Lightning Pose: improved animal pose estimation via semi-supervised learning, Bayesian ensembling, and cloud-native open-source tools

Dan Biderman[1,†], Matthew R Whiteway[1,†],
Cole Hurwitz[1], Nicholas Greenspan[1], Robert S Lee[2], Ankit Vishnubhotla[1],
Richard Warren[1], Federico Pedraja[1], Dillon Noone[1], Michael Schartner[3], Julia M Huntenburg[4],
Anup Khanal[5], Guido T Meijer[3], Jean-Paul Noel[6],
Alejandro Pan-Vazquez[7], Karolina Z Socha[8], Anne E Urai[9],
The International Brain Laboratory, John P Cunningham[1], Nathaniel Sawtell[1],
Liam Paninski[1]

[†]These authors contributed equally to this work.
[1]Columbia University, New York, USA, [2]Work done while at Lightning.ai, New York, USA,
[3]Champalimaud Centre for the Unknown, Lisbon, Portugal,
[4]Max Planck Institute for Biological Cybernetics, Tübingen, Germany,
[5]University of California Los Angeles, Los Angeles, USA, [6]New York University, New York, USA,
[7]Princeton University, Princeton, USA, [8]University College London, London, United Kingdom,
[9]Leiden University, Leiden, The Netherlands
[*]Correspondence: db3236@cumc.columbia.edu, m.whiteway@columbia.edu

Abstract

Pose estimation algorithms are shedding new light on animal behavior and intelligence. Most existing models are only trained with labeled frames (supervised learning). Although effective in many cases, the fully supervised approach requires extensive image labeling, struggles to generalize to new videos, and produces noisy outputs that hinder downstream analyses. We address each of these limitations with a semi-supervised approach that leverages the spatiotemporal statistics of unlabeled videos in two different ways. First, we introduce unsupervised training objectives that penalize the network whenever its predictions violate smoothness of physical motion, multiple-view geometry, or depart from a low-dimensional subspace of plausible body configurations. Second, we design a new network architecture that predicts pose for a given frame using temporal context from surrounding unlabeled frames. These context frames help resolve brief occlusions or ambiguities between nearby and similar-looking body parts. The resulting pose estimation networks achieve better performance with fewer labels, generalize better to unseen videos, and provide smoother and more reliable pose trajectories for downstream analysis; for example, these improved pose trajectories exhibit stronger correlations with neural activity. We also propose a Bayesian post-processing approach based on deep ensembling and Kalman smoothing that further improves tracking accuracy and robustness. We release a deep learning package that adheres to industry best practices, supporting easy model development and accelerated training and prediction. Our package is accompanied by a cloud application that allows users to annotate data, train networks, and predict new videos at scale, directly from the browser.

1 Introduction

Behavior is our window into the processes that underlie animal intelligence, ranging from early sensory processing to complex social interaction (Krakauer et al., 2017; Niv, 2021). Methods for automatically quantifying behavior from video (Branson et al., 2009; Berman et al., 2014; Wiltschko et al., 2015) have opened the door to high-throughput experiments that compare animal behavior across pharmacological (Wiltschko et al., 2020) and disease (Luxem et al., 2022) conditions. Moreover, when behavior is carefully monitored, motor signals are revealed in unexpected brain areas, even regions classically defined to be purely sensory (Musall et al., 2019; Stringer et al., 2019; Bimbard et al., 2023).

Pose estimation methods based on fully-supervised deep learning have emerged as a workhorse for behavioral quantification (Mathis et al., 2018; Graving et al., 2019; Pereira et al., 2019; Chen et al., 2020; Sauerbrei et al., 2020; Dunn et al., 2021; Segalin et al., 2021; Schneider et al., 2022). This technology reduces high-dimensional videos of behaving animals to low-dimensional time series of their poses, defined in terms of a small number of user-selected keypoints per video frame. Three steps are required to accomplish this feat. Users first create a training dataset by manually labeling poses on a subset of video frames; typically hundreds or thousands of frames are labeled to obtain reliable pose estimates. A neural network is then trained to predict poses that match user labels. Finally, the network is run on a new video to predict a pose for each frame separately. This process of labeling-training-prediction can be iterated until performance is satisfactory. The resulting pose estimates are used extensively in downstream analyses including quantifying predefined behavioral features (e.g., gait features such as stride length, or social features such as distance between subjects), estimation of neural encoding and decoding models, classification of behaviors into discrete “syllables,” and closed-loop experiments (Jones et al., 2020; Nilsson et al., 2020; Saxena et al., 2020; Hsu et al., 2021; Warren et al., 2021; Akiti et al., 2022; Padilla-Coreano et al., 2022; Pereira et al., 2022; Weinreb et al., 2023).

Although the supervised paradigm is effective in many cases, a number of critical roadblocks remain. To start, the labeling process can be laborious, especially when labeling complicated skeletons on multiple views. Even with large labeled datasets, trained networks are often unreliable: they output “glitchy” predictions that require further manipulation before downstream analyses (Karashchuk et al., 2021; Monsees et al., 2022), and struggle to generalize to animals and sessions that were not represented in their labeled training set (Gonschorek et al., 2021). Even well-trained networks that achieve low pixel error on a small number of labeled test frames can still produce a sufficient fraction of error frames that hinder downstream scientific tasks. Manually identifying these error frames is like finding a needle in a haystack (Rodgers, 2022): errors persist for a few frames at a time whereas behavioral videos can be hours long. Automatic approaches – currently limited to filtering low-confidence predictions and temporal discontinuities – can easily miss scientifically critical errors.

To improve the robustness and usability of animal pose estimation, we present *Lightning Pose*, a solution at three levels: modeling, software, and a cloud-based application.

First, we leverage semi-supervised learning, which involves training networks on both labeled frames and on unlabeled videos. Semi-supervised learning has been shown to improve network generalization and data-efficiency (Chapelle et al., 2009), and has played an important role in the recent success of deep learning, especially Large Language Models (Devlin et al., 2018; Brown et al., 2020). On unlabeled videos, the networks are trained to minimize a number of unsupervised losses that encode our prior beliefs about moving bodies: poses should evolve smoothly in time, be physically plausible, and be localized consistently when seen from multiple views. In addition, we leverage unlabeled frames in a novel *Temporal Context Network* architecture, which instead of taking in a single frame at a time, processes each frame with its neighboring (unlabeled) video frames. Our resulting models outperform their purely supervised counterparts across a range of metrics and datasets, providing more reliable predictions for downstream analysis including neural decoding analyses. We further improve our networks' predictions using a general Bayesian post-processing approach, which we coin the *Ensemble Kalman Smoother*: we aggregate ("ensemble") the predictions of multiple networks – which is known to improve their accuracy and robustness (Lakshminarayanan et al., 2017; Abe et al., 2022b) — and model those aggregated predictions with a spatially-constrained Kalman smoother that takes their collective uncertainty into account.

Importantly, we developed a deep learning software developer kit that capitalizes on recent developments in the deep learning ecosystem. New open-source technologies allow us to outsource engineering-heavy tasks (such as GUI development, or training orchestration), which simplifies our package and lets the users focus on scientific modeling decisions. We name our package *Lightning Pose*, as it is based on the PyTorch Lightning deep learning library (Falcon et al., 2020). Unlike most existing packages, Lightning Pose is video-centric and built for manipulating large videos directly on the GPU, to support our semi-supervised training (and enable fast evaluation on new videos). Our modular design allows users to quickly prototype new training objectives and network architectures without affecting any aspects of training. We support parallel hyperparameter searches, multi-GPU training, and the best practices for stably and efficiently training networks to convergence.

Finally, to make pose estimation tools accessible to the broader audience in life sciences, their adoption should not depend on programming skills or access to specialized hardware. Therefore, we developed a no-install cloud application that runs on the browser and allows users to perform the entire cycle of pose estimation: uploading raw videos to the cloud (drag-and-drop), annotating frames, training multiple networks in parallel, and diagnosing the reliability of the results using our unsupervised loss terms. Using our application, trained models can be deployed on parallel machines to predict behavioral videos in a massively accelerated manner. Our open-source code is available at `https://github.com/danbider/lightning-pose` and at the Python Package Index (PyPI), via `pip install lightning-pose`.

2 Results

We first describe the dominant supervised approach to pose estimation and illustrate its drawbacks, especially when applied to out-of-distribution data. Next, we introduce our novel unsupervised losses and Temporal Context Network architecture. We illustrate that these unsupervised losses can be used to identify outlier predictions in unlabeled videos, and find that context-aware networks trained with these losses lead to more reliable tracking compared to purely supervised models. We then introduce the Ensemble Kalman Smoother post-processing approach and show that it further improves tracking accuracy and robustness. We proceed to apply our models to large-scale datasets from the International Brain Lab, and show that the new methods significantly improve tracking of pupil diameter and paw location, thereby improving neural decoding. Finally, we showcase our software developer kit and cloud-hosted application, which supports rapid model prototyping and easy access to cloud resources. Further details on our models, losses, and training protocol are provided in Sec. 5.

2.1 Supervised pose estimation and its limitations

2.1.1 The standard pose estimation model

The leading packages for animal pose estimation – DeepLabCut (Mathis et al., 2018), SLEAP (Pereira et al., 2019), DeepPoseKit (Graving et al., 2019), and others – differ in architectures and implementation but all perform supervised heatmap regression on a frame-by-frame basis (Fig. 1A). A standard model is composed of a "backbone" that extracts features for each frame (typically, a ResNet-50 network) and a "head" that uses these features to predict body part location heatmaps. Networks are trained to match their outputs to manual labels.

Training supervised networks from a random initialization requires a large amount of labeled frames. Existing methods (Mathis et al., 2018) circumvent this requirement by relying on *transfer learning*: *pre-training* a network on one task (e.g., image classification on ImageNet, with over one million labeled frames), and *fine-tuning* it on another task, in this case pose estimation, using far fewer labeled frames ($\sim$100s-1000s). Typically, the backbone is fine-tuned and the head is trained from random initialization.

After training, a fixed model is evaluated on a new video by predicting pose on each frame separately. Each predicted keypoint on each frame is accompanied by an estimate of the network's confidence for that prediction; often low-confidence estimates are dropped in a post-processing step to reduce tracking errors.

2.1.2 Supervised pose estimation often yields unreliable predictions

Despite the use of many labeled training frames and simple post-processing techniques, pose estimation outputs may still be erroneous. To highlight this point, we trained a DeepLabCut model to convergence using 631 labeled frames from the "mirror-mouse" dataset, which features a head-fixed mouse running on a wheel and performing a sensory-guided locomotion task (Warren et al., 2021; for more details, see 5.1). Using a camera and a bottom mirror, the mouse's side is seen from the top view and its underside from the bottom view, recorded at 250 frames per second. 17 body parts are tracked, including all four paws in both views. Figure 1B shows the time series of the left-front paw position during one second of a running behavior. The time series exhibits the expected periodic pattern (due to the running gait), but includes numerous "glitches," some of which are undetected by the network's confidence. The corresponding video, linked in Fig. V1, shows that the network confuses between the left and right top front paws.

This example is in line with the recent observation that network predictions need to be extensively post-processed to better match ground-truth kinematics (Karashchuk et al., 2021; Monsees et al., 2022). Simpler post-processing approaches drop low-confidence frames and interpolate over them with a polynomial (The International Brain Laboratory, 2023) or an autoregressive integrated moving average (ARIMA) model (Nath et al., 2019). More complex post-processing schemes rely on specific body models and require expensive optimization or sampling techniques (Biderman et al., 2020; Joska et al., 2021; Zhang et al., 2021; Monsees et al., 2022), limiting their general applicability. We discuss these further in 5.9.7.

2.1.3 To generalize to unseen subjects, supervised networks require more labeled examples

It is standard to train a pose estimator using a representative sample of subjects, evaluate performance on held-out examples from that sample ("In Distribution" test set, henceforth InD), and then deploy the network

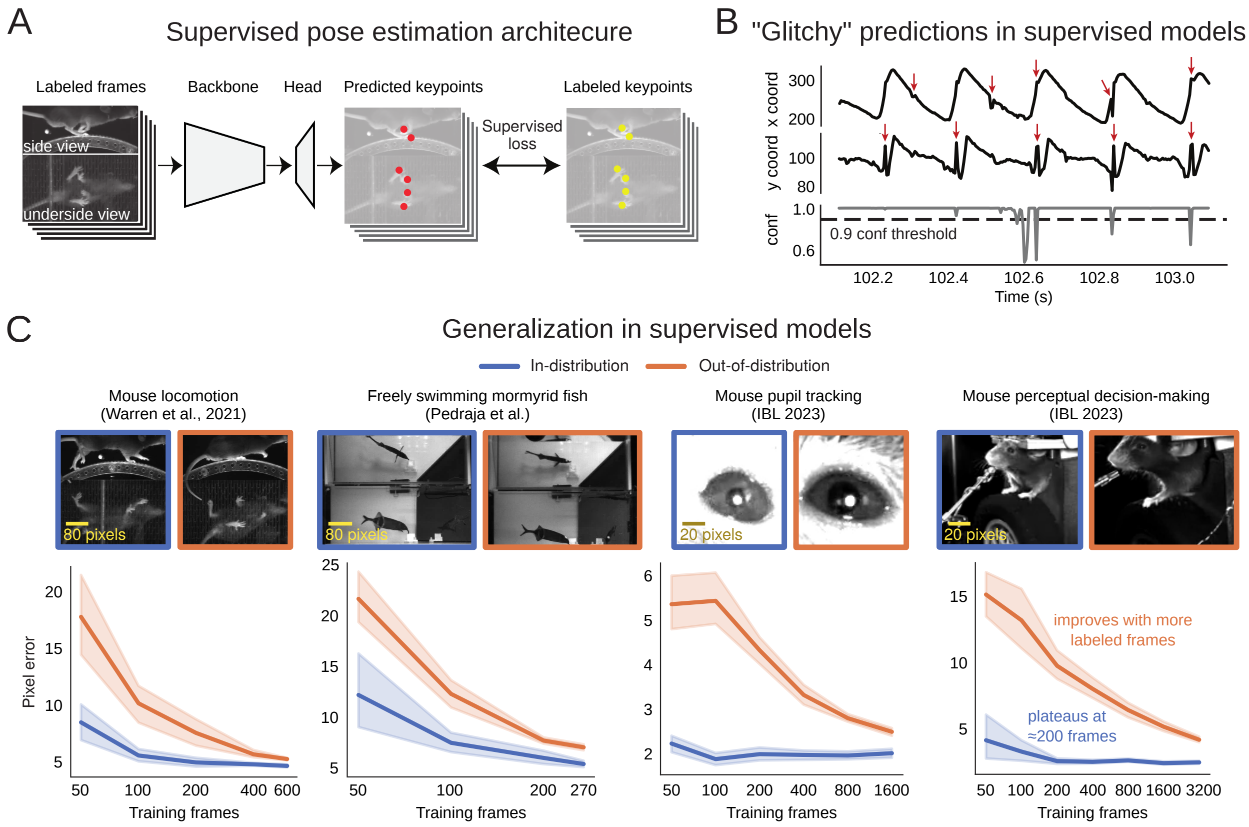


Figure 1: **Fully-supervised pose estimation often outputs “glitchy” predictions and requires many labels to generalize to new animals.** **A**. Diagram of a typical pose estimation model trained with supervised learning, illustrated using the mirror-mouse dataset (Warren et al., 2021). A dataset is created by labeling keypoints on a subset of video frames. A convolutional neural network, consisting of a "backbone" and a prediction "head", takes in a batch of frames as inputs, and predicts a set of keypoints for each frame. It is trained to minimize the distance from the labeled keypoints. **B**. Predictions from a supervised DeepLabCut network (trained with 631 labeled frames on the mirror-mouse dataset), for the left front paw position (top view) during one second of running behavior. *Top*: x-coordinate; *Middle*: y-coordinate; *Bottom*: confidence, applying a standard 0.9 threshold in a dashed line. The predictions demonstrate occasional discontinuities, only some of which are flagged by low confidence. The corresponding video is linked in Fig. V1. **C**. To generalize robustly to unseen animals, many more labels are required. Top row shows four example datasets, left to right: mirror-mouse, mirror-fish, IBL-pupil, and IBL-paw (note that the IBL-pupil frames were cropped from the IBL-paw dataset, and re-scaled to 100 × 100 pixels.) Each blue image is an example taken from the in-distribution (InD) test-set, that contains new images of animals that were seen in the training set. The orange images are test examples from unseen animals altogether (out-of-distribution (OOD) test set). Bottom row shows data-efficiency curves, measuring test-set pixel error as a function of the training set size. InD pixel error is in blue and OOD in orange; standard error is taken over 10 random subsets of InD training data. InD pixel error plateaus after ∼200 labeled frames whereas OOD performance continues to improve as many more labeled frames are added.

for incoming data. The incoming data may include new subjects, seen from different angles and lighting conditions (“Out of Distribution” test set, henceforth OOD). Differences between the InD and OOD test sets are termed “OOD shifts”; building models that are *robust* to such shifts is a contemporary frontier in machine learning research (Torralba et al., 2011; Recht et al., 2019; Miller et al., 2021; Tran et al., 2022).

We analyze four datasets: mirror-mouse (Warren et al., 2021), mirror-fish, IBL-paw (The International Brain Laboratory, 2023), and IBL-pupil (generated by cropping the pupil area in IBL-paw). We split each labeled dataset into two cohorts of subjects, InD and OOD (see dataset and split details in Sec. 5.1). We train supervised DeepLabCut-like networks (using a ResNet-50 backbone pretrained on ImageNet) on InD data with an increasing number of labeled frames. Ten networks are trained per condition, each on a different random subset of InD data. We evaluate the networks' performance on held-out InD and OOD labeled examples.

In Fig. 1C, we first replicate the observation that InD test-set error (blue curve) plateaus starting from ∼200 labeled frames (Pereira et al., 2022). From looking at this curve in isolation, it could be inferred that additional manual annotation is not necessary. However, the OOD error curve (red) is both overall higher, and keeps steeply declining as more labels are added. To obtain an OOD error comparable to InD, many more labels will be needed. The need for more labels resonates both with recent work showing that ∼50k labeled frames are needed to robustly track ape poses (Desai et al., 2022), and that mouse face tracking networks need to be explicitly fine-tuned on labeled OOD data to achieve agreeable performance (Syeda et al., 2022). For scarce labels, we find the gap between InD and OOD errors to be so large that it renders prediction on new animals unusable for downstream analyses.

2.2 Lightning Pose: a semi-supervised, context-aware approach

To address the limitations noted above, we propose a new framework, called "Lightning Pose," comprising two components: semi-supervised learning using spatiotemporal constraints on unlabeled video data, and a Temporal Context Network architecture. We present these components in order.

2.2.1 Semi-supervised learning via spatiotemporal constraints on unlabeled videos

Most animal pose estimation algorithms treat body parts as independent in time and space, and rely on the capacity of deep neural networks to separately localize each body part. Moreover, they do not utilize the vast amounts of available unlabeled videos for training the networks; instead, most video data are used just at prediction time.

These two observations offer an opportunity for semi-supervised learning (Chapelle et al., 2009). We train networks on both labeled frames (supervised) and large volumes of unlabeled videos (unsupervised). During training, the network is penalized whenever its pose predictions violate a set of spatiotemporal constraints on the unlabeled videos; these penalties are then back-propagated and used to modify the network's weights. The unsupervised penalties are applied only during training and not during video prediction. As a result, after training, our semi-supervised models predict videos as fast as their fully-supervised counterparts.

Our semi-supervised pose estimation paradigm is depicted in Fig. 2A. The top row, shaded in gray, is simply the supervised pose estimation approach à la DeepLabCut. In each training iteration, a Lightning Pose network additionally receives an unlabeled video clip (selected at random from a queue of videos), and outputs a time-series of pose predictions - one pose vector for each frame (bottom row). Those predictions are subjected to our unsupervised losses, which encode spatiotemporal constraints on body dynamics as seen from one or more cameras. We describe these unsupervised losses next.

2.2.1.1 Temporal Continuity Loss

The first spatiotemporal constraint we introduce is also one held by 4-month-old infants: objects should move continuously (Spelke, 1990) and not jump too far between video frames. We define the temporal loss for each body part as the Euclidean distance between consecutive predictions (in pixels). Similar temporal losses have been used by several practitioners to detect outlier predictions post hoc (Warren et al., 2021; Syeda et al., 2022), whereas our goal here, following Wu et al., 2020, is to incorporate these penalties directly into network training to achieve more accurate network output. Figure 2B illustrates this penalty: the cartoon in the left panel indicates a jump discontinuity (red dashed line) we would like to penalize. In the right panel we plot the loss landscape, evaluating the loss for every pixel in the image, with the white diamond indicating the paw's previous predicted position, and the loss increasing as we move farther away from this position.

If our losses are indeed viable proxies for pose prediction errors, then they should be correlated with pixel errors in test frames for which we have ground-truth annotations. To test this we trained a supervised model with 75 labeled frames, and computed the temporal norm on labeled OOD frames. Specifically, for temporal jumps, we anticipate a mild correlation with pixel error: prediction errors may persist across multiple frames and exhibit low temporal norms; in periods of fast motion, temporal norms may be high, yet keypoints may remain easily discernible. Indeed, in the bottom left panel of Fig. 2B, we see that the temporal loss is significantly, yet mildly, correlated with pixel error on these frames (Pearson $r = 0.25,\ 95\%$ CI $= [0.19, 0.30]$; each point is the mean across all keypoints for a given frame). As a comparison, lower confidence is a more reliable predictor of pixel error (Pearson $r = -0.47,\ 95\%$ CI $= [-0.52, -0.43]$).

2.2.1.2 Multi-view Consistency Loss

Our cameras see three-dimensional bodies from a two-dimensional perspective. It is increasingly common to record behavior using multiple synchronized cameras, train a network to estimate pose independently in each 2D view, and then use standard stereo vision techniques post hoc to fuse those 2D pose predictions into a 3D pose (Mathis et al., 2020; Karashchuk et al., 2021; Zhang et al., 2021; Ebrahimi et al., 2023). This approach has two limitations. First, to reconstruct 3D poses, one needs to calibrate each camera, that is, to precisely infer where it is in the 3D world and carefully model its intrinsic parameters such as focal length and distortion. This typically involves filming a so-called calibration board from all cameras after any camera adjustment; this adds experimental complexity and may be challenging or unreliable in some geometrically constrained experimental setups built for small model organisms, or in small aquariums. Second, and more importantly, localizing a body part in one view will constrain its allowed location in all other views (Hartley et al., 2003), and we want to exploit this structure during training to obtain a stronger network.

