Generation:** Raw coordinates are transformed into physics-based features (Egocentric, Kinematic, Social) using the same pipeline as the training set.
2) **Schema Alignment:** The test feature matrix is aligned to the training schema $\mathcal{S}$ using the Feature Mapping mechanism described above.

3) Prediction and Aggregation: The aligned features are passed through the Ensemble. Since the task requires segment-based output (Start Frame $\rightarrow$ End Frame), we convert the frame-by-frame probabilities into segments:

1) **Probability Averaging:** Calculate $P_{avg} = \frac{1}{3}(P_{xgb} + P_{cat} + P_{lgbm})$.
2) **Thresholding:** Apply the specific threshold τ_c^* to obtain binary masks for each class.
3) **Segmentation:** Consecutive frames with value 1 are grouped into start-stop intervals.

4) Final Clean-up (Robustify): Before saving to CSV, we apply a 'robustify' protocol to satisfy competition rules:

- **Overlap Removal:** If two predicted actions for the same mouse pair overlap in time, the one with lower confidence is trimmed or removed.
- **Validity Check:** Ensure $Start\_Frame < End\_Frame$ for all segments.
- **Dummy Generation:** For videos where no action is detected (which causes submission errors in some evaluation systems), a dummy placeholder is generated if necessary (though our sensitive model rarely requires this).

VI. Methodology and Improvements

Our solution represents a significant evolution from the baseline approach. We systematically upgraded the pipeline to address key challenges such as model variance, class imbalance, and the temporal nature of behavior. Below are the detailed improvements.

A. Transition from Single Model to Ensemble Learning

Problem: The baseline model relied on a single XGBoost regressor, which is prone to high variance and potential overfitting on specific noise patterns in the tracking data.

Improvement: We upgraded to a heterogeneous **Ensemble Learning** architecture using a Soft Voting mechanism. We combined three state-of-the-art Gradient Boosting Decision Tree (GBDT) frameworks:

- **XGBoost:** Utilized for its robust regularization (Ω) which effectively reduces model variance.
- **CatBoost:** Selected for its superior handling of noisy data and categorical features (e.g., mouse ID) through Ordered Boosting.
- **LightGBM:** Incorporated for its Gradient-based One-Side Sampling (GOSS), optimizing training speed and efficiency on large datasets.

Mathematical Formulation: Let $P_k(y = 1|x)$ be the probability predicted by model k, where $k \in \{XGB, CAT, LGBM\}$. The final ensemble probability $P_{ensemble}$ is the arithmetic mean:

$$P_{ensemble}(x) = \frac{1}{3}\sum_{k=1}^{3} P_k(y = 1|x) \tag{12}$$

Benefit: Each model captures different aspects of the data manifold. Averaging their outputs cancels out individual biases, significantly reducing overfitting and improving generalization stability on the Test set.

B. Data Strategy: Stratified Subsampling

Problem: The dataset is highly imbalanced. The "no-action"class constitutes the vast majority of frames (>99%), while critical behaviors like *attack* or *mount* are rare. Training on the full dataset is computationally wasteful and biases the model toward the null class.

Improvement: We implemented a custom `StratifiedSubsetClassifier`. This wrapper automates the downsampling of the majority class while rigorously preserving the minority classes. Let D_{pos} be the set of positive samples and D_{neg} be the set of negative samples. We construct a training set $D_{train} = D_{pos} \cup D'_{neg}$, where D'_{neg} is a random subset of D_{neg} such that $|D'_{neg}| \approx \alpha \cdot |D_{pos}|$.

Benefit:

- **Efficiency:** Drastically reduces training time.
- **Sensitivity:** Ensures the model focuses on learning the nuances of rare behaviors rather than fitting the background noise.

C. Deep Feature Engineering

We moved beyond raw coordinates to engineering features that capture the physics and context of movement.

1) Egocentric Features (Rotation Invariance): **Implementation:** We implemented `add_egocentric_features` to transform global coordinates (x, y) into a local frame attached to the mouse. Let the mouse's orientation angle be θ. The transformation of a target point (x_t, y_t) relative to the mouse center (x_c, y_c) is given by the rotation matrix:

$$\begin{bmatrix} x_{ego} \\ y_{ego} \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_t - x_c \\ y_t - y_c \end{bmatrix} \tag{13}$$

Benefit: This achieves **Rotation Invariance**, allowing the model to recognize spatial relationships (e.g., "target is in front") regardless of the mouse's absolute orientation in the cage.

2) Context Features (Temporal Awareness): **Implementation:** Behavior is not instantaneous but a sequence. We added `add_context` to incorporate Lag/Lead information. We constructed a feature vector X'_t by concatenating the current state with states from k frames in the past ('prev10') and future ('next10'):

$$X'_t = [X_{t-k}, \ldots, X_t, \ldots, X_{t+k}] \tag{14}$$

Benefit: This provides temporal context. For instance, a high velocity at $t - 10$ is a strong predictor that the current frame t might be part of an *attack* sequence.

3) Behavior-Specific Physics: **Implementation:** We encoded domain knowledge into mathematical formulas:

- **Head Geometry for Rearing:** To detect standing (rearing), we calculate the area of the triangle formed by

the nose (N), left ear (L), and right ear (R) using the Shoelace formula:

$$Area = \frac{1}{2}|x_N(y_L - y_R) + x_L(y_R - y_N) + x_R(y_N - y_L)| \tag{15}$$

A rapid contraction of this projected area indicates a change in head pitch consistent with rearing.

