Ghrelin Project Methods

Atanu Giri

September 2025

# Methods

## Video Acquisition and Preprocessing

Behavioral data were obtained from video recordings in which a single camera captured four mazes simultaneously. Each recording was therefore structured as a single video file containing the activity of animals in four spatially separated arenas. The videos can be found at [Harvard Dataverse](https://doi.org/10.7910/DVN/WHH7W2). To enable individual analysis of each maze, videos were automatically split into four quadrants using a custom Python script [split\_videos\_by\_quadrants.py](https://github.com/atanugiri/GhrelinBehaviorQuantification/blob/main/Python_scripts/Utility_functions/split_videos_by_quadrants.py). Each split video was stored and processed independently in subsequent steps [Link to data deposit]. The split videos can be found at [Harvard Dataverse](https://doi.org/10.7910/DVN/WHH7W2).

## Pose Estimation with DeepLabCut

Pose estimation was performed using **DeepLabCut (v3.0.0rc9)** [Mathis et al., 2018], running with **PyTorch v2.7.1** and CUDA acceleration on the laboratory workstation, equipped with an NVIDIA T100 GPU. Training was carried out entirely on this GPU system. After model training and generation of pose estimation output, all downstream analyses (feature extraction, statistical analysis) were conducted on a local Apple Silicon laptop. A DeepLabCut project was first initialized with representative videos. Following project creation, additional recordings were incorporated using the command:

deeplabcut.add\_new\_videos(config\_path, new\_videos, copy\_videos=True, extract\_frames=False)

For frame extraction, we sampled five frames per video using automatic uniform sampling. Each frame was cropped manually to retain only the maze region, thereby excluding irrelevant portions of the field of view:

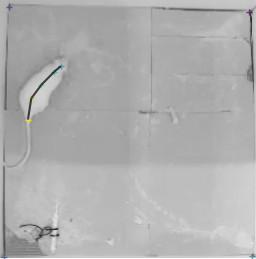
deeplabcut.extract\_frames(  
 config\_path, mode='automatic',  
 algo='uniform',  
 crop='GUI',  
 userfeedback=True  
)

The extracted frames were then labeled interactively using:

deeplabcut.label\_frames(config\_path)

This invoked the **napari-DeepLabCut GUI**, which allowed manual annotation of predefined body parts specified in the configuration file (Head, Neck, Midback, Lowerback, Tailbase, and four maze corners). Labels were visually verified with:

deeplabcut.check\_labels(config\_path)



Example of labeled frame.

before generating the training dataset with:

deeplabcut.create\_training\_dataset(config\_path)

Network training was performed with the following settings:

deeplabcut.train\_network(  
 config\_path,  
 shuffle=1,  
 trainingsetindex=0,  
 device="cuda:0",  
 max\_snapshots\_to\_keep=5,  
 displayiters=100,  
 save\_epochs=5,  
 epochs=200,  
)

A ResNet-50 backbone was used, and training proceeded for 200 epochs, with snapshots saved at 5-epoch intervals. After training, network accuracy was assessed with:

deeplabcut.evaluate\_network(config\_path, Shuffles=[3], plotting=True)

Both quantitative error metrics and diagnostic plots were inspected, comparing manual labels against predicted points across training and test frames. Once satisfactory accuracy was achieved, the trained network was applied to all videos using:

deeplabcut.analyze\_videos(  
 config\_path,  
 videos=videolist,  
 shuffle=1,  
 gputouse="cuda:0",  
 save\_as\_csv=True  
)

The resulting trajectory files ( per body part and frame) were further refined using the median filter implementation provided by DeepLabCut:

deeplabcut.filterpredictions(  
 config\_path,  
 videolist,  
 shuffle=1,  
 filtertype='median',  
 p\_bound=0.05  
)

Filtered CSV files were then used as the input for all downstream feature extraction analyses.

## Feature Extraction

### Trajectory curvature: extraction, aggregation, and statistics

#### Project structure and implementation.

All analyses were run from the project root /Users/atanugiri/Downloads/GhrelinBehaviorQuantification, which contains two primary directories: DLC-Jupyter-Notebooks/ and Python\_scripts/. The curvature feature was implemented in [trajectory\_curvature.py](https://github.com/atanugiri/GhrelinBehaviorQuantification/blob/main/Python_scripts/Feature_functions/trajectory_curvature.py) and executed from the notebook [37\_data\_analysis\_curvature.ipynb](https://github.com/atanugiri/GhrelinBehaviorQuantification/blob/main/DLC-Jupyter-Notebooks/37_data_analysis_curvature.ipynb). Source code will be made available at: <https://github.com/atanugiri/GhrelinBehaviorQuantification>.

#### Input data and normalization.

For each trial, pose trajectories (DeepLabCut outputs) were read from a PostgreSQL database table (dlc\_table). Per-trial frame rate (frame\_rate) was retrieved via SQL and used for all time derivatives. Body-part coordinates were obtained with [get\_normalized\_bodypart(trial\_id, conn, bodypart, normalize=True, interpolate=True)](https://github.com/atanugiri/GhrelinBehaviorQuantification/blob/main/Python_scripts/Data_analysis/normalized_bodypart.py), which returns temporally interpolated and spatially min–max normalized coordinates in a fixed, unit-like arena space. Unless otherwise noted, curvature was computed for the Midback body part (bodypart = ’Midback’).

#### Optional time windowing and smoothing.

Analyses can be restricted to the first seconds with a time\_limit argument (default: use the full trajectory). When smoothing is enabled, and are each convolved with a moving-average (boxcar) filter using scipy.ndimage.uniform\_filter1d. In the notebook runs shown here we used a symmetric window of window\_size = 23 samples (odd length enforced internally); smoothing was applied before derivative estimation.

#### Curvature computation.

Let and be the (optionally smoothed) coordinates, and . First and second time derivatives were computed via central differences (numpy.gradient):

Per-frame curvature was then defined as

to avoid unstable values at near-zero speeds, frames with speed below a threshold were set to . We used speed\_thresh = 1e-2 (normalized-units/s) unless stated otherwise. The per-trial summary statistic was the mean of all finite values:

where indexes frames with finite curvature.

#### Batch processing and grouping.

For group analyses, we used batch\_trajectory\_curvature(conn, trial\_ids, ...), which calls the per-trial routine and returns a table with columns id, mean\_curvature. In the notebook we defined trial-ID lists for experimental groups (e.g., Saline, Ghrelin, Inhibitory, Excitatory), computed mean\_curvature for each trial with window\_size = 23, and concatenated results into a single DataFrame with a group column.

#### Statistical testing and visualization.

Group summaries were visualized with bars (mean  SEM) using a custom plotting utility [(plot\_groupwise\_bar)](https://github.com/atanugiri/GhrelinBehaviorQuantification/blob/main/Python_scripts/Data_analysis/plot_groupwise_bar.py). For pairwise comparisons we report both a nonparametric Wilcoxon rank-sum test and a two-sample -test (SciPy ranksums and ttest\_ind).

#### Key hyperparameters.

Unless otherwise noted, we used: bodypart = ’Midback’, smooth = True, window = 23 samples, speed\_thresh = 1e-2 normalized-units/s, and time\_limit = None (full trial). Per-trial frame rate was read from the database and validated to be .

All computations were performed in Python (NumPy, SciPy, pandas, Matplotlib) in JupyterLab; code paths and modules are version-controlled within the project directories listed above.