Our multi-view consistency loss constrains the predictions for unlabeled videos to be consistent across views (Jafarian et al., 2018; He et al., 2020b; Zhang et al., 2020; Dunn et al., 2021), while bypassing the need for complicated camera calibration. Each multi-view prediction (containing (x, y) coordinates for a single body part seen from multiple views) is compressed to three dimensions (via simple principal components analysis; see Sec. 5.4.2 for full details), and then this three-dimensional representation is reprojected linearly back into the original pixel coordinates. If the predictions are consistent across views, no information should be lost when compressing them to three dimensions. Indeed, three dimensions explain $> 99.9\%$ of the multi-view ground-truth label variance for both the mirror-fish (three views) and mirror-mouse (two views) datasets (Fig. 2C, bottom right). We define the multi-view consistency loss as the pixel error between the original versus the reprojected prediction.

Figure 2C (top left) provides a cartoon illustration of the idea we have just described; here we visualize an inconsistency between the two views for which the multi-view loss will be high, and in the top right panel we compute the loss landscape for the left front paw on the top view, given its position in the bottom view. We see that the loss is minimized when the two views of the same paw are consistent with a single physical three-dimensional location. Finally, as we did for the temporal loss, we compute the correlation between the multi-view loss and objective prediction errors for a test-set of labeled OOD frames. The multi-view loss is strongly correlated with pixel error (Pearson $r = 0.83,\ 95\%$ CI $= [0.81, 0.84]$), much more so than the temporal loss or confidence, motivating its use both post-hoc as a quality control metric and as a penalty during training.

2.2.1.3 Pose plausibility loss

Not all body configurations are feasible, and of those that are feasible, many are unlikely. Even diligent yoga practitioners will find their head next to their foot only on rare occasions (Fig. 2D, top left). In other words, in many pose estimation problems there are fewer degrees of freedom than there are body parts. The Pose PCA loss constrains the full predicted pose (over all keypoints) to lie on a low-dimensional subspace of feasible and likely body configurations. It is defined as the pixel error between an original pose prediction and its reprojection after low-dimensional compression (see Sec. 5.4.3 for full details).

Our loss is inspired by the success of low-dimensional models in capturing biological movement (Bialek, 2022), ranging from worm locomotion (Stephens et al., 2010) to human hand grasping (Yan et al., 2020). We similarly find that across three of our datasets, 99% of the pose variance could be explained with far fewer dimensions than the number of pose coordinates (Fig. 2D, bottom right) – mirror-mouse: 14/34 components; mirror-fish: 6/45; ibl-pupil 3/8. As a specific example, Fig. 2D (top right) shows the Pose PCA loss landscape for the left hind paw location (true location shown as a white diamond) given the location of all the other body parts. As desired, the Pose PCA loss is lower around the paw's true location and accommodates plausible neighbouring locations. Here too, we find that Pose PCA loss closely tracks ground-truth pixel error on a subset of labeled OOD frames (Fig. 2D, bottom left; Pearson $r = 0.82,\ 95\%$ CI $= [0.80, 0.84]$)[1], making it another viable proxy for error in the absence of labels.

Finally, we note that since the subspace of plausible poses is estimated from the (potentially very small) labeled dataset, there might be valid postures that are not represented in that dataset; these will be erroneously deemed as implausible. In practice, however, we have found that with as little as 75 labeled frames, the subspace well captures plausible poses and excludes implausible ones.

[1] The correlation coefficient with pixel error is roughly identical between the Pose and Multi-view PCA losses for the mirror-mouse dataset, in which the Pose PCA includes both views.

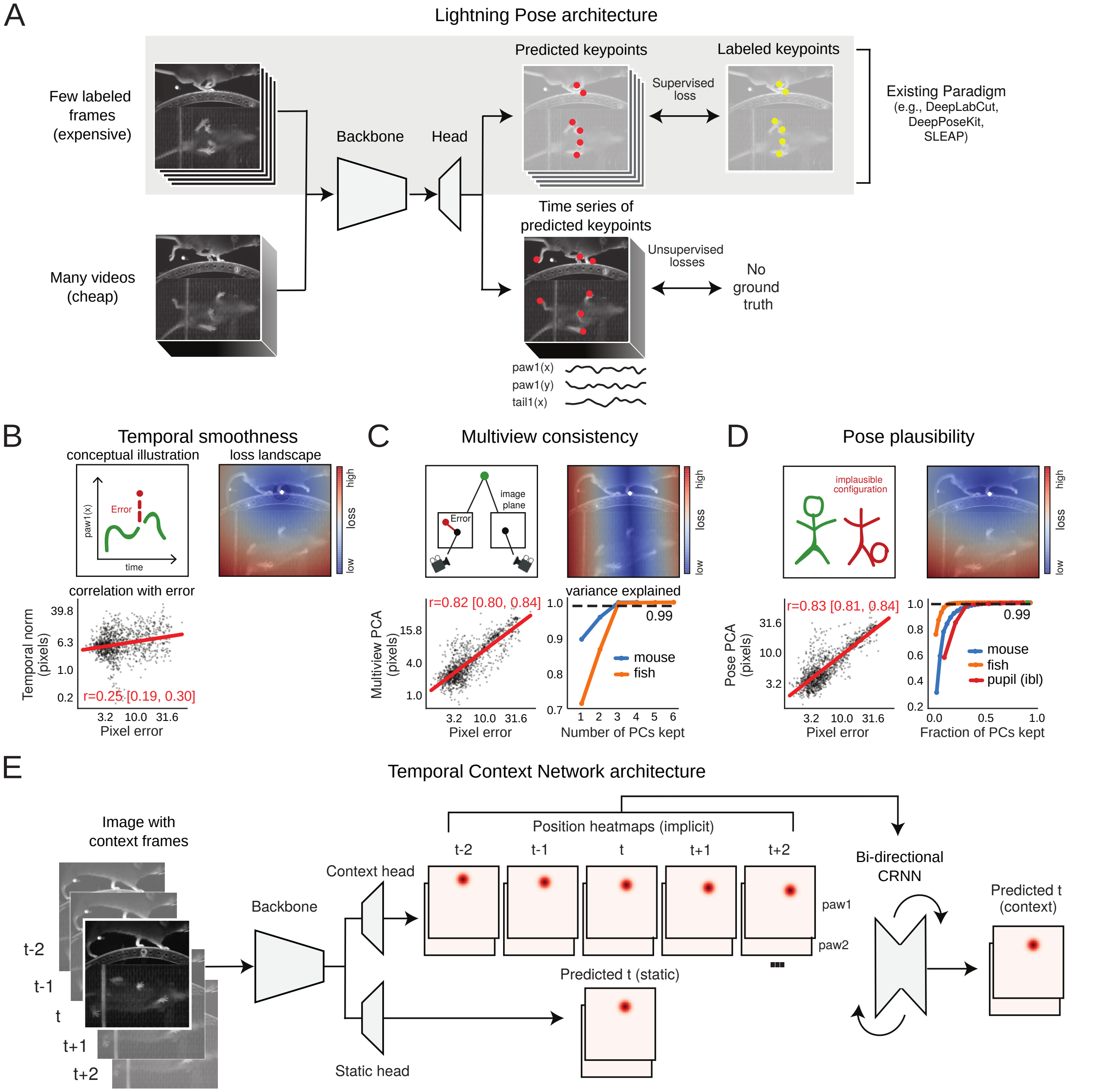


Figure 2: **Exploiting unlabeled videos in pose estimation model training.** Caption on following page.

Figure 2: **Exploiting unlabeled videos in pose estimation model training.** **A**. Diagram of our proposed semi-supervised pose estimation model. Top row illustrates the supervised component that is akin to existing approaches. The bottom row indicates the unsupervised component. The network is given a short unlabeled clip and forms a pose prediction for each frame. These predictions are subjected to our unsupervised losses. **B**. Temporal continuity loss penalizes the network for jump discontinuities in its predictions. Top left: illustration of a jump discontinuity. Top right: the temporal loss landscape for frame t given the prediction at $t-1$ (white diamond), for the left front paw on the top view. Red values indicate high loss; dark blue indicates a zero loss. The loss increases as we move farther away from the previous prediction, and the dark blue circle around it corresponds to the maximum allowed jump ($\epsilon = 20$ pixels for this dataset) below which the loss is set to zero. Bottom left: correlation between temporal norm and pixel error on a subset of labeled test frames. **C**. The multi-view consistency loss constrains each multi-view prediction of the same body part to lie on a three-dimensional subspace found by Principal Component Analysis (PCA). Top left: illustration of a 3D keypoint detected on the imaging plane of two cameras. The left detection is wrong and inconsistent with the right one. Top right: loss landscape for the left front paw (top view; white diamond) given its predicted location on the bottom view. Observe a blue band of low loss values (along the "epipolar line") on which the top-view paw could be located. Bottom left: multi-view consistency loss is strongly correlated with pixel error on a subset of labeled test frames. Bottom right: three principal components explain $> 99\%$ of the multi-view labels, on both the two-view mirror-mouse (blue) and three-view mirror-fish (orange) datasets. **D**. Pose plausibility loss constrains pose predictions to lie on a low-dimensional subspace of plausible poses, found by PCA. Top left: illustration of a plausible and implausible poses. Top right: loss landscape for the left front paw (top view; white diamond) given all other keypoints, which is minimized around the paw's actual position. Bottom left: strong correlation between pose plausibility loss and pixel error on a labeled test set. Bottom right: across four labeled datasets, $> 99\%$ of the variance in the pose vectors can be explained by retaining $< 50\%$ of the PCs. **E**. The Temporal Context Network (TCN) processes each frame with its adjacent video frames, using a bi-directional convolutional recurrent neural network (CRNN). It simultaneously forms two sets of location heatmap predictions, one using single-frame information and the second using temporal context.

2.2.2 Temporal Context Network

Some frames are more challenging to label than others, due to occlusions or ambiguities between similar body parts. In many cases, additional *temporal context* can help resolve ambiguities: e.g., if a keypoint is occluded briefly we can scroll backwards and forwards in the video to help “fill in the gaps.” However, this useful temporal context is not provided to standard frame-by-frame pose estimation architectures, which instead must make guesses about such challenging keypoint locations given information from just one frame at a time.

Therefore, we propose a Temporal Context network (TCN), illustrated in Fig. 2E, which uses a $2K+1$ frame sequence to predict the location heatmaps for the middle (i.e., $K+1$) frame; in the depicted example, $K = 2$, i.e., five frames are used to predict heatmaps for a single middle frame. As in the standard architecture, the TCN starts by pushing each image through a large neural network backbone that computes useful features from each frame. Then, instead of predicting the pose directly from each of these individual per-frame feature vectors, we combine this information across frames using a bi-directional convolutional recurrent neural network (CRNN), and then use the output of the CRNN to form predictions for the middle frame. The CRNN is lightweight compared to the neural network backbone, and we only apply the backbone once per frame; therefore the TCN is only slightly slower than the original frame-by-frame architecture.

2.3 Spatiotemporal losses identify outlier predictions on unlabeled video data

We have shown above that larger violations of our spatiotemporal constraints are associated with larger pixel errors in labeled test sets. However, focusing on small labeled test sets may yield an incomplete summary of network performance (Rodgers, 2022; Li et al., 2023); typical label test sets include tens or hundreds of frames, whereas trained networks predict poses for up to millions of unlabeled video frames across many videos and may considerably vary in their quality. Here we show that violations of our spatiotemporal losses indeed correspond to “error frames” in unlabeled videos. As a result, our losses can additionally serve as anomaly detectors for determining which predictions should be trusted downstream, in tasks such as behavioral segmentation (Nilsson et al., 2020; Hsu et al., 2021; Segalin et al., 2021; Weinreb et al., 2023) or pose representation learning (Whiteway et al., 2021). Moreover, they can be used as quality control metrics for comparing the output of various pose estimation networks.

We establish this claim using the mirror-mouse dataset, focusing on the four paws which are seen from the top and bottom views. We chose this dataset first because it is amenable to all unsupervised losses. Second, its unique mirror setup allows us to define “true outliers,” i.e., nonsensical predictions that violate the experiment’s geometric setup.

In Fig. 3A,B we present example video predictions from a DeepLabCut model (trained with 631 training frames), for the left hind paw on the bottom view. Temporal discontinuities severely contaminate the expected periodic signal associated with running on a wheel. Fig. 3A shows that the x coordinate’s discontinuity in frames 290-294, for example, is a result of the network confusing the left hind paw with the similar-looking left front paw, going back and forth between the two. These errors are missed by network confidence – the most commonly-employed outlier detector – which is largely above the 0.9 threshold across the entire time-series (Fig. 3B), despite major motion discontinuities. The temporal jump between successive frames, seemingly the second most popular outlier detector (Warren et al., 2021; Syeda et al., 2022), signals when the network jumps to and from a wrong location (see temporal norm trace in Fig. 3B). Yet, this metric will have a small value when the network’s prediction lingers at that wrong location (Fig. 3A, frame

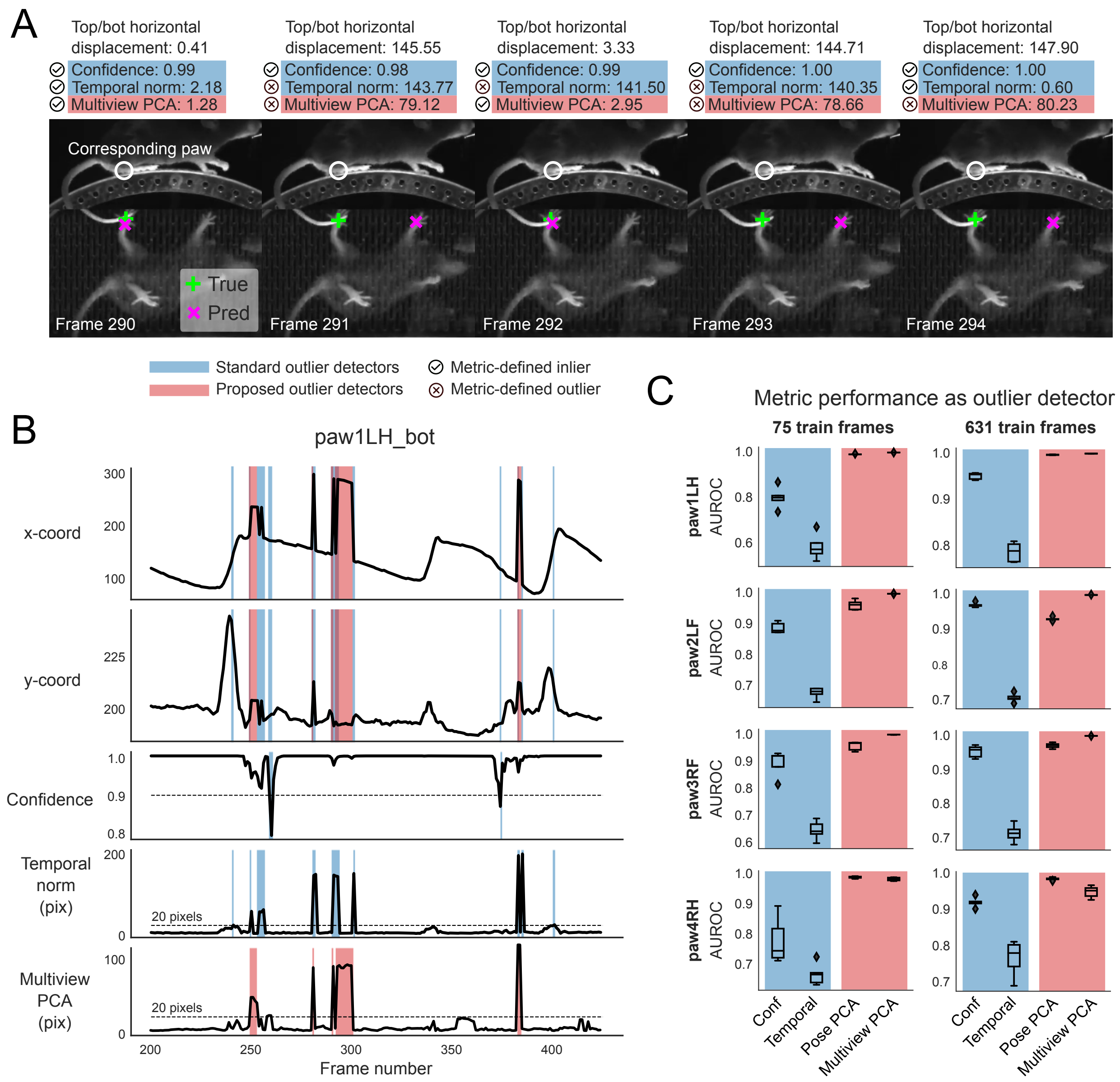


Figure 3: **Unsupervised losses identify outlier predictions, complementing network confidence.** **A**. Example frame sequence from a held-out video from the mirror-mouse dataset. Predictions from a DeepLabCut model (trained on 631 frames) are overlaid (magenta ×) along with the ground truth (green +). The open white circle in each frame denotes the location of the same body part (left hind paw) in the other (top) view; given the geometry of this particular setup, a large horizontal displacement between the top and bottom predictions indicates an error. Each frame is accompanied with "standard outlier detectors", including confidence, temporal norm (shaded in blue), and 'proposed outlier detectors", including multi-view PCA metric (shaded in red; not including Pose PCA for simplicity). ✓ indicates an inlier, and $\otimes$ indicates an outlier. Note that confidence is high for all frames shown, and that the temporal norm misses error frame 294 which does not contain an immediate jump, and flags frame 292 which demonstrates a jump to the correct location. Multi-view PCA captures these correctly. **B**. Example DeepLabCut traces from the same held-out video. Blue background denotes time points where standard outlier detection methods flag frames: confidence falls below a threshold (0.9) and/or the temporal norm exceeds a threshold (20 pixels). Red background indicates time points where the multi-view PCA error exceeds a threshold (20 pixels; the Pose PCA trace is highly correlated). Purple background indicates both conditions are met. **C**. For each of the four paws (in rows) we define a "true outlier" to be frames where the horizontal displacement between the top and bottom predictions exceeds 20 pixels. We then assess how well different metrics (on the x axis) capture outliers by computing the area under the Receiver Operating Characteristic curve (AUROC; on the y axis). The PCA losses are able to properly flag outliers while minimizing false positives better than confidence or the temporal norm. AUROC values are computed across all frames from 20 test videos; boxplot variability is over 5 random subsets of training data.

294). In Fig. 3B we use blue shading to mark moments in which confidence < 0.9 or the temporal jump > 20 pixels. In contrast, the multi-view PCA loss flags this chain of confident but incorrect predictions as inconsistent across views (see red shading when it exceeds a 20 pixel threshold in Fig. 3B), by utilizing the top-view prediction for this keypoint (white circle in Fig. 3A). We find the unsupervised losses can similarly flag outliers not captured by confidence or temporal jumps in the mirror-fish dataset (Fig. S1).

We proceed to statistically quantify the efficacy of these outlier detection methods on DeepLabCut predictions for 20 unlabeled videos, across five random seeds. We focus on two data regimes: scarce labels (75) and abundant labels (631 for the mirror-mouse dataset). The scarce labels regime mimics exploratory experimental stages, where a user labels < 100 frames to obtain workable pipelines for pose estimation and downstream analysis. The abundant label regime corresponds to a "production" setting, in which a pose estimation network has been thoroughly trained on larger labeled datasets (perhaps after multiple rounds of relabeling and retraining) and is used to predict incoming data. We concentrate on the four paws and conservatively define "true" outliers as frames for which the horizontal displacement between the top and bottom view predictions for a given paw exceeds 20 pixels, similar to Warren et al., 2021. Our definition indeed captures some but not all errors (such as incorrect vertical positions), and therefore we are lower-bounding the error rate. Finding these errors manually would be laborious: networks trained with 631 frames make on average about 800 such errors in $40,000$ video frames (about 3500 errors for the 75 training frames regime).

We then determine how well each unsupervised metric performs as an outlier detector: for example, with confidence, we set an arbitrary threshold, and if the keypoint confidence is below that threshold the bodypart prediction is considered an outlier. We repeat this analysis across a range of thresholds – thus, generalizing the specific threshold choices of Fig. 3B – and use the true and false positive rates to compute the area under the Receiver Operating Characteristic curve (AUROC). A value of 1.0 indicates the method finds all true positives and no false positives; a value of 0.0 indicates no true positives and all false positives. Across all paws and both training regimes the PCA losses outperform confidence and the temporal norm in outlier detection performance (Fig. 3C). Thus, as desired, our proposed constraints help capture additional outliers that would have been missed by standard confidence and temporal jump thresholding.

2.4 Unsupervised losses and temporal context improve tracking accuracy and reliability

Above we established that spatiotemporal constraint violations help identify network prediction errors. Next we quantify whether networks trained to avoid these constraint violations achieve more accurate and reliable tracking performance. We quantify networks' performance both on an out-of-distribution labeled test set as well as on many unlabeled videos. In this section, we compare the networks' raw predictions, without any post-processing, to focally assess the implications of our architecture and training losses.

2.4.1 Lightning Pose provides smoother, more reliable, and more confident video predictions

In Fig. 4A, we examine the mouse's right hind paw position (side view) during two seconds of running. We compare the raw video predictions from our full model (in blue), including all unsupervised losses and a TCN architecture, to the ones generated by a supervised model, both trained on 75 labeled frames. We find that our model's predictions are smoother (top two panels) and more confident (bottom panel), exhibiting a clearer periodic pattern expected for running on a stationary wheel. While some of the supervised model's discontinuities are flagged by low confidence, some reflect a confident confusion between similar body parts, echoing Fig. 3A. One such confident confusion is highlighted in gray shading, and further scrutinized in Fig. 4B, showing that the baseline model (red; fully-supervised, no temporal context) mistakenly switches to the left hind paw for two frames. The full semi-supervised temporal context model avoids paw switching first because each frame is processed with its context frames, and second, because switching would have been heavily penalized by both the temporal loss and Multi-view PCA loss.

2.4.2 An ablation analysis reveals that both unsupervised losses and temporal context improve tracking performance

Next, we perform an ablation study to isolate the contributions stemming from our semi-supervised losses, our TCN architecture, and their combination. We compare our models to DeepLabCut (release 2.3.0) and include a baseline supervised model to quantify the effects of our PyTorch Lightning training protocol and use of pre-trained AnimalPose10K backbone (Yu et al., 2021). For each model type, we trained five networks with different random subsets of InD data. Fig. 4B shows pixel error on 253 OOD test frames comparing models trained with scarce and abundant labels. In the scarce labels regime, we find that our models have lower pixel error compared to DeepLabCut (our best model improving by 40%; DeepLabCut pixel error 14.6±0.4; mean±s.e.m. of the average keypoint pixel error across frames). Most improvements were driven by the unsupervised losses (green bar, 10.0±0.03), and further refined by using the TCN ("semi-super context"; blue bar, 8.8±0.2). For the abundant labels regime we find comparable OOD pixel error across models. We also compute a set of diagnostic metrics on a much larger unlabeled dataset, including 20 OOD videos (Fig. 4D). These metrics reveal major improvements in model reliability which are not captured by pixel errors on the labeled dataset.