- **Chase Vector Alignment:** To detect chasing, we compute the alignment between the agent's velocity vector $\vec{v}$ and the vector pointing to the target $\vec{r}$:

$$Alignment = \frac{\vec{v} \cdot \vec{r}}{\|\vec{v}\|\|\vec{r}\|} \tag{16}$$

D. Threshold Tuning via Linear Scan

Problem: The default decision threshold of 0.5 is suboptimal for F1-score maximization in imbalanced datasets. Bayesian optimization (e.g., Optuna) is often slow and prone to overfitting the validation set.

Improvement: We employed a **Linear Scan** algorithm. We discretize the threshold space $T = \{0.1, 0.12, \ldots, 0.9\}$ and select the optimal τ^* for each class c:

$$\tau_c^* = \underset{\tau \in T}{\operatorname{argmax}} \, \text{F1}(\text{ValidSet}, \tau) \tag{17}$$

Benefit: This method is computationally efficient, deterministic, and allows for intuitive control over the precision-recall trade-off.

E. Post-Processing Smoothing

Problem: Raw predictions often contain high-frequency noise (1-frame spikes) or fragmented segments. **Improvement:** We introduced a `smooth_predictions` function that applies morphological operations on the time series:

- **Filtering:** Remove action segments with duration $L < 10$ frames (noise).
- **Bridging:** Merge two segments of the same action if the gap between them is < 5 frames.

Benefit: This aligns the predictions with biological reality, where behaviors are continuous and have a minimum duration, directly boosting the metric.

F. Robustness: Feature Mapping

Problem: Dynamic feature generation can lead to a mismatch in columns between Train and Test sets (e.g., a specific body part missing in a Test video). **Improvement:** We implemented a **Feature Mapping** mechanism. The exact feature schema $\mathcal{S}$ from training is serialized. During inference, the test data $\mathcal{X}_{test}$ is enforced to conform to $\mathcal{S}$:

$$\mathcal{X}_{test}^{(j)} = \begin{cases} \mathcal{X}_{test}^{(j)} & \text{if } j \in \mathcal{S} \\ 0 & \text{if } j \notin \mathcal{X}_{test} \text{ (padding)} \end{cases} \tag{18}$$

Benefit: This prevents runtime dimensional errors and ensures the model can robustly handle varied input qualities during submission.

VII. A SIDE ATTEMPT AT AN LSTM MODEL

This section details an attempt at creating a LSTM model from scratch. However, the implementation is currently non-functional due to unmitigated class imbalance (leading to model collapse), dimension mismatch errors in the data windowing process, and an incompatible mix of PyTorch data loaders with Keras model architecture.

A. Inspiration

The MABe problem can be considered a temporal problem, which models like LSTM or GRU excels at. At the time of creation, we could not found a working base LSTM notebook to improve upon, therefore we decided to create our own.

B. What we did

- **Data pre-processing**: This step remains largely the same as in the XGBoost notebook before: filter out videos with no annotations, fill or drop rows with missing data. However, we could not implement any mechanisms to prevent class imbalance.
- **Model idea**: The model does not output a single prediction. Instead, it has **16 separate output layers**. Since there are 4 mouses, **4 outputs** are dedicated to predicting **self actions**, and **12 outputs** for **pair actions**. Each output corresponds to a multiclass prediction, predicting its own set of behaviors (there are 11 self-actions and 26 pair-actions available from all videos)
- **Feature engineering**: This feature remains largely unimplemented. Due to inexperience in handling deep learning libraries, we could, we could not create a dataset from the data provided.
- **Model training and evaluation**: Since we have not implemented mechanisms to prevent class imbalance, the model collapsed, predicting mostly "class 0" (no action) for all outputs. We have also only tested the model on a small set of video data.

VIII. TEAM CONTRIBUTIONS

Bảng I
CONTRIBUTIONS OF MEMBERS

Member	Main Responsibilities
Nguyen Xuan Dung	Feature Engineering (Egocentric, Context), Mathematical formulation of physics features.
Le Minh Dat	Implementation of Ensemble Model (XGB, Cat, LGBM), Stratified Subsampling logic.
Bui An Huy	EDA, Data Cleaning (Filtering valid videos), Post-processing (Smoothing), Feature Mapping robustness, LSTM attempt

IX. Conclusion

Our solution effectively tackles the MABe challenge by combining the predictive power of Ensemble Learning with domain-specific feature engineering. The introduction of Stratified Subsampling ensures efficient training on imbalanced data, while robust engineering practices like Feature Mapping and Post-processing ensure the reliability of the submission pipeline.

***Tóm tắt nội dung*—This report presents our attempts at the Kaggle Mouse Social Action Recognition challenge (MABe). The goal is to label mouse behaviors based on tracking data converted into a tabular format. By analyzing the dataset, we identified significant challenges regarding data sparsity and heterogeneity. Our solution improves upon the baseline XGBoost method by creating a Soft-voting ensemble combining XGBoost, CatBoost, and LightGBM. Key technical contributions include a Stratified Subsampling strategy to handle class imbalance, advanced physics-based feature engineering (Egocentric coordinates, Contextual lag/lead), Linear Scan threshold tuning, and a robust feature mapping mechanism.**

Source code is available at: https://github.com/Dungx1107/BTL-machine-learning