2.4.3 Lightning Pose provides more reliable predictions across a wide range of videos

We compute each of our loss terms – temporal, multi-view consistency, and Pose PCA – for every predicted keypoint on every video frame. In the mirror-mouse dataset, any deviations of these losses above > 20 pixels corresponded to a clear error by visual inspection. Thus, we can lower-bound the error rate by counting these > 20 pixel "violator frames." For every keypoint, we calculate the proportion of violator frames in a video. This "violator rate" decreases as we move from the sparse label regime to the abundant label regime and as the corresponding networks become more accurate (Fig. 4D). Importantly, this observation holds not only for the full semi-supervised context model, but also for the DeepLabCut and baseline models which were not trained with our spatiotemporal losses. Therefore, better supervised models exhibit fewer constraint violations without being explicitly optimized to do so. Furthermore, we find that, in both the scarce and abundant label regimes, Lightning Pose (LP) models discard far fewer frames compared to DeepLabCut so its violator rate is about half of DeepLabCut's for this dataset.

In Fig. 4D we pooled all violator frames together. In Fig. 4E-F we examine each of the violator types individually, to better understand how the full semi-supervised context model improves over the baseline fully-supervised model. For both data regimes, the new LP models are more confident, more consistent across views, smoother, and more strictly obey plausible body configurations. As in panel D, we see that the baseline model's losses decrease as the label set size increases from the scarce to the abundant regime (c.f. the leftward shift between panels E and F), again without being explicitly trained to reduce these violators. We find similar results in the mirror-fish dataset (Fig. S2).

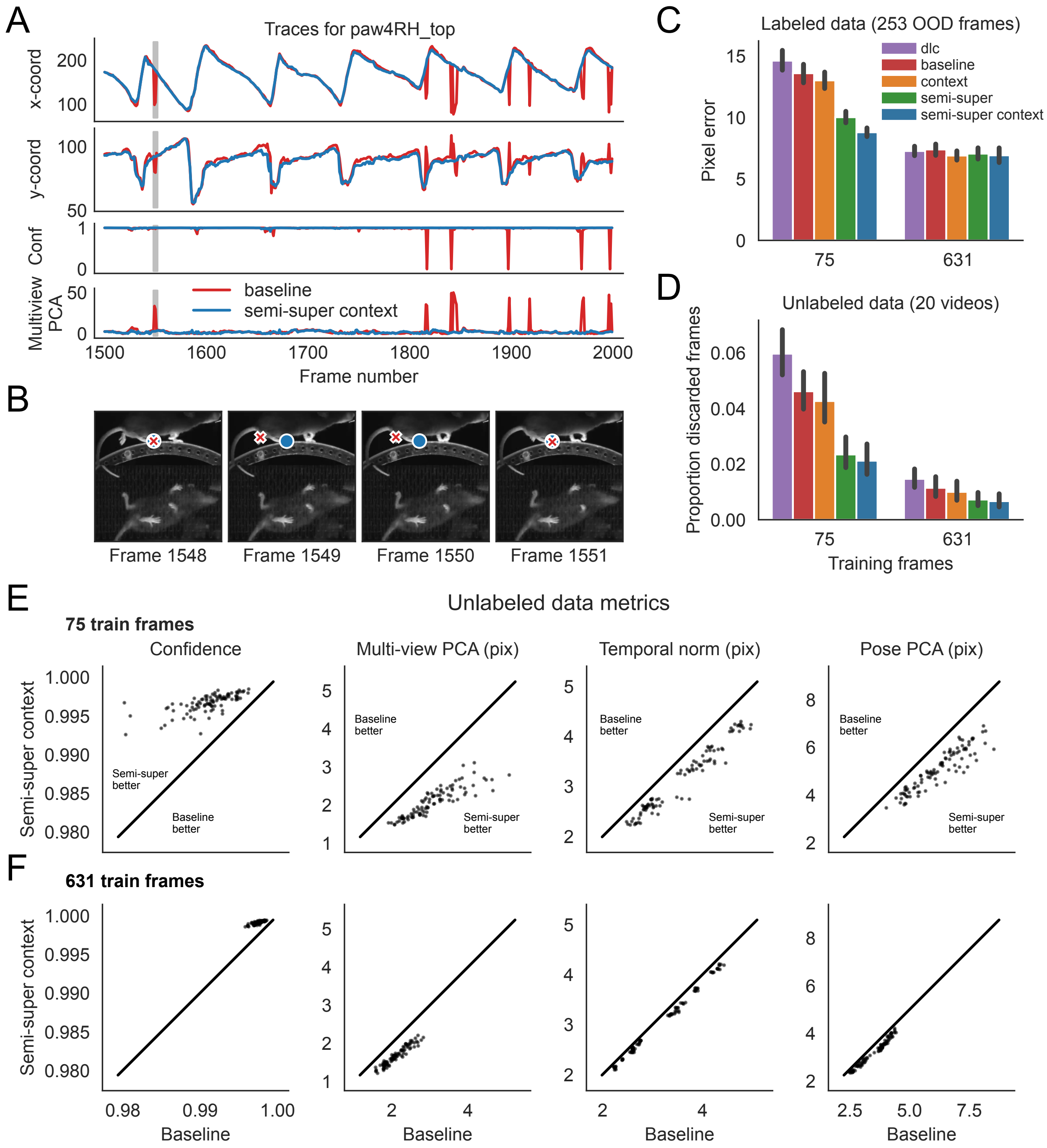


Figure 4: **Unlabeled frames improve pose estimation (raw network predictions.)** **A**. Example traces from the baseline model and the semi-supervised context model (trained with 75 labeled frames) for a single keypoint (right hind paw; top view) on a held-out video. See Fig. V1 for link to video. The semi-supervised context model is able to resolve the visible glitches in the trace, only some of which are flagged by the baseline model's low confidence. One erroneous paw switch missed by confidence – but captured by Multi-view PCA loss – is shaded in gray. **B**. A sequence of frames (1548-1551) corresponding to the gray shaded region in panel A in which a paw switch occurs. The estimates from both models are initially correct, then at Frame 1549 the baseline model prediction jumps to the incorrect paw, and stays there until it jumps back at Frame 1551. **C**. Pixel error over OOD frames for four models in both the scarce and abundant label regime. Error bars are computed over 5 random shuffles of training data. **D**. Proportion of frames in unlabeled video data discarded by each model, averaged across keypoints; keypoint prediction is discarded if any of the unsupervised losses is above a 20 pixels threshold, indicating a severe constraint violation. Error bars are over 5 random shuffles and 20 held-out videos. Our models keep significantly more usable frames per keypoint as compared to DeepLabCut. **E**. The loss terms that together comprise panel D are individually scattered for the scarce label regime, with our model's values on the y-axis, and a baseline model's values on the x-axis. Each dot is an average across a single OOD video. **F**. The performance improvements of the semi-supervised context model hold when using all training frames.

2.5 Ensemble Kalman Smoothing (EKS) substantially improves tracking accuracy and robustness

While our networks are significantly more accurate than their supervised counterparts, they still make occasional mistakes. Our software package (described in Sec. 2.7) conveniently allows us to further improve accuracy and out-of-distribution robustness by *deep ensembling* (Lakshminarayanan et al., 2017; Fort et al., 2019; Ovadia et al., 2019), that is, averaging the predictions of multiple networks, each trained from a different random initialization, using a different subset of data, or a different order of data presentation (Fig. S3A; see Methods for our specific implementation). Another benefit of deep ensembling is that it produces a more diverse set of predictions for difficult examples including occlusions or confusions between two similar keypoints. This *ensemble variance* – which varies for each keypoint on every frame – is a useful signal of uncertainty (Fig. S3A; Osband et al., 2016; Beluch et al., 2018; Abe et al., 2022a) that complements the network's confidence and unsupervised losses discussed above.

Most deep ensemble implementations simply average the ensemble predictions (Lakshminarayanan et al., 2017; Fort et al., 2019; Ovadia et al., 2019), but we found that simple ensemble averages do not lead to clear benefits in our mirror-mouse and mirror-fish datasets (Fig. S3C,E). Hence, we developed an ensemble prediction framework that integrates spatiotemporal prior knowledge with the ensemble means and variances via a probabilistic "state-space" model approach (Fig. S3B): our model has a latent "state" that evolves smoothly in time, and is projected onto the keypoint positions (the ensemble means) via our spatial constraints. The model output represents a Bayesian compromise between the spatiotemporal constraints (prior) and the information provided by the ensemble observations (likelihood). Concretely, if a keypoint's uncertainty is low (i.e., all ensemble members agree) then this observation will be upweighted relative to the spatiotemporal prior and will only be lightly smoothed. Conversely, when a keypoint's uncertainty is high, the spatiotemporal priors and other more-confident keypoints' predictions will be used to "infill" over these uncertain observations. In the simplest case, this probabilistic state-space model corresponds to a Kalman filter-smoother model (Bishop et al., 2006) and so we name the resulting probabilistic smoothing approach the "ensemble Kalman smoother" (EKS); see Sec. 5.9 for full details. Unlike previous approaches (Warren et al., 2021), the EKS requires no manual selection of confidence thresholds or (suboptimal) temporal linear interpolation separately for each dropped keypoint. Finally, note that this post-processing approach is agnostic to the type of networks used to generate the ensemble predictions.

Returning to the mirror-mouse dataset, we find that EKS provides significant improvements on OOD pixel error even in the abundant label regime (Fig. S3C), where we had previously seen little difference between model types (Fig. 4C). In the scarce label regime, EKS achieves smooth and accurate tracking even when the semi-supervised context models make frequent errors due to occlusion and paw confusion (Fig. S3D, Fig. S4; see videos in Fig. V2). These performance improvements arise both through reducing outliers and through cleaning up noise in nearly-correct predictions (Fig. S3D). In the following section we demonstrate how pose estimation improvements due to our unsupervised losses, TCN architecture, and EKS enhance downstream analysis tasks.

2.6 Significantly improved tracking accuracy on large-scale public International Brain Laboratory datasets

Next we turn to an analysis of large-scale public datasets from the International Brain Laboratory (IBL) (The International Brain Laboratory, 2023). The major scientific goal of the IBL is to map neural activity across the mouse brain at single-neuron resolution in the context of a standardized behavioral task. In each experimental session, a mouse was observed performing a simple visually-guided decision-making task; the mouse signaled decisions by manually moving a rotary wheel, and behavior was recorded using three cameras. Accurate pose estimation from the video datasets is critically important for the scientific goals of the IBL, since detailed behavioral quantification (e.g. 3D tracking, behavioral classification, neural decoding) is a crucial component of the analysis and interpretation of brainwide neural recordings.

Despite significant efforts at standardization, the resulting IBL video data are fairly diverse, and we have found empirically that this presents a substantial challenge for existing pose estimation methods. In IBL's preliminary data release we used DeepLabCut followed by ad hoc post-processing to estimate a number of behavioral variables, including the pupil diameter and paw location. As detailed in (The International Brain Laboratory, 2023), this approach fails in a large majority of pupil recordings: the signal-to-noise ratio of the estimated pupil diameter is too low for reliable downstream use. Paw tracking tends to be more accurate, but nonetheless in some sessions many tracking "glitches" lead to unreliable downstream analyses. These issues persisted despite the collection of a large labeled dataset (many thousands of labeled frames; see Table 1), collected over multiple rounds of active learning and retraining.

In an effort to improve pose estimation performance, we trained Lightning Pose models on these datasets and then applied the EKS. Figures 5 and S5 summarize our results for the IBL pupil dataset. We evaluate three pose estimators: the DeepLabCut baseline (DLC; left column of example session in Fig. 5), the Lightning Pose semi-supervised context model (LP; middle column), and the EKS applied to the LP models (LP+EKS; right column). We define several quality metrics in this setting to quantify the accuracy of the different models[2]. One such quality metric can be computed by comparing the "vertical" pupil diameter (the difference of the vertical position of the top and bottom pupil keypoints) vs the "horizontal" diameter (the difference of the horizontal positions of the left and right keypoints). The horizontal and vertical diameters should be equal (or at least highly correlated), and, therefore, the correlation between these two values can serve to quantify the accuracy of different estimators. This analysis is carried out in an example session in Fig. 5C, and quantified over 65 sessions in Fig. 5D. We find the LP model (Pearson's r=0.88±0.01, mean±sem) significantly improves over the baseline DeepLabCut model (r=0.36±0.03). (The LP+EKS estimate uses a low-dimensional state space model to denoise the data, therefore enforcing equality between the horizontal and vertical diameter estimates by construction.)

Scientifically, we are interested in how behaviorally-relevant events impact pupil dynamics, as well as the correlation between pupil dynamics and neural activity. We expect that noise in our estimates of pupil diameter would reduce the apparent consistency of pupil dynamics across trials and would also reduce any correlations between pupil diameter and neural activity; this is exactly what we observe (Fig. 5E-H). In Fig. 5E, we align diameter estimates across multiple trials to the time of reward delivered at the end of each successful trial. We define a trial consistency metric by taking the variance of the mean pupil diameter trace and dividing by the variance of the mean-subtracted traces across all trials. This metric is 0 if there are no reproducible dynamics across trials; it is infinity if the pupil dynamics are identical across trials. The LP and LP+EKS estimates show significantly greater trial-to-trial consistency than the baseline DeepLabCut estimates, both within a single session (Fig. 5E) and across multiple sessions (Fig. 5F; DeepLabCut 0.34±0.05; LP 0.61±0.07; LP+EKS 0.73±0.08). Similarly, when we decode pupil diameter from simultaneously recorded neural activity (Fig. 5G), the decoding accuracy is again increased when we use the LP or LP+EKS estimators compared to the DLC estimator (DLC R^2=0.27±0.02; LP 0.33±0.02; LP+EKS 0.35±0.02). Thus, we see that the accuracy improvements offered by LP and LP+EKS here lead to significantly increased reliability of the estimates for this important behavioral signal.

[2] To directly compare our methods to the publicly released IBL DeepLabCut traces, we do not split the data into InD/OOD subsets (which would allow us to compute OOD pixel error), but rather train on all available data and evaluate on held-out unlabeled videos.

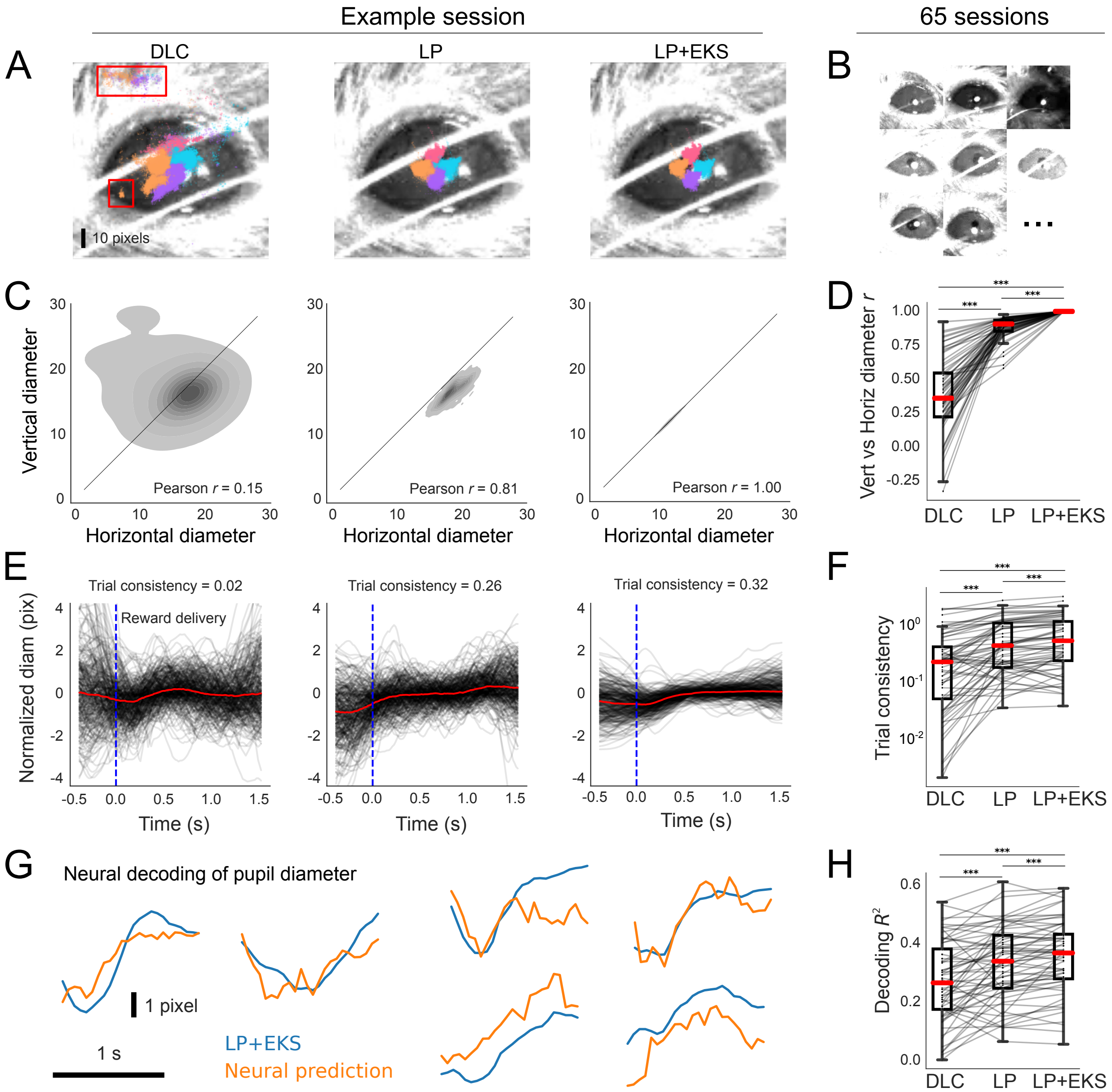


Figure 5: **Lightning Pose models and ensemble smoothing improve pose estimation on IBL pupil data.** **A**. Sample frame overlaid with a subset of pupil markers estimated from DeepLabCut (DLC; *left*), Lightning Pose using a semi-supervised context model (LP; *center*), and a 5-member ensemble using semi-supervised context models (LP+EKS; *right*); note erroneous DeepLabCut predictions near the top of the frame. **B**. Example frames from 65 IBL sessions, illustrating the diversity of imaging conditions in the dataset. **C**. Empirical distribution of vertical diameter measured from top and bottom markers scattered against horizontal pupil diameter measured from left and right markers. These estimates should ideally be equal, i.e., the distribution should lie near the diagonal line. Column arrangement as in panel A. Vertical vs horizontal correlation is greatly increased in LP compared to DeepLabCut; the LP+EKS estimate imposes a low-dimensional model that enforces perfectly correlated vertical and horizontal diameters by construction. **D**. Vertical vs horizontal diameter correlation is computed across 65 sessions for each model. The correlation is generally low for DeepLabCut models and is greatly improved by the LP model. The LP+EKS model has a correlation of 1.0 by construction. **E**. Pupil diameter is plotted for correct trials aligned to feedback onset; each trial is mean-subtracted. DeepLabCut and LP diameters are smoothed using IBL default post-processing (see Methods), compared to LP+EKS outputs. We compute a *trial consistency* metric (the variance explained by the mean over trials; see text) as indicated in the panel titles; LP significantly reduces the variability across trials compared to DeepLabCut. See Fig. V3 for a link to the corresponding video. **F**. The trial consistency is also computed across 65 sessions; DeepLabCut has the most variable traces, which is improved by the LP model and again by LP+EKS. **G**. Example traces of LP+EKS pupil diameters (blue) and predictions from neural activity (orange) for several trials using cross-validated, regularized linear regression (Methods). **H**. DeepLabCut decoding performance across 65 sessions is improved by the LP model, and further improved by LP+EKS.

We obtain similar performance gains when tracking the paws in the IBL videos, a dataset which also contains thousands of labeled frames (Figs. S6 and S7). We use two cameras to track the paws and, therefore, we can use multi-view consistency to help quantify the rate of paw tracking errors. Specifically, we use canonical correlations analysis (CCA) to find directions of motion that must match in the left and right cameras, and then we quantify the correlation values of these two directions of motion. We find that the correlation values obtained by DLC can be fairly low in some sessions (r=0.83±0.02, mean±sem over 40 sessions), largely due to occlusions or to frames in which one paw is confused for the other. LP networks improve these correlations slightly (r=0.86±0.02), and the EKS that enforces multi-view consistency pushes these correlation values to 1, by construction (Fig. S7C,D). Next we align trials to movement onset and examine the reproducibility of paw speed across trials, again finding LP and LP+EKS improvements (Fig. S7E,F; DLC 0.09±0.01; LP 0.12±0.02; LP+EKS 0.15±0.02). As in the pupil setting, we also quantify the correlation between paw speed and neural activity using a simple decoding analysis, and find that LP+EKS but not LP leads to greater decoding accuracy (Fig. S7H; DLC R^2=0.23±0.02; LP 0.22±0.02; LP+EKS 0.25±0.02). All quantities reported here refer to the right paw; see Fig. S7 caption for left paw values, which are qualitatively similar.

2.7 Lightning Pose and the Cloud App

2.7.1 Lightning Pose: software developer kit for scalable training and fast prototyping of new models

We built Lightning Pose with the following philosophy. To begin with, computer vision is a vast field, of which animal pose estimation is a small part. The thriving deep learning software ecosystem offers well-engineered and well-tested solutions for every stage of the pose estimation pipeline. We can therefore outsource code to these frameworks to a large degree, leaving us with a smaller code base to maintain.

Within this broader context, we built Lightning Pose to be: 1) *video-centric*: we train and evaluate networks on raw videos, rather than a single image at a time; 2) *modular and extensible*: our goal is to facilitate prototyping of new models and losses; 3) *simple*: we aim to minimize boilerplate code by outsourcing to industry-grade packages and graphical user interfaces; 4) *scalable*: we support efficient parallel training and evaluation; 5) *interactive*: we offer a variety of tracking performance metrics and visualizations during and after training, enabling easy model comparison and outlier detection. Below we describe how we achieve each of these goals. We start with Lightning Pose's core components, which are depicted in the innermost purple box in Fig. 6A.

First, most networks are built to ingest images, not videos. The standard approach converts raw behavioral videos into formatted ("augmented") images using CPUs. This approach is inefficient and may cause the network to spend most of its time idly waiting for data instead of training ("data bottleneck"; Zolnouri et al., 2020.) We built high-performance video readers using NVIDIA's data loading library (DALI; https://github.com/NVIDIA/DALI; leftmost box inside innermost purple box in Fig. 6A). DALI uses the native capabilities of Graphics Processing Units (GPUs) to both read ("decode") and augment videos (resize, crop, scale, etc) to greatly accelerate video handling at training and prediction time.

Moreover, Lightning Pose decouples network architectures from the losses which they minimize (center and right boxes, respectively, inside innermost purple box in Fig. 6A). First, network backbones and prediction heads have a dedicated module that can be arbitrarily extended and is isolated from other modules pertaining to data and losses. Second, we built a "loss factory" that enables developers to quickly prototype new losses, whether they are generally applicable or dataset-specific. Losses can be applied at any level of representation in the network, ranging from the timeseries of predicted keypoints, through heatmaps, to hidden network features. New losses require minimal extra code, they are automatically logged during training, and can contain their own trainable parameters and even trainable sub-networks.

Having established how we handle data, design networks, and select losses, we still need a procedure for training networks. We offload this task to PyTorch Lightning (Falcon et al., 2020; middle box in Fig. 6), which is an increasingly popular wrapper around the PyTorch deep learning framework (Paszke et al., 2019). This enables us to use the latest strategies for training models, logging the results, and distributing computation across multiple GPUs, without having to modify any of our core modules described above. As new training techniques emerge at a rapid pace, PyTorch Lightning enables us to adapt quickly. In addition, we use Hydra (Yadan, 2019) to configure, launch, and log network training jobs (Fig. 6A, outermost purple box.) We do away with a substantial amount of "boilerplate" code while increasing the reproducibility of training, which often depends on choices of random number generator, batch sizes, etc.

Finally, we developed a suite of interactive training diagnostics and model comparison tools, facilitating hyperparameter sensitivity analyses (see Fig. 6, right gray box.) During training, we provide online access to TensorBoard (`https://www.tensorflow.org/tensorboard`) to monitor the individual losses. After training, we use a Streamlit (`https://streamlit.io/`) user interface to visualize per-keypoint diagnostics for both labeled frames and unlabeled videos. We also use a FiftyOne user interface (`https://voxel51.com/`) for viewing images and videos along with multiple models' predictions, enabling users to filter body parts and models, and browse moments of interest in predicted videos.

2.7.2 A cloud-hosted application for pose estimation as a service

Deep learning pipelines – including animal pose estimation software like DeepLabCut and Lightning Pose – require access to GPU-accelerated hardware with a set of pre-installed drivers. More and more laboratories have access to local or remote accelerated computers, but unfortunately, even experienced software developers face hurdles when installing, executing, and maintaining deep learning pipelines on them. Oftentimes, it takes developers more time to set up the stack of software and hardware than to configure, train, and visualize pose estimation pipelines.

We built a browser application that uses cloud computers and allows users with no coding expertise to estimate animal pose using any computer, phone, or tablet with access to internet (see demo). Our app supports the full life cycle of animal pose estimation from data annotation via LabelStudio (`https://labelstud.io/`) to model training to video prediction and disagnostic visualization (via the open-source ecosystem introduced above.) When launched by a user, the app starts a number of cloud machines equipped with the necessary hardware and software, which will turn off when idle. Our app is built on Lightning.ai's (`https://lightning.ai/`; Fig. 6B) infrastructure for cloud-hosted deep learning applications, removing technical obstacles related to resource provisioning, secure remote access, and software dependency management.

As argued in Abe et al., 2022b, the cloud-centric approach we take serves to democratize analysis tools, improving scalability, code maintenance requirements, and computation time and cost. Our app enables developers who have created new losses or network architectures within the Lightning Pose software package to easily make these advances available to the broader audience through the cloud-based app. This ability significantly accelerates the process of moving model development from the prototyping to production stage. Finally, we note that the app can simply be run locally if the correct hardware and software are installed. More details on the application appear in 5.7.

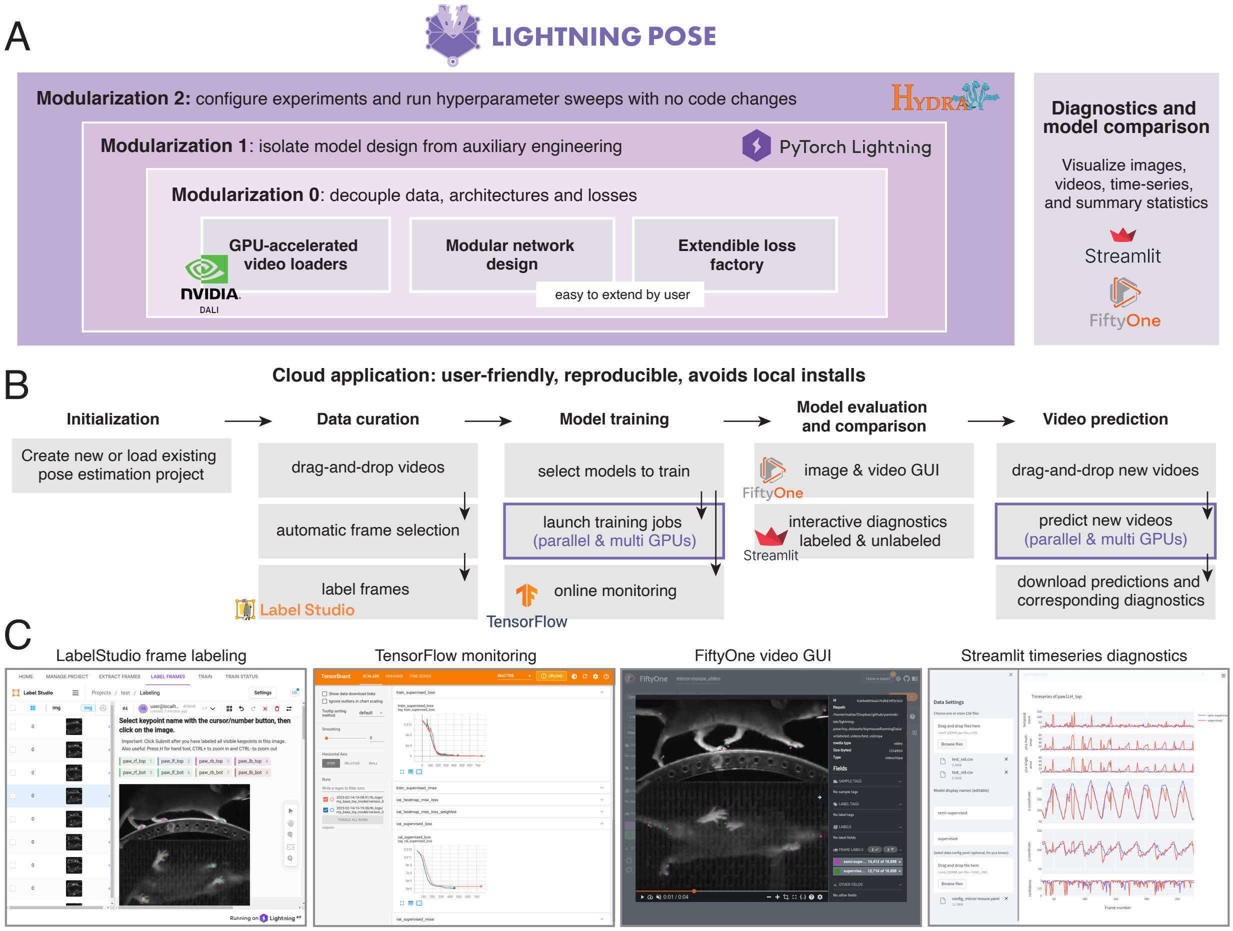

A
LIGHTNING POSE
Modularization 2: configure experiments and run hyperparameter sweeps with no code changes
HYDRA
Modularization 1: isolate model design from auxiliary engineering
PyTorch Lightning
Modularization 0: decouple data, architectures and losses
GPU-accelerated video loaders
NVIDIA DALI
Modular network design
Extendible loss factory
easy to extend by user
Diagnostics and model comparison
Visualize images, videos, time-series, and summary statistics
Streamlit
FiftyOne
B
Cloud application: user-friendly, reproducible, avoids local installs
Initialization
Create new or load existing pose estimation project
Data curation
drag-and-drop videos
automatic frame selection
label frames
Label Studio
Model training
select models to train
launch training jobs (parallel & multi GPUs)
online monitoring
TensorFlow
Model evaluation and comparison
image & video GUI
FiftyOne
interactive diagnostics labeled & unlabeled
Streamlit
Video prediction
drag-and-drop new vidoes
predict new videos (parallel & multi GPUs)
download predictions and corresponding diagnostics
C
LabelStudio frame labeling
TensorFlow monitoring
FiftyOne video GUI
Streamlit timeseries diagnostics

Figure 6: **Lightning Pose enables easy model development, fast training, and is accessible via a no-install cloud application served from the browser. A**. Our software developer kit outsources many tasks to existing tools within the deep learning ecosystem, remaining with a lighter, modular package that is easy to maintain and extend. The innermost purple box indicates the core components: accelerated video reading (via NVIDIA DALI), modular network design, and our general-purpose loss factory. The middle purple box denotes the training (and logging) operations which we outsource to PyTorch Lightning, and the outermost purple box denotes our use of the Hydra job manager. The right box depicts a rich set of interactive diagnostic metrics which are served via Streamlit and FiftyOne GUIs. **B**. A diagram of our cloud application. The application's critical components are dataset curation, parallel model training, interactive performance diagnostics, and parallel prediction of new videos. **C**. Screenshots from our cloud application (see demo). From left to right: LabelStudio GUI for frame labeling, TensorFlow monitoring of training performance overlaying two different networks, FiftyOne GUI for comparing these two networks' predictions on a video, and a Streamlit application that shows these two networks' timeseries of predictions, confidences, and spatiotemporal constraint violations.

3 Discussion

We have presented a deep learning system for easy prototyping and scalable training of pose estimation networks using both labeled frames and many unlabeled videos (semi-supervised learning). We have introduced spatiotemporal constraints that capture a set of prior beliefs on postural dynamics. Violations of these constraints signal erroneous network predictions, and training networks to minimize the violation rate significantly improves network reliability and efficiency. Furthermore, our cloud software infrastructure allowed us to train ensembles of our new networks in parallel, and fuse their collective uncertainty with our spatiotemporal constraints via the ensemble Kalman smoother. Our new approach dramatically improves pose estimation accuracy and reliability in the large public International Brain Laboratory video database. This improved pose estimation accuracy in turn exposes clearer correlations between neural activity and behavior in these datasets.

Semi-supervised Learning. Humans learn throughout their lives both with and without explicit supervision (Bröker et al., 2022). Semi-supervised learning is a well-known technique for improving generalization performance with limited labeled data and abundant unlabeled data (Chapelle et al., 2009), and is one of the ingredients that have recently enabled training ever larger networks on ever larger unlabeled datasets for contemporary achievements in language and vision tasks (Dai et al., 2015; Devlin et al., 2018; Brown et al., 2020). Our work builds on previous semi-supervised animal pose estimation algorithms that used spatiotemporal losses on unlabeled videos. Wu et al., 2020 used a frame-by-frame architecture for single-view datasets, and introduced a temporal continuity loss, as well as a spatial loss that enforces fixed distances between joints. In addition to the major software engineering improvements detailed in Sec. 2.7, we introduce the Temporal Context Network architecture, generalize the spatial penalty to whole-body configurations using Pose PCA, and incorporate a multi-view consistency loss (Jafarian et al., 2018; Zhang et al., 2020).

Ensembling of pose estimation networks. Previous work (Abe et al., 2022b) showed that using the mean prediction of an ensemble of multiple networks leads to better pose estimates. Our ensemble Kalman smoother (EKS) greatly denoises the ensemble means and leads to a significant reduction in pixel errors (Fig. S3C). EKS leverages the time-varying ensemble variance to determine the degree of smoothing by our spatiotemporal priors. Put differently, our efficient Bayesian inference scheme uses reliable keypoints and frames to infill unreliable ones. The combination of deep ensemble predictions with Bayesian priors is a technical frontier in the literature; it has been recently shown that training deep ensembles with added Bayesian priors leads to generalization improvements (Osband et al., 2018; Ciosek et al., 2020; He et al., 2020a). The benefits of training an EKS model end-to-end should be studied in future work.

Unsupervised keypoint discovery. A recent line of work abandons supervision altogether: by training to predict the spatiotemporal differences between pairs of video-frames, a network can "discover" a predefined number of keypoints in an unsupervised fashion (Sun et al., 2022b). When tracking a set of fully visible keypoints, such as a handful of dorsal keypoints on a mouse, this approach works well without any manual annotation. However, it struggles to discover keypoints that are not always visible due to occlusion or motion blur (a problem somewhat alleviated by multi-view videos; Sun et al., 2022a). These are scenarios for which we tailored our semi-supervised losses and TCN. However, our approach necessitates a small set of labeled frames to estimate parameters for semi-supervised learning (PCA subspaces for multi-view and pose plausability losses). It could be useful to use the predictions of a keypoint discovery algorithm as (pseudo) labeled frames from which Lightning Pose can embark on semi-supervised training.

Alternative methods for improving supervised pose estimation. Semi-supervised learning is not the only technique that enables improvements over standard transfer learning protocols. First, it has been suggested that supervised pose estimation networks can be significantly improved by pre-training them on a diverse set of large labeled animal pose datasets (Ye et al., 2022), to an extent that might eliminate dataset-specific training. Other work avoids pre-training altogether by using lighter architectures (Pereira et al., 2022). These ideas are complementary to ours: any robust backbone that will be obtained following these new procedures could be easily integrated into Lightning Pose, and be further refined with our semi-supervised learning protocols.

Multi-view pose estimation. Two camps coexist in multi-view animal pose estimation: those who use 3D information during training (Jafarian et al., 2018; Günel et al., 2019; Dunn et al., 2021; Schneider et al., 2022; Sun et al., 2022a) and those who train standard 2D networks and perform 3D reconstruction post-hoc (Mathis et al., 2020; Karashchuk et al., 2021). Either approach involves camera calibration, whose limitations we discussed in Sec. 2.2.1.2. Lightning Pose would fall under the first camp, except that it trains with 3D constraints without an explicit camera calibration. At the same time, Lightning Pose does not provide an exact 3D reconstruction of the animal (rather a scaled, rotated and shifted version thereof). Our predictions – which have shown to be more geometrically consistent, smoother, and overall more reliable – could be readily used as inputs to existing 3D reconstruction pipelines. Some authors employed convolutional networks that operate on 3D location heatmaps, one video frame at a time (Dunn et al., 2021). Moving to such voxel representations has a high computational cost, even when the 3D heatmaps are made sparse (Schneider et al., 2022). Operating on 2D location heatmaps while enforcing multi-view constraints, as we do here, has lighter memory and run-time costs. Recently, 3D convolutional networks have been trained in a semi-supervised fashion with temporal constraints (akin to to Wu et al., 2020 and the current work) on the predicted 3D trajectories (Li et al., 2023). Our temporal constraints are enforced during training not only by a temporal loss, but also by the TCN architecture that explicitly operates over short video clips. Finally, we note one recent promising line of work that offers 3D reconstruction directly from a monocular video (Gosztolai et al., 2021).

The next generation of unsupervised losses. One promising direction involves implementing richer models of moving bodies as losses. Multiple approaches have been recently proposed for modeling pose trajectories post-hoc. These include probabilistic body models (Joska et al., 2021; Zhang et al., 2021; Monsees et al., 2022), mechanical models (Biderman et al., 2020), switching linear dynamical systems (Wu et al., 2020; Whiteway et al., 2021), and autoencoders (Karashchuk et al., 2021). These models could be made even more effective by being integrated into network training in a so-called end-to-end manner. Any model of pose dynamics, as long as it is differentiable, could be incorporated as an unsupervised loss.

Future work. A number of additional important directions remain for future work. We close with two. First, while the ensemble approach leads to significant improvements in robustness and accuracy, this comes at a cost: we need to train, store and run several network versions, and postprocess their predictions with EKS. One natural approach would be *knowledge distillation* (Hinton et al., 2015; also known as *model compression*; Buciluǎ et al., 2006): one can train a single ("student") network to emulate the ensemble-KS output ("teacher"), by using a much larger training dataset that includes the predictions for millions of video frames (not just the labeled set); the hope here is that, in prediction time,we could achieve similarly accurate output for a fraction of the computational cost. Second, while the methods proposed here can currently handle tracking distinguishable animals (e.g. a black mouse and white mouse), our methods would not apply directly to multi-animal tracking problems involving multiple similar animals (Lauer et al., 2022; Pereira et al., 2022), since to compute our unsupervised losses we need to be able to know which keypoint belongs to which animal. Thus adapting our approaches to the multi-animal setting remains an important open avenue for future work.

4 Code and Data Availability

4.1 Code

Lightning Pose's GitHub repository is available at https://github.com/danbider/lightning-pose. From the command-line interface, running `pip install lightning-pose` will install the latest release of Lightning Pose via the Python Package Index (PyPi).

The code for the cloud-hosted application is available at https://github.com/Lightning-Universe/Pose-app, and enables launching an **experimental** version of our app on cloud resources or locally by creating a Lightning.ai account.

As of late April 2023, we offer early access to our deployed cloud app, which is accessed directly from the browser, see demo. Please sign up for early access here https://forms.gle/A4cHr4qgj2FuAoGG9.

4.2 Data

We have made the IBL data used in this manuscript publicly available.

To access the labeled data used for training the pupil network, see https://ibl-brain-wide-map-public.s3.amazonaws.com/aggregates/Tags/2023_Q1_Biderman_Whiteway_et_al/_ibl_videoTracking.trainingDataPupil.27dcdbb6-3646-4a50-886d-03190db68af3.zip.

To access the labeled data used for training the paw network, see https://ibl-brain-wide-map-public.s3.amazonaws.com/aggregates/Tags/2023_Q1_Biderman_Whiteway_et_al/_ibl_videoTracking.trainingDataPaw.7e79e865-f2fc-4709-b203-77dbdac6461f.zip.

To access the data analyzed in Figs. 5 and S7, see the documentation at https://int-brain-lab.github.io/ONE/FAQ.html#how-do-i-download-the-datasets-cache-for-a-specific-ibl-paper-release and use the tag `2023_Q1_Biderman_Whiteway_et_al`. This will provide access to spike sorted neural activity, trial timing variables (stimulus onset, feedback delivery, etc.), the original IBL DLC traces, and the raw videos.

		In-distribution (Train)			Out-of-distribution (Test)		
	Size	Frames	Animals	Videos	Frames	Animals	Videos
Mirror-mouse	406×396	789	10	17	253	3	5
Mirror-fish	384×512	341	9	26	94	3	10
IBL-paw	102×108	6071	35	84	1446	10	19
IBL-pupil	100×100	2612	26	52	1012	8	8

Table 1: **Dataset details**. In-distribution (InD) frames are selected from one set of animals/videos. The number of videos is generally larger than the number of animals, indicating that we have multiple experimental sessions from each animal. Out-of-distribution (OOD) frames are selected from a non-overlapping subset of animals/videos. Frame size is (height × width).

5 Methods

5.1 Datasets

We consider diverse datasets collected via different experimental paradigms for mice and fish. For each dataset, we collected a large number of videos including different animals and experimental sessions, and labeled a subset of frames from each video. We then split this data into two non-overlapping subsets (i.e., a given animal and/or session would appear only in one subset). The first subset is the "in-distribution" (InD) data that we use for model training. The second subset is the "out-of-distribution" (OOD) data that we use for model evaluation. This setup mimics the common scenario in which a network is thoroughly trained on one cohort of subjects, and is then used to predict new subjects. Table 1 details the number of frames for each subset per dataset, as well as the number of unique animals and videos those frames came from.

Mirror-mouse. Head-fixed mice ran on a circular treadmill while avoiding a moving obstacle (Warren et al., 2021). The treadmill had a transparent floor and a mirror mounted inside at $45°$, allowing a single camera to capture two roughly orthogonal views (side view and bottom view via the mirror) at 250 Hz. Frames are (406×396) and reshaped during training to (256×256). 17 keypoints were labeled across the two views including 7 keypoints on the mouse's body per view, plus 3 keypoints on the moving obstacle.

Mirror-fish. Weakly-electric Mormyrid fish (of the species *Gnathonemus petersii*) swam freely in and out of an experimental tank, capturing worms from a well. The tank had a side mirror and a top mirror, both at $45°$, providing three different views seen from a single camera at 300 Hz. Frames are (384×512) and reshaped during training to (256×384). 15 body parts were tracked across all three views for a total of 45 keypoints.

Fish (15-22 cm in length) were housed in 60-gallon tanks in groups of 5-20. Water conductivity was maintained between 60-100 microsiemens both in the fish's home tanks and during experiments. All experiments performed in this study adhere to the American Physiological Society's Guiding Principles in the Care and Use of Animals and were approved by the Institutional Animal Care and Use Committee of Columbia University, protocol number AABN0557.

IBL-paw. This dataset (The International Brain Laboratory, 2023) comes from the International Brain Lab and consists of head-fixed mice performing a decision-making task (The International Brain Laboratory et al., 2021; The International Brain Laboratory et al., 2022a). Two cameras – "left" (60 Hz) and "right" (150 Hz) – capture roughly orthogonal side views of the mouse's face and upper trunk during each session. For model training, frames from the right camera were flipped to match the orientation of the body in the left camera. Frames were initially downsampled to (102×128) for labeling and video storage; frames were reshaped during training to (128×128). We tracked two keypoints per view, one for each paw. More information on the IBL video processing pipeline can be found in The International Brain Laboratory et al., 2022b. For the large scale analysis in Fig. S7 we selected 40 additional test sessions that were not represented in either the InD or OOD sessions listed in Table 1; these could be considered additional OOD data.

IBL-pupil. The pupil dataset also comes from the International Brain Lab. Frames from the right camera were spatially upsampled and flipped to match the left camera. Then, a 100×100 pixel ROI was cropped around the pupil. The frames were reshaped in training to (128×128). Four keypoints were tracked on the top, bottom, left and right edges of the pupil, forming a diamond shape. For the large scale analysis in Fig. 5 we selected left videos from 65 additional sessions that were not represented in either the InD or OOD sessions listed in Table 1.

5.2 Problem Formulation

Let K denote the number of keypoints to be tracked, and N the number of labeled frames. After manual labeling, we are given a dataset:

$$\mathcal{D}_s = \left\{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right\}_{i=1}^{N}, \quad \mathbf{x}^{(i)} \in \mathbb{R}^{W \times H}, \quad \mathbf{y}^{(i)} = \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_K \end{pmatrix} \in \mathbb{R}^{2K}, \tag{1}$$

where $\mathbf{x}^{(i)}$ is the i-th image and $\mathbf{y}^{(i)}$ its associated label vector, stacking the annotated (x, y) pixel coordinates for each of the K tracked keypoints.

It is standard practice to represent each annotated keypoint $\mathbf{y}_k,\ k = 1, \dots K$ as a heatmap $\mathbf{h}_k^{(i)} \in \mathbb{R}^{W_s \times H_s}$ with width W_s and height H_s, thus converting $\mathbf{y}^{(i)}$ to a set of K heatmaps $\left\{\mathbf{h}_k^{(i)}\right\}_{k=1}^{K}$. This is done by defining a bivariate Gaussian centered at each annotated keypoint with variance σ^2 (a controllable parameter), and evaluating it at 2D grid points (for more details, see Pereira et al., 2022). If $\mathbf{y}_k^{(i)}$ lacks an annotation (e.g. if it is occluded), we do not form a heatmap for it. We normalize the heatmaps $\sum_{l,m} h_k^{(i)}(l, m) = 1,\ \forall i, k,$ which allows us to both evenly scale the outputs during training and use losses that operate on heatmaps as valid probability mass functions. Then, the dataset for training supervised networks is just frames and heatmaps $\mathcal{D} = \left\{\mathbf{x}^{(i)}, \{\mathbf{h}_k^{(i)}\}_{k=1}^{K}\right\}_{i=1}^{N}$. To accelerate training, the heatmaps are made 4 or 8 times smaller than the original frames.

5.3 Model architectures

5.3.1 Baseline

Our baseline model performs heatmap regression on a frame-by-frame basis, akin to DeepLabCut (Mathis et al., 2018), SLEAP (Pereira et al., 2019), DeepPoseKit (Graving et al., 2019), and others. It has roughly the same architecture: a "backbone" network that extract a feature vector per frame, and a "head" that transforms these into K predicted heatmaps. In the results reported here, we use a ResNet-50 backbone network pretrained on AnimalPose10K dataset (Yu et al., 2021; 10,015 annotated frames from 54 different animal species), though our package, like others, is largely agnostic to backbone choices. The head includes a fixed PixelShuffle(2) layer that reshapes the features tensor from $(B, C \times r^2, H, W)$ to $(B, C, H \times r, W \times r)$ and a series of identical ConvTranspose2D layers that further double it in size (kernel size (3×3), stride (2×2), input padding (1×1), and output padding (1×1)) (Paszke et al., 2019). The number of ConvTranspose2D layers is determined by the desired shape of the output heatmaps, and most commonly two such layers are used. Each heatmap is normalized with a 2D spatial softmax with a temperature parameter $\tau = 1$. The supervised loss is a divergence between predicted heatmaps and labeled heatmaps. Here, we use squared error $\mathcal{L}_s = \sum_{l,m} \left(\hat{h}_k^{(i)}(l,m) - h_k^{(i)}(l,m) \right)^2$ but our package supports additional divergences like Kullback-Leibler and Jensen-Shannon which achieve comparable performance.

Once heatmaps have been predicted for each keypoint, we must transform these 2D arrays into estimates of the (x, y) coordinates in the original image space. We first upsample each heatmap $\mathbf{h}_k^{(i)} \in \mathbb{R}^{W_s \times H_s}$ to $\tilde{\mathbf{h}}_k^{(i)} \in \mathbb{R}^{W \times H}$ using bicubic interpolation. We then compute a subpixel maximum akin to DeepPoseKit (Pereira et al., 2022). A 2D spatial softmax renormalizes the heatmap to sum to 1, and we apply a high temperature parameter ($\tau = 1000$) to suppress non-global maxima. A 2D spatial expectation then produces a subpixel estimate of the location of the heatmap's maximum value. These two operations – spatial softmax followed by spatial expectation – are together known as a soft argmax (Wu et al., 2020). Importantly, this soft argmax operation is differentiable (unlike the location refinement strategy employed in Mathis et al., 2018), and allows the estimated coordinates to be used in downstream losses. To compute the confidence value associated with the (x, y) estimate we sum the values of the normalized heatmap within a configurable radius of the soft argmax.

5.3.2 Temporal Context Network

Many detection ambiguities and occlusions in a given frame can be resolved by considering some video frames before and after it. The Temporal Context Network (TCN) uses a sequence of $2n + 1$ frames to predict the labeled heatmaps for the middle frame,

$$\mathcal{D}_s = \left\{ \{\mathbf{x}_m^{(i)}\}_{m=-2n}^{2n}, \{\mathbf{h}_k^{(i)}\}_{k=1}^{K} \right\}_{i=1}^{N}, \tag{2}$$

where $\mathbf{x}_0^{(i)}$ is the labeled frame and, for example, $\mathbf{x}_{-1}^{(i)}$ is the preceding (unlabeled) frame in the video.

During training, batches of $2n+1$ frame sequences are passed through the backbone to obtain $2n+1$ feature vectors. The TCN has two upsampling heads, one "static" and one "context-aware," each identical to the baseline model's head. The static head takes the features of only the central frame and predicts location heatmaps for that frame. The context-aware head generates predicted location heatmaps for each of the

$2n + 1$ frames (note, these are the same shape as the location heatmaps, but we do not explicitly enforce them to match labeled heatmaps). Those heatmaps are passed as inputs to a bi-directional convolutional recurrent neural network whose output is the context-aware predicted heatmap for the middle frame. We then apply our supervised loss to both predicted heatmaps, forcing the network to learn the standard static mapping from an image to heatmaps, while independently learning to take advantage of temporal context when needed. (Recall Fig. 2E, which provides an overview of this architecture.)

The network described above outputs two predicted heatmaps per keypoint, one from each head, and applies the computations described in Sec. 5.3.1 to obtain two sets of keypoint predictions with confidences. For each keypoint, the more confident prediction of the two is selected for downstream analysis.

5.4 Semi-supervised learning

We perform semi-supervised learning by jointly training on labeled dataset $\mathcal{D}_s$ (constructed as described in Sec. 5.2) and an unlabeled dataset $\mathcal{D}_u$:

$$\mathcal{D}_{ss} \equiv \mathcal{D}_s \cup \mathcal{D}_u, \tag{3}$$

where $\mathcal{D}_u$ is constructed as follows.

Assume we have access to one or more unlabeled videos; we splice these into a set of U disjoint T-frame clips (discarding the very last clip if it has fewer than T frames),

$$D_u = \left\{ \mathbf{x}_u^{(1)}, \ldots, \mathbf{x}_u^{(T)} \right\}_{u=1}^{U}, \tag{4}$$

where, typically, $T = 32/64/96/128/256$ with with smaller frame sizes freeing up memory for longer sequences.

Now, assume we selected a mechanism (baseline model or TCN) for predicting keypoint heatmaps for a given frame. At each training step, in addition to a batch of labeled frames drawn from $\mathcal{D}_s$, we present the network with a short unlabeled video clip randomly drawn from $\mathcal{D}_u$. The network outputs a time-series of keypoint predictions (one pose for each of the T frames in the clip), which is then subjected to one or more of our unsupervised losses.

All unsupervised losses are expressed as pixel distance between a keypoint prediction and the constraint. Since our constraints are merely useful approximate models of reality, we do not require the network to perfectly satisfy them. We are particularly interested in preventing, and having the network learn from, severe violations of these constraints. Therefore, we enforce our losses only when they exceed a tolerance threshold ϵ, rendering them ϵ-insensitive:

$$\mathcal{L}(\epsilon) = \max(0, \mathcal{L} - \epsilon). \tag{5}$$

The ϵ threshold could be chosen using prior knowledge, or estimated empirically from the labeled data, as we will demonstrate below. $\mathcal{L}(\epsilon)$ is computed separately for each keypoint on each frame, and averaged to obtain a scalar loss to be minimized. Multiple losses can be jointly minimized via a weighted sum, with weights determined by a parallel hyperparameter search, which is supported in Lightning Pose with no code changes.

5.4.1 Temporal Continuity Loss

Keypoints should not jump too far between consecutive frames. We measure the jump in pixels and ignore jumps smaller than ϵ, the maximum jump allowed by user,

$$\mathcal{L}^{k,t}_{\text{temporal}}(\epsilon) = \max\left(0, ||\mathbf{y}_k(t) - \mathbf{y}_k(t-1)||_2 - \epsilon\right), \tag{6}$$

where ϵ could be easily determined based on image size, frame rate, and rough viewing distance from the subject. We compute this loss for a pair of successive predictions only when both have confidence greater than 0.9 to avoid artificially enforcing smoothness in stretches where the keypoint is unseen. We average the loss across keypoints and unlabeled frames:

$$\mathcal{L}_{\text{temporal}} = \frac{1}{TK}\sum_{t=1}^{T}\sum_{k=1}^{K}\mathcal{L}^{k,t}_{\text{temporal}}(\epsilon), \tag{7}$$

and minimize $\mathcal{L}_{\text{temporal}}$ during training. Lightning Pose also offers the option to apply the temporal loss on predicted heatmaps instead of the keypoints. We have found both methods comparable and focus on the latter for clarity.

5.4.2 Multi-view Consistency Loss

5.4.2.1 Background

Let $\bar{\mathbf{y}} \in \mathbb{R}^3$ be an unknown 3D keypoint of interest. Assume that we have V cameras and that each $v = 1, \dots, V$ camera sees a single 2D perspective projection of $\bar{\mathbf{y}}$ denoted as $\mathbf{y}(v) \in \mathbb{R}^2$, in pixel coordinates[3]. Thus, we have a $2V$-dimensional measurement $\begin{pmatrix}\mathbf{y}_k(1)^T & \cdots & \mathbf{y}_k(V)^T\end{pmatrix}$ of our 3D keypoint $\bar{\mathbf{y}}_k$.

The multi-view geometry approach. It is standard to model each view as a pinhole camera (Hartley et al., 2003): such a camera has intrinsic parameters (focal length and distortion) and extrinsic parameters (its 3D location and orientation, a.k.a "camera pose"), that together specify where a 3D keypoint will land on its imaging plane, i.e., the transformation from $\bar{\mathbf{y}}$ to $\mathbf{y}(v)$. This transformation involves a linear projection (scaling, rotation, translation) followed by a nonlinear distortion. While one might know a camera's focal length and distortion, in general, both the intrinsic and extrinsic parameters are not exactly known and have to be estimated. A standard way to estimate these involves "calibrating" the camera; filming objects with ground-truth 3D coordinates, and measuring their 2D pixel coordinates on the camera's imaging plane. Physical checkerboards are typically used for this purpose. They have known patterns that can be presented to the camera and detected using traditional computer vision techniques. Now with a sufficient set of 3D inputs and 2D outputs, the intrinsic and extrinsic parameters can be estimated via (nonlinear) optimization.

Multi-view PCA on the labels (our approach). We take a simpler approach which does not require camera calibration. Our first insight is that the multi-view ($2V$-dimensional) labeled keypoints could be used as keypoint correspondences to learn the geometric relationship between the views. We approximate the pinhole camera as a linear projection (with zero distortion), and estimate the parameters of this linear projection by fitting PCA on the labels (details below), and keeping the first 3 PCs, since all we are measuring from our different cameras is a single 3D object. Fig. 2C (bottom-right) confirms that our PCA model can explain

[3] Note that is standard to express $\bar{\mathbf{y}}$ and $\mathbf{y}_{(v)}$ in "homogeneous coordinates" (i.e., appending another element for each vector), yet we omit this for simplicity and for a clearer connection with our PCA approach.

$> 99\%$ of the variance with the first 3 PCs in several multi-view experimental setups, indicating that our linear approximation is suitable at least for the mirror-mouse and mirror-fish datasets, in which the camera is relatively far from the subject. We do anticipate cases where our linear approximation will not be sufficiently accurate (e.g., strongly distorted lenses, or highly zoomed in); the more general epipolar geometry approach of Jafarian et al., 2018; Zhang et al., 2020 could be applicable here. Note that our three-dimensional PCA coordinates do not exactly match the (x, y, z) physical coordinates of the keypoints in space; instead, these two sets of three-dimensional coordinates are related via an affine transformation.

5.4.2.2 Before training: fitting multi-view Principal Component Analysis (PCA) on the labels

Our goal is to estimate a projection from $2V$ dimensions ((x, y) pixel coordinates for V views) to three dimensions, which we could use to relate the different views to each other. We form a tall and thin design matrix by vertically stacking all the $2V$-dimensional multi-view labeled keypoints. We denote this matrix as $\mathbf{Y}_{MV} \in \mathbb{R}^{NK\times 2V}$,

$$\mathbf{Y}_{MV} = \begin{pmatrix} \mathbf{y}_1^1(1)^T & \cdots & \mathbf{y}_1^1(V)^T \\ \mathbf{y}_2^1(1)^T & \cdots & \mathbf{y}_2^1(V)^T \\ \vdots & \vdots & \vdots \\ \mathbf{y}_K^N(1)^T & \cdots & \mathbf{y}_K^N(V)^T \end{pmatrix}, \tag{8}$$

where $\mathbf{y}_k^n(v) \in \mathbb{R}^2$ represents the (x, y) coordinates on frame n for keypoint k in camera v. To reiterate, each row contains the labeled coordinates for a single body part seen from V views. The rows of this matrix contain examples from all available labeled keypoints, which are all used for learning the 3D projection. We exclude rows in which a body part is missing from one or more views. The number of examples used to estimate PCA is, as desired, always much larger than the label dimension ($NK >> 2V$). We perform PCA on $\mathbf{Y}_{MV}$ and keep the first three PCs, which we denote as $\mathbf{P} = \begin{pmatrix}\mathbf{P}_1 & \mathbf{P}_2 & \mathbf{P}_3\end{pmatrix} \in \mathbb{R}^{2V\times 3}$ and the data mean $\boldsymbol{\mu} \in \mathbb{R}^{2V}$. The three PCs form three orthogonal axes in $2V$ dimensions, and projecting the $2V$-dimensional labels on them will provide (x, y, z)-like coordinates. These 3D coordinates are related to the "real-world" 3D coordinates (relative to some arbitrary "origin" point) by an affine transformation (they need to be rotated, stretched and translated), but critically, we do not need these "real-world" coordinates to apply the multi-view constraints during network training, as described below.

5.4.2.3 During training: Penalizing the unlabeled data for PCA reconstruction errors

Let $\hat{\mathbf{y}}_k^t = \begin{pmatrix}\hat{\mathbf{y}}_k^t(1)^T & \cdots & \hat{\mathbf{y}}_k^t(V)^T\end{pmatrix} \in \mathbb{R}^{2V}$ be the network's prediction for the k-th body part on the t-th unlabeled video frame, on all V views. The prediction's multi-view PCA reconstruction is given by projecting it down to 3 dimensions and then back up to $2V$ dimensions:

$$\bar{\mathbf{y}}_k^t = (\hat{\mathbf{y}}_k^t - \boldsymbol{\mu})\mathbf{P}\mathbf{P}^\top + \boldsymbol{\mu}. \tag{9}$$

When the prediction $\hat{\mathbf{y}}_k^t$ is consistent across views, i.e., on the 3D hyperplane specified by $\mathbf{P}$, we will get $\bar{\mathbf{y}}_k^t = \hat{\mathbf{y}}_k^t$, a perfect reconstuction. The loss is defined as the average pixel distance between each 2D predicted keypoint $\hat{\mathbf{y}}_k^t(v)$ and its multi-view PCA reconstruction $\bar{\mathbf{y}}_k^t(v)$:

$$\mathcal{L}_{\text{MV-PCA}}^{k,t,v}(\epsilon) = \max\left(0, \left|\left|\hat{\mathbf{y}}_k^t(v) - \bar{\mathbf{y}}_k^t(v)\right|\right|_2 - \epsilon\right). \tag{10}$$

The loss encourages the predictions to stay within the fixed 3D hyperplane estimated by PCA, and thus be consistent across views. In training, we minimize its average across views, body parts, and frames

$$\mathcal{L}_{\text{MV-PCA}} = \frac{1}{TKV} \sum_{t,k,v} \mathcal{L}^{k,t,v}_{\text{MV-PCA}}(\epsilon) \tag{11}$$

We choose ϵ by computing the PCA reconstruction errors (in pixels) for each of the labeled keypoints, and taking the maximum. This represents the maximal multi-view inconsistency observed in the labeled data.

5.4.3 Pose PCA Loss

There are certain things that bodies cannot do. We might track $2K$ pose coordinates but it does not mean that they can all move independently and freely. Indeed, there is a long history of using low-dimensional models to describe animal movement (Tresch et al., 2009; Yan et al., 2020; Bialek, 2022). Here, we extend the PCA approach to full pose vectors, and constrain the $2K$-dimensional poses to lie on a low-dimensional hyperplane of plausible poses, which we estimate from the labels.

5.4.3.1 Before training: fitting Pose PCA on the labels

This approach is identical to multi-view PCA, with the following exceptions. First, our observations are full pose vectors and not single keypoints seen from multiple views. The design matrix of labels is therefore shorter and wider $\mathbf{Y}_{\text{P-PCA}} \in \mathbb{R}^{N\times 2K}$; it has as many rows as labeled frames, and each row contains the entire pose vector. Rows (poses) with missing bodyparts are discarded from this matrix. The number of examples available for PCA estimation is now simply the number of non-discarded labeled frames, N_{train}, which is not allowed to be smaller than the number of pose coordinates, i.e., $N_{\text{train}} \geq 2K$. A second exception is that instead of keeping three PCs, we keep as many PCs needed to explain 99% of the pose variance, denoted as $R << 2K$. We collect the kept PCs as columns of a $(2K \times R)$ matrix $\mathbf{P} = \begin{pmatrix}\mathbf{P}_1 & \cdots & \mathbf{P}_R\end{pmatrix}$. Each of the PCs represents an axis of plausible whole-body movement, akin to previous approaches (Stephens et al., 2008; Yan et al., 2020). Figure 2D shows that the number of kept PCs is usually less than half of the observation dimensions. We now keep $\mathbf{P}$ and $\boldsymbol{\mu} \in \mathbb{R}^{2K}$ to be used in training. For multi-view setups, it is possible to form an even wider $(N \times 2KV)$ design matrix, appending all V views, to jointly enforce the multi-view consistency loss. We have done so in the mirror-mouse and mirror-fish datasets.

5.4.3.2 During training: penalizing for implausible poses

As in Eq. 9, we project the full predicted poses down to the low-dimensional hyperplane, then back up to $2K$ dimensions, to form their Pose PCA reconstructions. Then, for each 2D keypoint on each unlabeled video frame, we define the loss as the pixel error between the raw prediction $\hat{\mathbf{y}}^t_k$ and its reconstruction $\bar{\mathbf{y}}^t_k$:

$$\mathcal{L}^{k,t}_{\text{P-PCA}}(\epsilon) = \max\left(0, \left|\left|\hat{\mathbf{y}}^t_k - \bar{\mathbf{y}}^t_k\right|\right|_2 - \epsilon\right). \tag{12}$$

This loss tells us how many pixels are needed to move the predicted keypoint onto the hyperplane of plausible poses. During training, we minimize the average loss across keypoints and frames,

$$\mathcal{L}_{P-PCA} = \frac{1}{TK} \sum_{t,k} \mathcal{L}^{k,t}_{\text{P-PCA}}(\epsilon) \tag{13}$$

Here too, ϵ is chosen by reconstructing the labeled pose vectors, computing the pixel error between each 2D labeled keypoint and its PCA reconstruction, and taking the maximum value.

5.5 Training

Batch sizes are determined based on image size and GPU memory. We used a batch size of 8 labeled frames for mirror-mouse, 8 for mirror-fish, 32 for IBL-paw, and 64 for ibl-pupil. The TCN models' batch sizes were five times larger, as we selected a 5-frame window around each label frame (two frames before and two after each labeled frame). We use an Adam optimizer (Kingma et al., 2014) with an initial learning rate of 0.001, halving it at epochs 150, 200, and 250. In the experiments reported here, the ResNet50 backbone was kept frozen for the first 20 epochs. We trained our models for a minimum number of 300 training epochs and a maximum number of 750 epochs. During training we split the InD data into training (80%), validation (10%), and test (10%) sets. We performed early stopping by checking the heatmap loss on validation data every five epochs and exiting training if it does not improve for three consecutive checks.

During training we apply standard image augmentations to labeled frames including geometric transforms (e.g. rotations and crops), color space manipulations (e.g. histogram equalization) and kernel filters (e.g. motion blur), following Mathis et al., 2018. A different random combination of augmentations is used for each frame in a batch. For the TCN architecture, the same augmentation combination is used for a labeled frame and its associated context frames. For the semi-supervised models, we apply augmentations to unlabeled video frames using DALI. A single random combination of augmentations is used for all video frames in a batch. Because the PCA losses are sensitive to geometric transforms, once the (x, y) coordinates have been inferred using the soft argmax described in Sec. 5.3.1 we apply the inverse geometric transform before computing unsupervised losses.

5.6 Diagnostics and model selection

5.6.1 Constraint violations as diagnostic metrics

After training, we evaluate the network on the the labeled frames and on unlabeled videos. We then compute our individual loss terms (defined in Eq. 6, 10,12) for each predicted keypoint, on each frame, and on each view for a multi-view setup, and use them as diagnostic metrics. For labeled frames, we compute the Euclidean pixel error. All metrics are measured as pixel distances on the full-sized image.

5.6.2 Model Selection based on pixel errors and constraint violations

Our versatile loss factory might require users to select among different applicable losses, and for each loss, determine its weight (note that we offer robust default values in our package). We start by fitting a baseline model to the data (typically with 3 random seeds). Then, for each of the applicable losses, we search over $4 - 8$ possible weights, in a parallel manner. We then compare the diagnostic metrics specified above on a held-out validation set (ignoring errors below a tolerance threshold). We pick the weight that exhibits the minimal loss across the majority of our diagnostics.

5.7 Cloud application

When an app is launched either from the browser of from the command-line interface, cloud machines are provisioned. The machines needed to initially configure a project or visualize results are simple CPU machines that cost a few cents an hour; for heavy video operations, network training, or video prediction, the app starts GPU machines that cost between 1.5-5 dollars an hour (training models will typically take between five minutes to an hour, depending on the size of the dataset). These will turn off when unused.

Once the app has started, the user can initialize a new project or load an existing project (with Lightning Pose, DLC-compatible folder structure). For a new project, the user enters some project metadata (name, body parts to track, etc) and then begins the data annotation process. This requires the user to first upload raw videos from their local machine (or phone, or tablet) to the browser using a simple drag-and-drop interface. The user then specifies how many frames they wish to label from each video. Then, a diverse set of frames are selected automatically as follows: we focus on half of the frames with higher motion energy (denoting that the animal is moving), compress the images via PCA, and cluster the compressed images using K-means. Finally, we pick one frame from each cluster, as in Mathis et al., 2018. Once the initial frames are selected, the user proceeds to a LabelStudio GUI (`https://labelstud.io/`) to label the previously selected body parts in each frame. A Lightning Pose and DLC-compatible dataset is generated from these annotations.

With a labeled training dataset in hand, the user specifies which and how many models they wish to train. They can specify different losses, different hyperparameters, or ensemble multiple models with different random seeds. At this stage, GPU-accelerated machines will be provisioned (in parallel), the annotated data and unlabeled videos will be transferred to them, and the network training will begin.

Following training, a number of diagnostic GUIs appear with detailed performance reports. First, the FiftyOne GUI enables users to inspect model predictions overlaid on labeled images and unlabeled videos. Users can zoom in, scroll, and slow down videos to better understand their networks' behavior. Individual keypoints or models can be examined separately and closely scrutinized. Second, we offer Streamlit GUIs including quantitative per-keypoint diagnostics for both labeled frames and unlabeled videos as described in Sec. 2.7. At any point, users can also upload a new set of videos and predict poses in parallel across multiple machines, as well as view the available diagnostics. With these systematic diagnostics in mind, the user can decide whether to label more frames or use more unlabeled videos for training. Then the training-prediction-diagnostics loop could be repeated.

As of late April 2023, we offer early access to our app. Please sign up at `https://forms.gle/A4cHr4qgj2FuAoGG9`.

5.8 Ensembling

To perform ensembling, we needed a collection of models that output a diverse set of predictions. While this can be achieved through various means, for our EKS analysis (Fig. S3) we chose to study a single split of the data, and achieved diversity by randomly initializing the head of each model, as well the order in which the data was sent to the model during training. Despite these seemingly minor differences, the ensemble of models produced a variety of outputs (Fig. S3D,F). For the other figures and videos related to ensembling (Figs. 5, S4, S5, S6, S7, V2, V3) we achieved diversity by training each model with a different subset of training data (in line with the analyses performed in, e.g., Fig. 4).

5.9 Ensemble Kalman smoother

The Ensemble Kalman smoother begins with the output of the ensemble of pose-estimation networks, and then performs probabilistic inference to denoise and summarize this ensemble to obtain more accurate and robust pose estimates. To be more specific, as above, let K be the number of keypoints and T the number of frames. Additionally denote m as the ensemble size; here we use ensembles of size $m \approx 5$. The ensemble Kalman smoother takes as input the output of the ensemble: this is an $m \times 2K \times T$ tensor. We compute the mean across the ensemble to obtain the $2K \times T$ ensemble mean M, and similarly compute the variance for each keypoint across the ensemble to obtain the $2K \times T$ ensembled per-keypoint variance C.

We consider several different cases below, but in each case we specify a latent *state* variable q_t, a linear Gaussian Markov *dynamics* model for this state variable of the form

$$q_t = A_t q_{t-1} + e_t, \ e_t \sim N(0, E_t), \tag{14}$$

and a linear Gaussian *observation* model describing the relationship between the latent state variable q_t and the observed data O_t,

$$O_t = B_t q_t + n_t, \ n_t \sim N(\mu, Q_t), \tag{15}$$

for some appropriate (potentially time-varying) system parameters A_t, B_t, E_t, Q_t, μ.

5.9.1 Single-keypoint, single-camera case

This is the simplest case to consider: let's imagine that we want to denoise each keypoint individually, and we only have observations from a single camera. Here the latent state q_t is the true two-dimensional position of the keypoint on the camera. Now our model is

$$q_t = q_{t-1} + e_t, \ e_t \sim N(0, sI) \tag{16}$$

$$O_t = q_t + n_t, \ n_t \sim N(0, (1/m)D_t), \tag{17}$$

Comparing these equations to the general dynamics and observations equations above, we see that $A_t = B_t = I$ here.

In the observation equation, O_t is the 2×1 keypoint vector, and D_t is a 2×2 diagonal matrix specifying the ensemble confidence about each observation. We use the t-th column of the ensemble mean M to fill in the observation O_t, and the covariance from the t-th frame of the ensemble covariance C to fill in the observation variance D_t (note that larger values of D_t correspond to lower confidence in the corresponding observation O_t). The factor of $(1/m)$ in the observation variance follows from the fact that O_t is defined as a sample mean over m ensemble members.

Finally, s is an adjustable smoothing parameter: larger s leads to less smoothing. This smoothness parameter could be selected by maximum likelihood (e.g., using the standard expectation-maximization algorithm for the Kalman model) but can be set manually for simplicity.

Now, given the specified dynamics and observation model, we can run the standard Kalman forward-backward smoother to obtain the posterior mean state Q given the observations O (i.e., all the states q_t given all the observations O_t). The smoother will "upweight" high-confidence observations O_t (i.e., small D_t), and "downweight" low-confidence observations (large D_t), e.g., from occlusion frames.

Note that this Kalman approach is the Bayesian optimal estimator under the assumption that the model in Equations 16-17 is accurate. In reality, this model holds only approximately: in general, neither the observation noise nor the state dynamics are exactly Gaussian. Therefore the ensemble Kalman smoother should be interpreted as an approximation to the optimal Bayesian estimator here. Generalizations (to handle multi-modal observation densities, or switching or stochastic volatility dynamics models) are left for future work.

5.9.2 Single-keypoint, multi-camera, synchronized cameras case

Given multiple cameras, we can estimate the true three-dimensional position of each keypoint. So letting the state vector q_t be the three-dimensional vector $q_t = (x_t, y_t, z_t)$, we have the model

$$q_t = q_{t-1} + e_t,\ e_t \sim N(0, E) \tag{18}$$

$$O_t = Bq_t + n_t,\ n_t \sim N(0, (1/m)D_t). \tag{19}$$

B is $2V \times 3$ where V is the number of camera views; this maps the three-dimensional state vector q_t onto the V camera coordinates (assuming linear observations here; this can be generalized but was not necessary for the data analyzed here). O_t is $2V \times 1$ and D_t is block-diagonal with 2×2 blocks. As above, observations O_t with high D_t (low confidence) will be downweighted by the resulting ensemble Kalman smoother: in practice, this means that cameras with an unobstructed view on a given frame (small D_t) can help to correct frames that are occluded in other camera views (resulting in larger ensemble variance D_t)[4].

We initialize our estimates by restricting to confident frames and computing PCA to estimate B; then we take temporal differences of the resulting PCA projections and compute their covariance to initialize E.

Finally, note that this simple Kalman model does not output the true 3d location here, because the model is non-identifiable; instead we learn q_t up to a fixed invertible affine transformation.

5.9.3 Pupil

For the IBL-pupil dataset, we track $K = 4$ keypoints arranged in a diamond shape around the perimeter of the pupil. Therefore, at each frame we have $2K = 8$ observations which are are constrained to lie in a three-dimensional subspace defined by the pupil center (denoted as (x_t, y_t)) and diameter d_t. Given the state variable $q_t = (d_t, x_t, y_t)$, we can (linearly) predict the location of each of the 4 diamond corners.

In addition, we have strong prior information about the dynamics of the state variable: we know that the diameter d_t is a smooth function of time t, while the pupil center (x_t, y_t) can change more abruptly, due to saccades and rapid face movements that move the eye as well.

Together, these assumptions lead to the model

$$q_t = Aq_{t-1} + e_t,\ e_t \sim N(0, E), \tag{20}$$

[4] In poorly trained models the opposite can also (on rarer occasions) be true: the ensemble in one camera view can make “confident mistakes" on some frames, in which all ensemble members output the same wrong estimate (with corresponding small D_t, i.e., high ensemble confidence) and induce errors in the other camera views after running the ensemble Kalman smoother. These errors can be detected as deviations between the Kalman smoother output and the original ensemble outputs; the training label set can then be augmented to correct these confident mistakes, followed by network ensemble retraining.

$$O_t = Bq_t + n_t,\ n_t \sim N\Big((\mu_d, 0, 0), (1/m)D_t\Big). \tag{21}$$

In the observation equation above, μ_d denotes the mean diameter, O_t is the 8×1 keypoint vector, B is a fixed 8×3 matrix that translates the state variable q_t into the keypoints, and D_t is a block-diagonal matrix (with 2×2 blocks) specifying the ensemble confidence about each observation.

In the dynamics model above, A and E are both diagonal. This means that we model the priors for d_t, x_t, and y_t using independent autoregressive (AR(1)) processes. (The posteriors for these variables will not be independent, due to the non-separable structure of the observation model 21.) We want to choose the diagonal values $diag(A)$ and $diag(E)$ so that these processes have the desired variance and time constant. The variance in a stationary AR(1) model with noise variance e and autoregressive parameter a is $e/(1-a^2)$. We can crudely estimate the marginal mean and variance of x_t, y_t, and d_t from the ensembled mean M, and match the AR(1) marginal mean and variance accordingly. This leaves us with just two autoregressive parameters to choose: $A(1,1)$ and $A(2,2)$ (with $A(3,3)$ set equal to $A(2,2)$). The time constant corresponding to $A(1,1)$ should be significantly larger than the time constant corresponding to $A(2,2)$, since as noted above the diameter d_t varies much more smoothly than the center (x_t, y_t).

5.9.4 Single-keypoint, multi-camera, asynchronous cameras case

In some datasets (e.g. the IBL paw dataset) frames from different cameras may be acquired asynchronously, perhaps with different frame rates. The Kalman model can be easily adapted to handle this case. Define the sampling times and camera ID for the i-th frame as: $\{t_i, c_i\}$, where t_i denotes the time the frame was acquired, and c_i denotes the camera that took the i-th frame. Again the state vector q_t is the true three-dimensional location of the keypoint, $q_t = (x_t, y_t, z_t)$. We have the model

$$q_{t_i} = q_{t_{i-1}} + e_i,\ e_i \sim N(0, E(t_i - t_{i-1})) \tag{22}$$

$$O_i = B_{c_i} q_{t_i} + n_i,\ n_i \sim N(0, (1/m)D_i), \tag{23}$$

where now B_{c_i} is 2×3; this tells us how the latent 3d coordinates are mapped into the c_i'th camera. O_i is a 2×1 vector, and D_i is a 2×2 matrix. Here the KS is run only at frame acquisition times $\{t_i\}$, but if desired we can perform predictions / interpolation at any desired time t.

5.9.5 Pose PCA case

Let q_t represent the "compressed pose," the $R \times 1$ vector obtained by projecting the true pose into the R-dimensional Pose PCA subspace. Here we have the model

$$q_t = q_{t-1} + e_t,\ e_t \sim N(0, E) \tag{24}$$

$$O_t = Bq_t + n_t,\ n_t \sim N(0, (1/m)D_t). \tag{25}$$

B is $2K \times R$; this maps the R-dimensional state vector q_t onto the $2K$ camera coordinates. O_t is $2K \times 1$ and D_t is block-diagonal with 2×2 blocks. As in the synchronous multi-camera setting, we initialize our estimates by restricting to confident frames and computing PCA to estimate B; then we take temporal differences of the resulting PCA projections and compute their covariance to initialize E.

The output of this smoother is useful for diagnostic purposes, but we do not recommend using this model to generate the final tracking output, since rare (but real) poses may lie outside the Pose PCA subspace, while the output of this smoother is restricted to lie within this subspace (the span of B) by construction.

5.9.6 Nonlinear observations and constrained dynamics

What if we have access to multiple well-calibrated cameras, as in e.g. (Dunn et al., 2021)? In this setting we can apply a strong constraint: limb lengths are constant, i.e., the 3D distance between body parts j and k should be fixed at some value L_{jk} (Karashchuk et al., 2021; Zhang et al., 2021).

We may also want to handle potential nonlinearities in the mapping from the true three-dimensional coordinates onto the camera plane.

How to incorporate these constraints and nonlinearities into the EKS? We can use the well-known fact that the Kalman smoother can be written as the solution to an optimization problem involving a block-tridiagonal Hessian (Paninski et al., 2010):

$$\hat{Q} = \arg\max_Q \log p(Q|O) = \arg\max_Q \log p(Q) + \log p(O|Q), \tag{26}$$

with

$$\log p(Q) = -\frac{1}{2}\sum_{tk}(q_{tk} - q_{t+1,k})^T E_k^{-1}(q_{tk} - q_{t+1,k}) \tag{27}$$

and

$$\log p(O|Q) = -\frac{1}{2}\sum_{tkv}(f_v(q_{tk}) - O_{tkv})^T D_{tkv}^{-1}(f_v(q_{tk}) - O_{tkv}). \tag{28}$$

Here t indexes frames, k bodyparts, and v camera views. The first term encodes the (Gaussian) dynamics model as above; the second term encodes the observation model, where the (potentially nonlinear) functions f_v map the keypoints into the v-th camera plane.

Finally, we can express the constraints in terms of penalties of the form

$$\sum_{tjk} -\lambda\Big(||q_{tj} - q_{tk}||_2 - L_{jk}\Big)^2, \tag{29}$$

for an appropriate Lagrange parameter λ.

Adding all these terms, we arrive at a non-quadratic objective function with a block tridiagonal Hessian; if we use a good initial Q then this objective function should be locally concave, and we can efficiently ascend via the usual block tridiagonal Newton-based iterations (with each Newton step requiring $O(T)$ computation). The output of this approach simultaneously triangulates the observations to obtain the true three-dimensional body part locations, while temporally smoothing and enforcing geometric constraints.

5.9.7 Relationship to previous work

We are far from the first to notice that the output of pose tracking networks can contain "glitches," and a number of strategies have been proposed for post-processing the network output to remove these glitches.

The simplest and perhaps most commonly applied strategy (Warren et al., 2021; Syeda et al., 2022; Weinreb et al., 2023) is to detect "bad" keypoints and frames and remove them, followed by simple temporal interpolation to fill in the resulting gaps in the estimated pose location traces. A number of criteria have been proposed to find bad frames, e.g., low network confidence, large temporal jumps, and/or multicamera inconsistency.

While attractively simple, this "remove-then-interpolate" strategy is suboptimal for several reasons. First, it can be challenging to automatically and reliably choose thresholds to determine which bad frames should be dropped. Second, using simple temporal interpolation to replace removed keypoints ignores useful spatial constraints (such as multi-view consistency constraints) which, as we have seen, can significantly improve the estimation of uncertain keypoints. Third, networks often make errors confidently (recall Fig. 1B) that may not be corrected with this simple strategy. Finally, even error frames often contain partial information about keypoint location (for example, a keypoint may be near the estimated value, but not match the estimated value exactly), and removing error frames completely discards this useful partial information.

A number of more complex denoising strategies have been proposed in the single-animal pose tracking literature (Karashchuk et al., 2021; Zhang et al., 2021; Monsees et al., 2022; Ebrahimi et al., 2023), in addition to post-processing strategies for the multi-animal tracking case (Lauer et al., 2022; Pereira et al., 2022) that are beyond the scope of this paper. These advanced techniques vary widely in their complexity, computational demands, assumptions, generality, outlier-handling logic, etc., but they all operate on the output of a single network. As we have seen, the output of a single network (particularly a fully-supervised network) can be highly unreliable, and moreover the reliability (as measured by the ensemble variance) can vary sharply from frame to frame. Without well-calibrated information about the reliability of each frame, it can be difficult to correct network errors.

Our ensemble Kalman smoother approach, on the other hand, uses a very simple Bayesian smoothing model, but takes advantage of the ensemble variance in each frame (and keypoint) to downweight unreliable estimates. Thus the ensemble-KS approach makes use of information even in "bad" points, and uses both spatial and temporal information to denoise these bad points, all without the need for the user to set any manual thresholds to detect these "bad" points. Once the ensemble has been run our method is simpler, faster, more interpretable, and easier to tune than the more complex strategies discussed in (Karashchuk et al., 2021; Zhang et al., 2021; Monsees et al., 2022). However, it is important to note that these more complex methods are complementary to ours: future work could combine our ensembling strategy with the more realistic nonlinear constraints and non-Gaussian observation models developed in these previous papers, to potentially obtain further accuracy improvements (at the cost of longer computational post-processing times).

Finally, a note on terminology: the ensemble Kalman smoother we use here is different from the Ensemble Kalman filter commonly used e.g. in weather prediction (Katzfuss et al., 2016). The two approaches differ in whether ensembling is performed in the dynamics step or the observation step of the Kalman filter model. In our case, the ensemble is used to generate the observation model (which is then smoothed with a simple linear-Gaussian dynamical system model), whereas in the weather prediction context ensembling is performed over multiple instances of nonlinear dynamics models, which are then combined with a simple Kalman-like observation update.

5.10 Neural decoding

We performed neural decoding using cross-validated linear regression with L2 regularization (the Ridge module in scikit-learn (Pedregosa et al., 2011)). The decoding targets – pupil diameter or paw speed – are binned into non-overlapping 20 ms bins. For each successful trial, we select an alignment event – reward delivery for pupil diameter and wheel movement onset for paw speed – and decode the target starting 200 ms before and ending at 1000 ms after the alignment event. We bin spike counts similarly using all recorded neurons in each session. The target value for a given bin (ending at time t) is decoded from spikes in a preceding (causal) window spanning B bins (ending at times t, . . . , t-B+1). Therefore, if decoding from N neurons, there are BN predictors of the target variable in a given bin. In practice we use $B = 10$.

To improve decoding performance, we smoothed the target variables. For pupil diameter, both the DLC and Lightning Pose (LP) predictions of pupil diameter were smoothed using a Savitzky-Golay filter that linearly interpolates over low-confidence time points (confidence <0.9). The right video filter window is set to 75 frames (500 ms) and the left video filter window is set to 31 frames (517 ms). For more details of this method, see (The International Brain Laboratory, 2023). We did not apply additional smoothing to the output of the ensemble Kalman smoother (LP+EKS) model. For paw speed, small errors in the paw position will be magnified when taking the derivative. To compensate for this we lightly smoothed the paw position estimates using a Savitzky-Golay filter after linearly interpolating over low-confidence time points (confidence <0.9), and then computed paw speed. The right video filter window is set to 13 frames (87 ms) and the left is set to 7 frames (117 ms). This smoothing was applied to the outputs of all three models (DLC, LP, LP+EKS).

All decoding results use nested cross-validation. Each of five cross-validation folds is based on a training/validation set comprising 80% of the trials and a test set of the remaining 20% of trials. Trials are selected at random (in an "interleaved" manner). The training/validation set of a fold is itself split into five sub-folds using an interleaved 80%/20% partition. A model is trained on the 80% training set using various regularization coefficients ($\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}\}$, denoted as input parameter α by sklearn), and evaluated on the held-out validation set. This procedure is repeated for all five sub-folds. The coefficient which achieves the highest R^2 value, averaged across all five validation sets, is selected as the "best" coefficient and used to train a new model across all trials in the 80% training/validation set. The model is then used to produce predictions for each trial in the 20% test set. This train/validate/test procedure is repeated five times, each time holding out a different 20% of test trials such that, after the five repetitions, 100% of trials have a held-out decoding prediction. The final reported decoding score is the R^2 computed across all held-out predictions. Code for performing this decoding analysis can be found at `https://github.com/int-brain-lab/paper-brain-wide-map`.

5.11 Canonical correlations analysis (CCA)

In Fig. S6 and Fig. S4 we use canonical correlations analysis to compute the directions of motion that should match in the left and right cameras and top and bottom cameras, respectively. (These canonical correlations directions are orthogonal to the epipolar lines familiar from multiple view geometry (Hartley et al., 2003).) In this subsection we provide details of this computation. Let $\hat{O}_t = B\hat{q}_t$ be the output of the multi-camera ensemble Kalman smoother at time step t, projected back onto the camera planes. We can further decompose $\hat{O}_t$ as $\hat{O}_t = \{\hat{O}_t^{c_1}, \hat{O}_t^{c_2}\}$, where $\hat{O}_t^{c_1}$ is the two-dimensional prediction for the first camera and $\hat{O}_t^{c_2}$ is the two-dimensional prediction for the second camera. Now, we compute $CCA(\hat{O}^{c_1}, \hat{O}^{c_2})$ to find the one-dimensional linear projection of the outputs for each camera that maximizes their correlation. Since $\hat{O}_t$ is generated from a lower-dimensional set of latents q_t, the projection of $\hat{O}^{c_1}$ and $\hat{O}^{c_2}$ onto the first canonical component will be perfectly correlated. We can then project the original model predictions for each camera onto the first canonical component for each camera. Any frames where the two camera-views do not have the same projected value will most likely be outliers. This can be seen in Fig. S6, where outlier frames due to paw switching and paw occlusions cause the model predictions for the two camera views to have different CCA projections. This can also be seen in Fig. S4, where outlier frames due to paw occlusions and paw switching also cause the model predictions for the two camera views to have different CCA projections.

5.12 Acknowledgments

Thanks to William Falcon, Luca Antiga, Thomas Chaton and Adrian Wälchi (Lightning AI) for their technical support and advice on implementing our package and the cloud application, and to Kelly Buchanan, Taiga Abe, and Geoff Pleiss for helpful discussions on ensembling. We thank Peter Dayan and Nick Steinmetz for serving on our IBL paper board. We are grateful to Natalie Biderman for interesting discussions and help with visualization. We thank Matteo Carrandini and Jacob Portes for helpful comments. This work was supported by the following grants: Gatsby Charitable Foundation GAT3708, Irma T Hirschl Trust, NIH K99NS128075, NIH NS075023, NIH U19NS123716, NSF 1707398, NSF IOS-2115007, Simons Foundation 543023. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

5.13 Contributions

Conceptualization: DB, MRW, LP; Software Package — core development: DB, MRW, NRG; Software Package — contribution: CH, AV; Cloud Application — development: MRW, DB, RL, AV; First draft — writing: DB, MRW, LP; First draft — editing: DB, MRW, CH, LP; Data collection: RW, FP, DN, MS, JMH, AK, GTM, JPN, APV, KZS, AEU; Funding — JPC, NS, LP; Semi-supervised learning algorithms: DB, MRW, NRG, LP; deep ensembling: DB, MRW, CH, LP; Kalman Smoothing: CH, LP; Temporal Context Network: CH, DB, MRW, LP; Diagnostic tools and visualization: MRW, DB, AV; Neural network experiments and analysis: DB, MRW.

References

[1] Taiga Abe et al. “Deep ensembles work, but are they necessary?” *arXiv preprint arXiv:2202.06985* (2022) (page 16).

[2] Taiga Abe et al. “Neuroscience Cloud Analysis As a Service: An open-source platform for scalable, reproducible data analysis.” *Neuron* 110.17 (2022), pp. 2771–2789 (pages 3, 20, 22).

[3] Korleki Akiti et al. “Striatal dopamine explains novelty-induced behavioral dynamics and individual variability in threat prediction.” *Neuron* 110.22 (2022), pp. 3789–3804 (page 2).

[4] William H Beluch et al. “The power of ensembles for active learning in image classification.” *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 9368–9377 (page 16).

[5] Gordon J Berman et al. “Mapping the stereotyped behaviour of freely moving fruit flies.” *Journal of The Royal Society Interface* 11.99 (2014), p. 20140672 (page 2).

[6] William Bialek. “On the dimensionality of behavior.” *Proceedings of the National Academy of Sciences* 119.18 (2022), e2021860119 (pages 8, 31).

[7] Dan Biderman et al. “Inverse articulated-body dynamics from video via variational sequential Monte Carlo” (2020) (pages 4, 23).

[8] Célian Bimbard et al. “Behavioral origin of sound-evoked activity in mouse visual cortex.” *Nature Neuroscience* (2023), pp. 1–8 (page 2).

[9] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006 (page 16).

[10] Kristin Branson et al. “High-throughput ethomics in large groups of Drosophila.” *Nature methods* 6.6 (2009), pp. 451–457 (page 2).

[11] Franziska Bröker, Bradley C Love, and Peter Dayan. “When unsupervised training benefits category learning.” *Cognition* 221 (2022), p. 104984 (page 22).

[12] Tom Brown et al. “Language models are few-shot learners.” *Advances in neural information processing systems* 33 (2020), pp. 1877–1901 (pages 2, 22).

[13] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. “Model compression.” *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2006, pp. 535–541 (page 23).

[14] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. “Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews].” *IEEE Transactions on Neural Networks* 20.3 (2009), pp. 542–542 (pages 2, 6, 22).

[15] Zexin Chen et al. “AlphaTracker: a multi-animal tracking and behavioral analysis tool.” *biorxiv* (2020), pp. 2020–12 (page 2).

[16] Kamil Ciosek et al. “Conservative uncertainty estimation by fitting prior networks.” *International Conference on Learning Representations*. 2020 (page 22).

[17] Andrew M Dai and Quoc V Le. “Semi-supervised sequence learning.” *Advances in neural information processing systems* 28 (2015) (page 22).

[18] Nisarg Desai et al. “OpenApePose: a database of annotated ape photographs for pose estimation.” *arXiv preprint arXiv:2212.00741* (2022) (page 6).

[19] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding.” *arXiv preprint arXiv:1810.04805* (2018) (pages 2, 22).

[20] Timothy W Dunn et al. “Geometric deep learning enables 3D kinematic profiling across species and environments.” *Nature methods* 18.5 (2021), pp. 564–573 (pages 2, 7, 23, 37).

[21] Aghileh S Ebrahimi et al. “Three-dimensional unsupervised probabilistic pose reconstruction (3D-UPPER) for freely moving animals.” *Scientific Reports* 13.1 (2023), p. 155 (pages 7, 38).

[22] William Falcon et al. “PyTorchLightning/pytorch-lightning: 0.7. 6 release.” *Zenodo: Geneva, Switzerland* (2020) (pages 3, 20).

[23] Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. “Deep ensembles: A loss landscape perspective.” *arXiv preprint arXiv:1912.02757* (2019) (page 16).

[24] Dominic Gonschorek et al. “Removing inter-experimental variability from functional data in systems neuroscience.” *Advances in Neural Information Processing Systems* 34 (2021), pp. 3706–3719 (page 2).

[25] Adam Gosztolai et al. “LiftPose3D, a deep learning-based approach for transforming two-dimensional to three-dimensional poses in laboratory animals.” *Nature methods* 18.8 (2021), pp. 975–981 (page 23).

[26] Jacob M Graving et al. “DeepPoseKit, a software toolkit for fast and robust animal pose estimation using deep learning.” *Elife* 8 (2019), e47994 (pages 2, 4, 27).

[27] Semih Günel et al. “DeepFly3D, a deep learning-based approach for 3D limb and appendage tracking in tethered, adult Drosophila.” *Elife* 8 (2019), e48571 (page 23).

[28] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003 (pages 7, 29, 39).

[29] Bobby He, Balaji Lakshminarayanan, and Yee Whye Teh. "Bayesian deep ensembles via the neural tangent kernel." *Advances in neural information processing systems* 33 (2020), pp. 1010–1022 (page 22).

[30] Yihui He et al. "Epipolar transformers." *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*. 2020, pp. 7779–7788 (page 7).

[31] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network." *arXiv preprint arXiv:1503.02531* (2015) (page 23).

[32] Alexander I Hsu and Eric A Yttri. "B-SOiD, an open-source unsupervised algorithm for identification and fast prediction of behaviors." *Nature communications* 12.1 (2021), p. 5188 (pages 2, 11).

[33] Yasamin Jafarian, Yuan Yao, and Hyun Soo Park. "Monet: Multiview semi-supervised keypoint via epipolar divergence." *arXiv preprint arXiv:1806.00104* (2018) (pages 7, 22, 23, 30).

[34] Jessica M Jones et al. "A machine-vision approach for automated pain measurement at millisecond timescales." *Elife* 9 (2020), e57258 (page 2).

[35] Daniel Joska et al. "AcinoSet: a 3D pose estimation dataset and baseline models for Cheetahs in the wild." *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 13901–13908 (pages 4, 23).

[36] Pierre Karashchuk et al. "Anipose: a toolkit for robust markerless 3D pose estimation." *Cell reports* 36.13 (2021), p. 109730 (pages 2, 4, 7, 23, 37, 38).

[37] Matthias Katzfuss, Jonathan R. Stroud, and Christopher K. Wikle. "Understanding the Ensemble Kalman Filter." *The American Statistician* 70.4 (2016), pp. 350–357. DOI: 10.1080/00031305.2016.1141709 (page 38).

[38] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980* (2014) (page 32).

[39] John W Krakauer et al. "Neuroscience needs behavior: correcting a reductionist bias." *Neuron* 93.3 (2017), pp. 480–490 (page 2).

[40] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. "Simple and scalable predictive uncertainty estimation using deep ensembles." *Advances in neural information processing systems* 30 (2017) (pages 3, 16).

[41] Jessy Lauer et al. "Multi-animal pose estimation, identification and tracking with DeepLabCut." *Nature Methods* 19.4 (2022), pp. 496–504 (pages 23, 38).

[42] Tianqing Li et al. "Improved 3D Markerless Mouse Pose Estimation Using Temporal Semi-supervision." *International Journal of Computer Vision* (2023), pp. 1–17 (pages 11, 23).

[43] Kevin Luxem et al. "Identifying behavioral structure from deep variational embeddings of animal motion." *Communications Biology* 5.1 (2022), p. 1267 (page 2).

[44] Alexander Mathis et al. "DeepLabCut: markerless pose estimation of user-defined body parts with deep learning." *Nature neuroscience* 21.9 (2018), pp. 1281–1289 (pages 2, 4, 27, 32, 33).

[45] Mackenzie Weygandt Mathis and Alexander Mathis. "Deep learning tools for the measurement of animal behavior in neuroscience." *Current opinion in neurobiology* 60 (2020), pp. 1–11 (pages 7, 23).

[46] John P Miller et al. “Accuracy on the line: on the strong correlation between out-of-distribution and in-distribution generalization.” *International Conference on Machine Learning*. PMLR. 2021, pp. 7721–7735 (page 5).

[47] Arne Monsees et al. “Estimation of skeletal kinematics in freely moving rodents.” *Nature Methods* 19.11 (2022), pp. 1500–1509 (pages 2, 4, 23, 38).

[48] Simon Musall et al. “Single-trial neural dynamics are dominated by richly varied movements.” *Nature neuroscience* 22.10 (2019), pp. 1677–1686 (page 2).

[49] Tanmay Nath et al. “Using DeepLabCut for 3D markerless pose estimation across species and behaviors.” *Nature protocols* 14.7 (2019), pp. 2152–2176 (page 4).

[50] Simon RO Nilsson et al. “Simple Behavioral Analysis (SimBA)–an open source toolkit for computer classification of complex social behaviors in experimental animals.” *BioRxiv* (2020), pp. 2020–04 (pages 2, 11).

[51] Yael Niv. “The primacy of behavioral research for understanding the brain.” *Behavioral Neuroscience* 135.5 (2021), p. 601 (page 2).

[52] Ian Osband, John Aslanides, and Albin Cassirer. “Randomized prior functions for deep reinforcement learning.” *Advances in Neural Information Processing Systems* 31 (2018) (page 22).

[53] Ian Osband et al. “Deep exploration via bootstrapped DQN.” *Advances in neural information processing systems* 29 (2016) (page 16).

[54] Yaniv Ovadia et al. “Can you trust your model’s uncertainty? evaluating predictive uncertainty under dataset shift.” *Advances in neural information processing systems* 32 (2019) (page 16).

[55] Nancy Padilla-Coreano et al. “Cortical ensembles orchestrate social competition through hypothalamic outputs.” *Nature* 603.7902 (2022), pp. 667–671 (page 2).

[56] Liam Paninski et al. “A New Look at State-Space Models for Neural Data.” *J. Comput. Neurosci.* 29.1–2 (2010), pp. 107–126 (page 37).

[57] Adam Paszke et al. “Pytorch: An imperative style, high-performance deep learning library.” *Advances in neural information processing systems* 32 (2019) (pages 20, 27).

[58] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python.” *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (page 38).

[59] Talmo D Pereira et al. “Fast animal pose estimation using deep neural networks.” *Nature methods* 16.1 (2019), pp. 117–125 (pages 2, 4, 27).

[60] Talmo D Pereira et al. “SLEAP: A deep learning system for multi-animal pose tracking.” *Nature methods* 19.4 (2022), pp. 486–495 (pages 2, 6, 23, 26, 27, 38).

[61] Benjamin Recht et al. “Do imagenet classifiers generalize to imagenet?” *International conference on machine learning*. PMLR. 2019, pp. 5389–5400 (page 5).

[62] Chris C Rodgers. “A detailed behavioral, videographic, and neural dataset on object recognition in mice.” *Scientific Data* 9.1 (2022), p. 620 (pages 2, 11).

[63] Britton A Sauerbrei et al. “Cortical pattern generation during dexterous movement is input-driven.” *Nature* 577.7790 (2020), pp. 386–391 (page 2).

[64] Shreya Saxena et al. “Localized semi-nonnegative matrix factorization (LocaNMF) of widefield calcium imaging data.” *PLoS computational biology* 16.4 (2020), e1007791 (page 2).

[65] Artur Schneider et al. “3D pose estimation enables virtual head fixation in freely moving rats.” *Neuron* 110.13 (2022), pp. 2080–2093 (pages 2, 23).

[66] Cristina Segalin et al. “The Mouse Action Recognition System (MARS) software pipeline for automated analysis of social behaviors in mice.” *Elife* 10 (2021), e63720 (pages 2, 11).

[67] Elizabeth S Spelke. “Principles of object perception.” *Cognitive science* 14.1 (1990), pp. 29–56 (page 6).

[68] Greg J Stephens et al. “Dimensionality and dynamics in the behavior of C. elegans.” *PLoS computational biology* 4.4 (2008), e1000028 (page 31).

[69] Greg J Stephens et al. “From modes to movement in the behavior of Caenorhabditis elegans.” *PloS one* 5.11 (2010), e13914 (page 8).

[70] Carsen Stringer et al. “Spontaneous behaviors drive multidimensional, brainwide activity.” *Science* 364.6437 (2019), eaav7893 (page 2).

[71] Jennifer J Sun et al. “BKinD-3D: Self-Supervised 3D Keypoint Discovery from Multi-View Videos.” *arXiv preprint arXiv:2212.07401* (2022) (pages 22, 23).

[72] Jennifer J Sun et al. “Self-supervised keypoint discovery in behavioral videos.” *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 2171–2180 (page 22).

[73] Atika Syeda et al. “Facemap: a framework for modeling neural activity based on orofacial tracking.” *bioRxiv* (2022) (pages 6, 7, 11, 37).

[74] The International Brain Laboratory. “Data release - Brainwide map - Q4 2022” (Jan. 2023). DOI: `10.6084/m9.figshare.21400815.v6`. URL: `https://figshare.com/articles/preprint/Data_release_-_Brainwide_map_-_Q4_2022/21400815` (pages 4, 5, 16, 17, 25, 39).

[75] The International Brain Laboratory et al. “Standardized and reproducible measurement of decision-making in mice.” *Elife* 10 (2021), e63711 (page 25).

[76] The International Brain Laboratory et al. “Reproducibility of in-vivo electrophysiological measurements in mice.” *bioRxiv* (2022) (page 26).

[77] The International Brain Laboratory et al. “Video hardware and software for the International Brain Laboratory.” en. *figshare* (2022). DOI: `10.6084/m9.figshare.19694452` (page 26).

[78] Antonio Torralba and Alexei A Efros. “Unbiased look at dataset bias.” *CVPR 2011*. IEEE. 2011, pp. 1521–1528 (page 5).

[79] Dustin Tran et al. “Plex: Towards reliability using pretrained large model extensions.” *arXiv preprint arXiv:2207.07411* (2022) (page 5).

[80] Matthew C Tresch and Anthony Jarc. “The case for and against muscle synergies.” *Current opinion in neurobiology* 19.6 (2009), pp. 601–607 (page 31).

[81] Richard A Warren et al. “A rapid whisker-based decision underlying skilled locomotion in mice.” *Elife* 10 (2021), e63596 (pages 2, 4, 5, 7, 11, 13, 16, 25, 37).

[82] Caleb Weinreb et al. “Keypoint-MoSeq: parsing behavior by linking point tracking to pose dynamics.” *bioRxiv* (2023), pp. 2023–03 (pages 2, 11, 37).

[83] Matthew R Whiteway et al. “Partitioning variability in animal behavioral videos using semi-supervised variational autoencoders.” *PLoS computational biology* 17.9 (2021), e1009439 (pages 11, 23).

[84] Alexander B Wiltschko et al. “Mapping sub-second structure in mouse behavior.” *Neuron* 88.6 (2015), pp. 1121–1135 (page 2).

[85] Alexander B Wiltschko et al. “Revealing the structure of pharmacobehavioral space through motion sequencing.” *Nature neuroscience* 23.11 (2020), pp. 1433–1443 (page 2).

[86] Anqi Wu et al. “Deep Graph Pose: a semi-supervised deep graphical model for improved animal pose tracking.” *Advances in Neural Information Processing Systems* 33 (2020), pp. 6040–6052 (pages 7, 22, 23, 27).

[87] Omry Yadan. “Hydra-a framework for elegantly configuring complex applications.” *Github* 2 (2019), p. 5 (page 20).

[88] Yuke Yan et al. “Unexpected complexity of everyday manual behaviors.” *Nature communications* 11.1 (2020), pp. 1–8 (pages 8, 31).

[89] Shaokai Ye, Alexander Mathis, and Mackenzie Weygandt Mathis. “Panoptic animal pose estimators are zero-shot performers.” *arXiv preprint arXiv:2203.07436* (2022) (page 23).

[90] Hang Yu et al. “Ap-10k: A benchmark for animal pose estimation in the wild.” *arXiv preprint arXiv:2108.12617* (2021) (pages 14, 27).

[91] Libby Zhang et al. “Animal pose estimation from video data with a hierarchical von Mises-Fisher-Gaussian model.” *International Conference on Artificial Intelligence and Statistics*. PMLR. 2021, pp. 2800–2808 (pages 4, 7, 23, 37, 38).

[92] Yilun Zhang and Hyun Soo Park. “Multiview supervision by registration.” *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2020, pp. 420–428 (pages 7, 22, 30).

[93] Mahdi Zolnouri, Xinlin Li, and Vahid Partovi Nia. “Importance of data loading pipeline in training deep neural networks.” *arXiv preprint arXiv:2005.02130* (2020) (page 19).

6 Supplemental figures

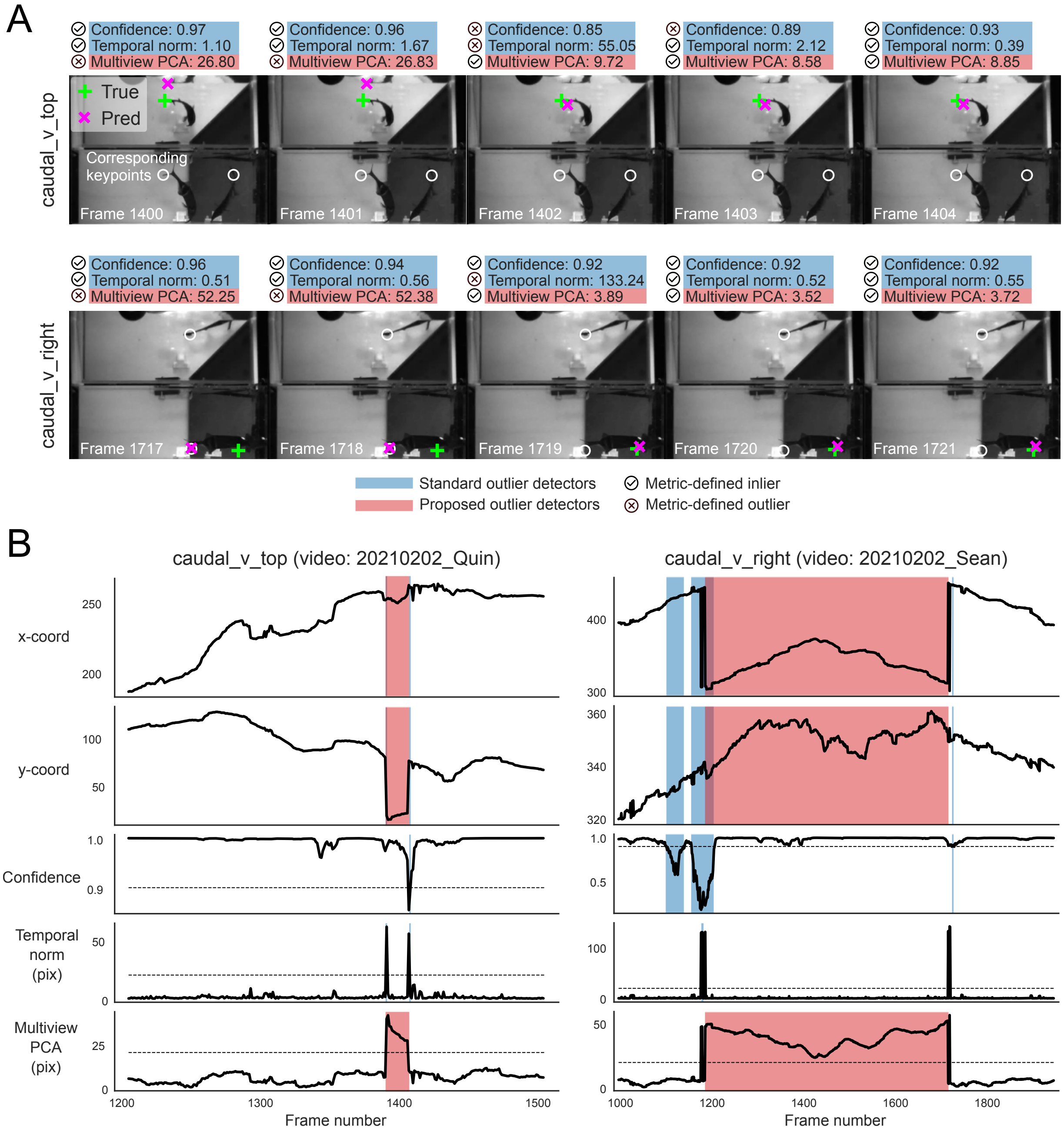


Figure S1: **Unsupervised losses complement model confidence for outlier detection on mirror-fish dataset.** Example traces, unsupervised metrics, and predictions from a DLC model (trained on 268 frames) on held-out videos. Conventions as in Fig. 3. In the first row of A, note that the initial predictions lie on a reflection from the underside of the water with high confidence, but are flagged as problematic by the multi-view PCA loss. In the second row of A, the initial predictions lie on the same body part as seen in another view, again with high confidence. The multi-view PCA loss flags these erroneous predictions.

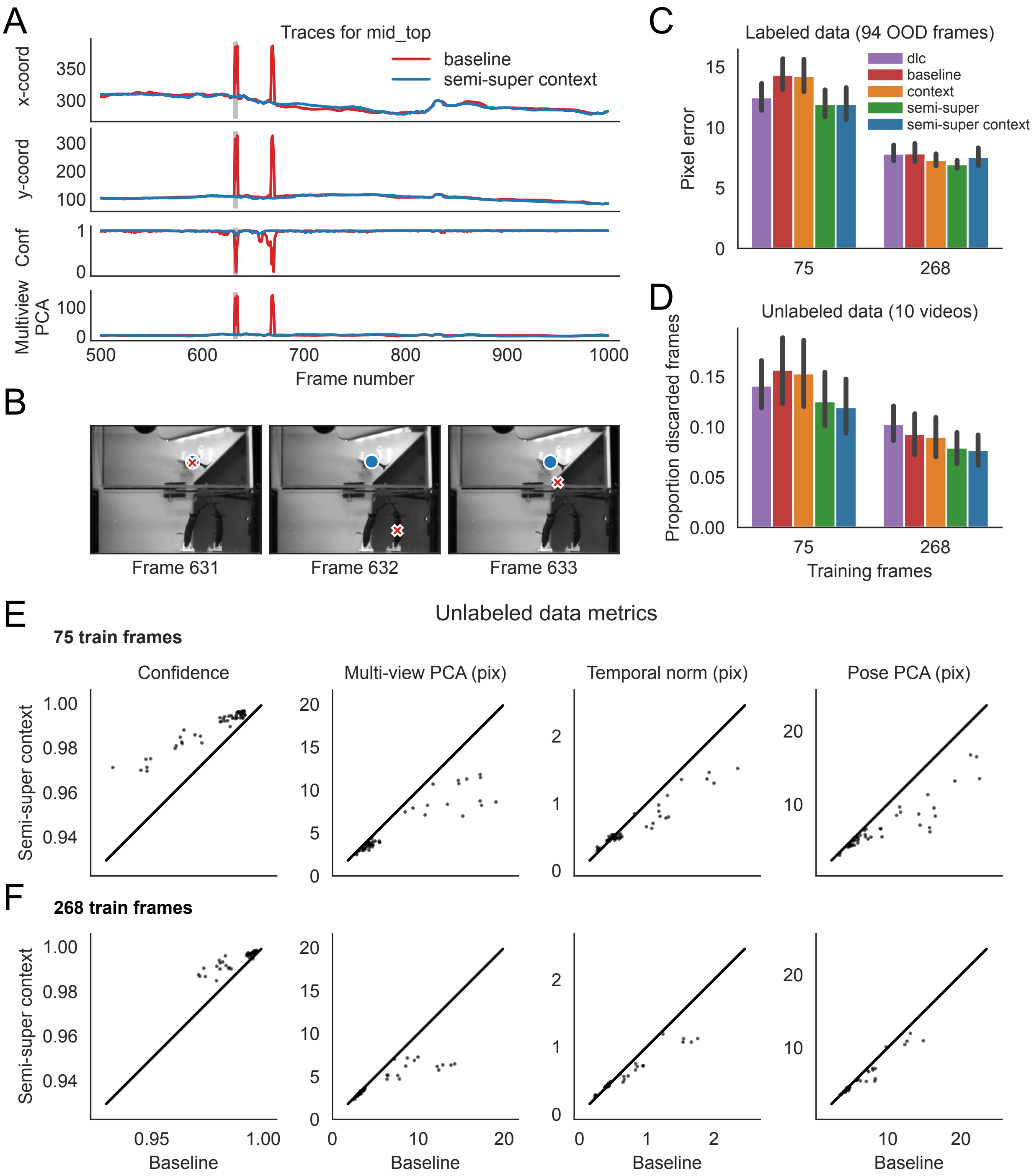


Figure S2: **Unlabeled frames improve pose estimation in mirror-fish dataset.** Conventions as in Fig. 4. See Fig. V1 for link to video corresponding to panel A.

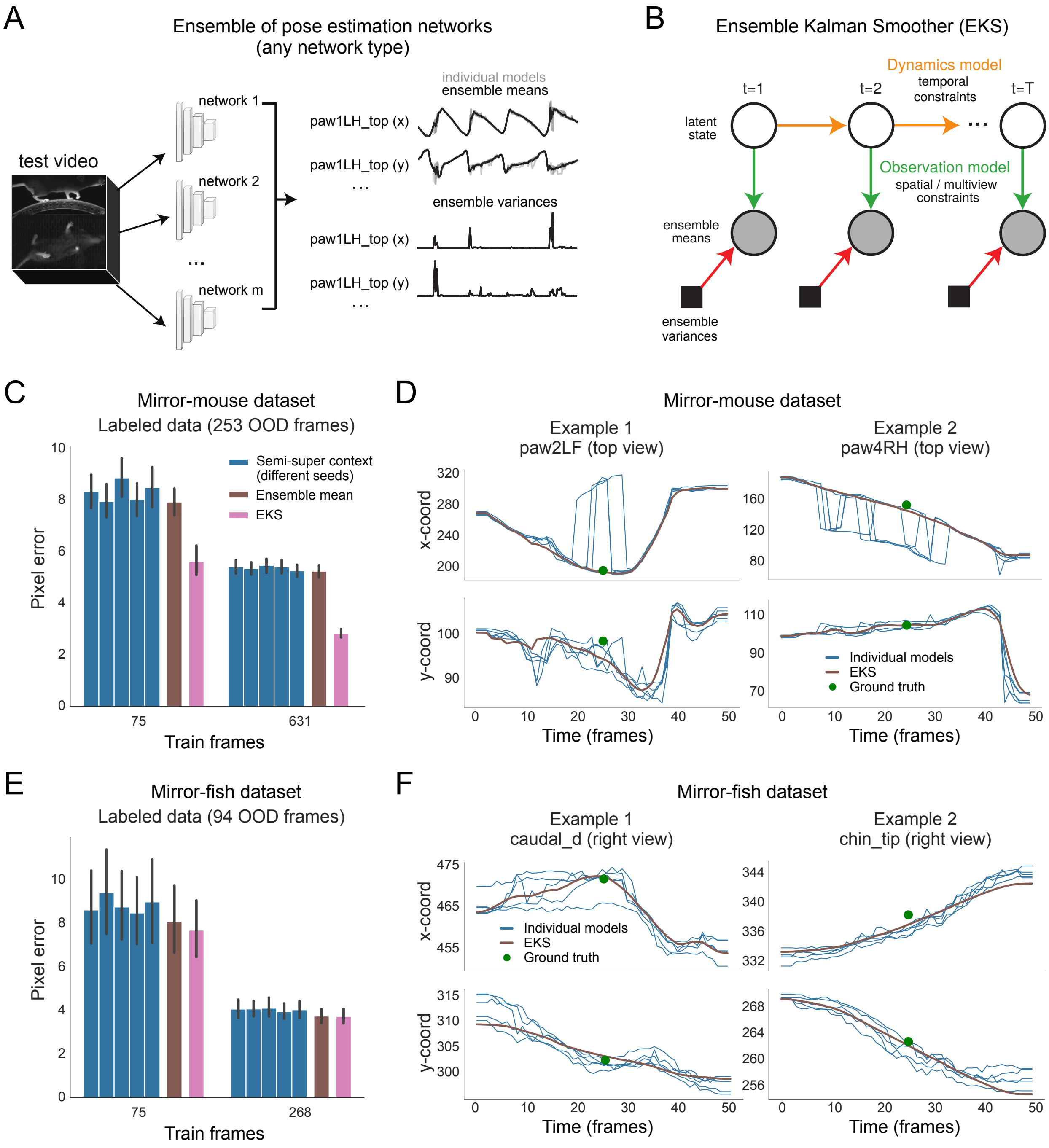
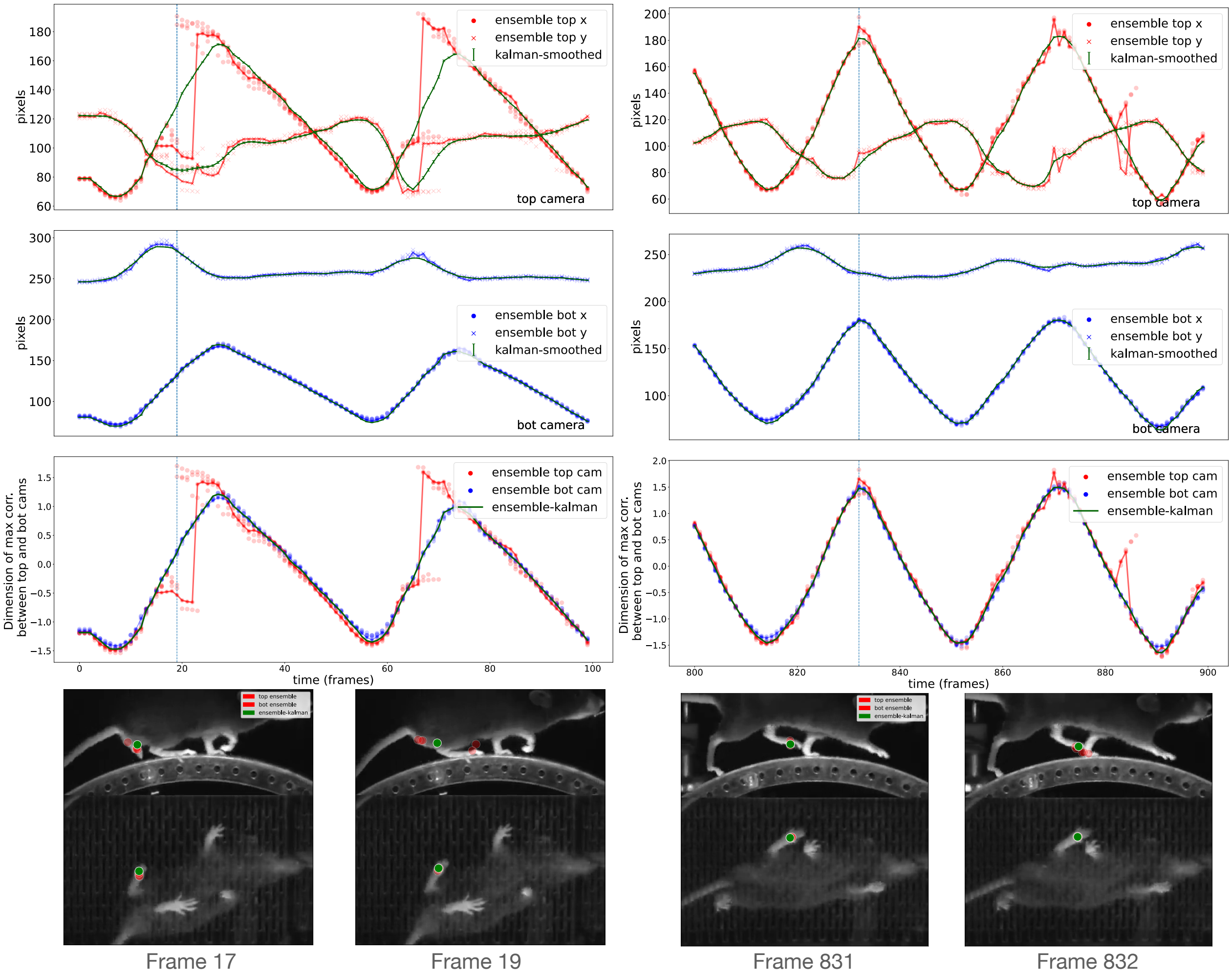


Figure S3: **Ensemble Kalman Smoother improves pose estimation.** See caption on following page.

Figure S3: **Ensemble Kalman Smoother improves pose estimation.** **A**. *Deep ensembling* is a process that combines the predictions of multiple networks trained from different random initializations, or using different subsets of data, or even using different architectures. The *ensemble mean* is potentially more accurate than single model predictions, and the *ensemble variance* (computed across individual models for a given frame and keypoint) can be a useful measure of uncertainty that is complementary to single model confidence values. **B**. The Ensemble Kalman Smoother (EKS) is our proposed post-processing approach that leverages both the spatiotemporal constraints introduced by the unsupervised losses as well as uncertainty measures from the ensemble variance in a probabilistic state-space model. The ensemble means of the keypoints are modeled with a latent linear dynamical system; the temporal smoothness constraints are enforced through the linear dynamics (yellow arrows) and the spatial constraints (Pose PCA or multi-view PCA) are enforced through a fixed observation model that maps the latent state to the observations (green arrows). Instead of learning the observation noise, we employ the time-varying ensemble variance (red arrows). EKS predictions use a Bayesian approach to weight the relative contributions from the prior and the observations. **C**. Pixel error over OOD frames (paw keypoints only for simplicity) for 5 semi-supervised context models trained with the same data but using different random initializations of the head, and data serving order (blue bars); the ensemble mean of these 5 networks (brown bars); and the EKS prediction (pink bars). The ensemble mean by itself does not lead to noticeable improvements, but the smoothing provided by EKS provides strong performance increases. **D**. Two example traces that show predictions from individual semi-supervised context models (75 training frames; gray lines) and EKS (black lines), as well as ground truth labels (green dots). **E,F**. EKS results for the mirror-fish dataset.



(a) Paw occlusion

(b) Paw switching

Figure S4: **Ensemble Kalman smoothing example with the 75 training frame mirror-mouse dataset.** All ensemble model predictions are from five of our best semi-supervised temporal context networks. (As noted in Fig. 4, the baseline and DLC models have significantly worse performance in this setting.) The two columns show two illustrative examples. Top panels: x and y coordinates of the left hind paw viewed on the top camera. Conventions as in Fig. S5. Second from top: x and y coordinates of the left paw viewed on the bottom camera. Third from top: CCA coordinates computed from the top and bottom camera views. Similarly to the IBL-paw dataset (Fig. S6), these CCA coordinates should be equal at each frame. The top view is more challenging (the camera is facing the side of the mouse rather than the bottom, and we are tracking the distant paw here, so more occlusions occur), and the ensemble variance is correspondingly larger for the top view; therefore the ensemble-KS tracks the bottom view more closely here. Bottom: example frames (indicated with the vertical dashed line above). In the left column, we see an example of paw occlusion - when the left hind paw goes behind the back of the animal all members of the ensemble jump to the nearest visible keypoint. Tracking is accurate in frame 17 and then the occlusion and ensemble confusion is visible in frame 19. Note that the ensemble-KS accurately tracks the correct paw here, since it uses information from the more confident camera view to resolve confusion in the more challenging camera view. In the right column, we see an example in which the ensemble-KS is able to correct an error due to paw switching in the indicated frames (831-2). For the full video used here, see Section 7.

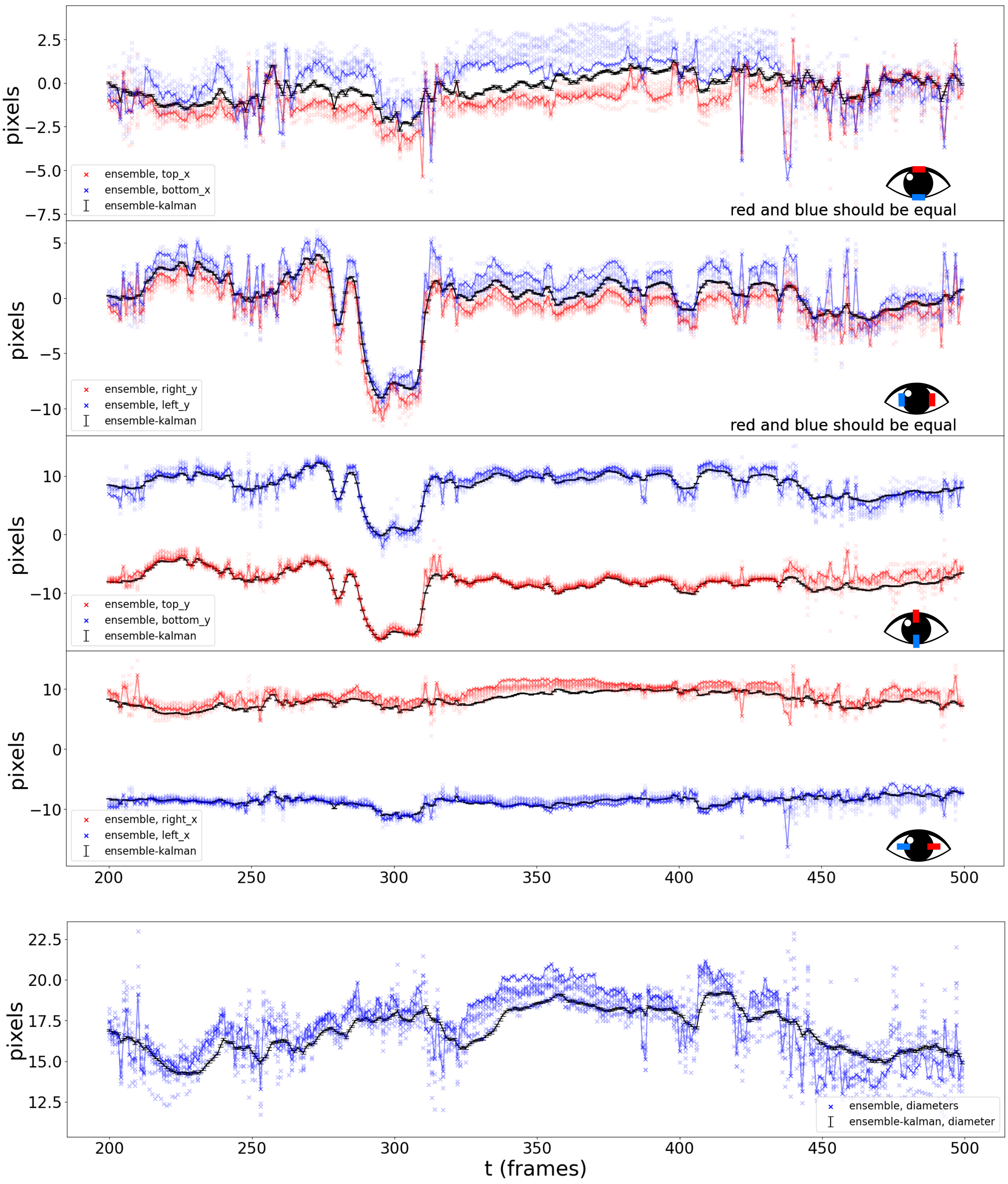
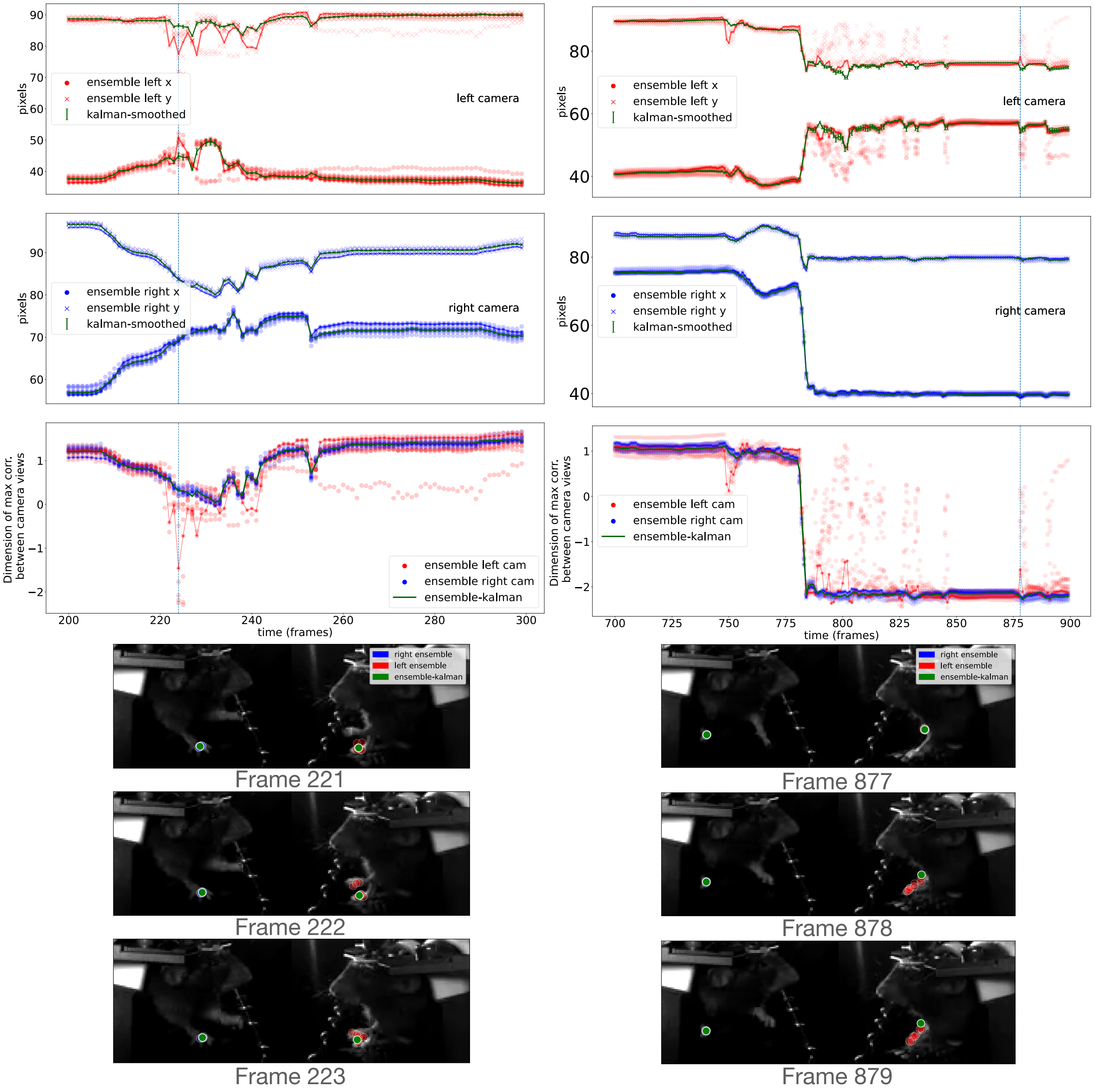


Figure S5: **Ensemble Kalman smoothing example with pupil traces**. The top four panels show the eight tracked coordinates, while the bottom panel shows the estimated diameter. In each panel, the ×'s indicate the output of individual ensemble members (baseline fully-supervised networks here); we have connected the output of a single ensemble member with a solid line across time. The black trace indicates the ensemble-Kalman-smoothed output. As the pupil keypoints are arranged in a diamond shape, they are paired naturally, e.g. the x coordinate of the top and bottom keypoints should be equal, as should the y coordinate of the left and right keypoint (as indicated by the insets at the bottom right of each of the first four panels). This pairing makes it easy to detect errors: for example, in the first and second panels, any divergence between the red and blue traces are due to tracking errors. Note that tracking errors are common here but are largely resolved by the ensemble Kalman smoother. For the full video used in this supplement, see Section 7.



(a) Paw switching

(b) Paw occlusion

Figure S6: **Ensemble Kalman smoothing of baseline supervised models improves pose estimation on IBL paw data.** The two columns show two illustrative examples. Top panels: x and y coordinates of the left paw viewed on the left camera. Conventions as in Fig. S5. Second from top: x and y coordinates of the left paw viewed on the right camera. Third from top: canonical correlation analysis (CCA) coordinates computed from the left and right camera views (see Section 5.11 for details). Due to the geometry of multiple cameras recording the same body parts from different directions, these CCA coordinates should be equal at each frame (this is true for the ensemble-KS by construction). The left view is more challenging (the paw is further from the camera and the sampling rate of the video is lower), and the ensemble variance is correspondingly larger for the left view; therefore the ensemble-KS tracks the right view more closely here. Bottom: example frames (indicated with the vertical dashed line above). In the left column, we see an example of paw confusion - some members of the ensemble (correctly) track one paw, and some (mistakenly) track the other. Tracking is accurate in frame 221 and then confusion between the two paws is visible in frame 222. Note that the ensemble-KS accurately tracks the correct paw here, since it uses information from the more confident camera view to resolve confusion in the more challenging camera view. In the right column, we see an example in which the ensemble-KS is able to correct an error due to paw occlusion in the indicated frames (878-9). The right ensemble is highly confidence so it is tightly packed behind the ensemble-kalman prediction. For the full video used here, see Section 7.

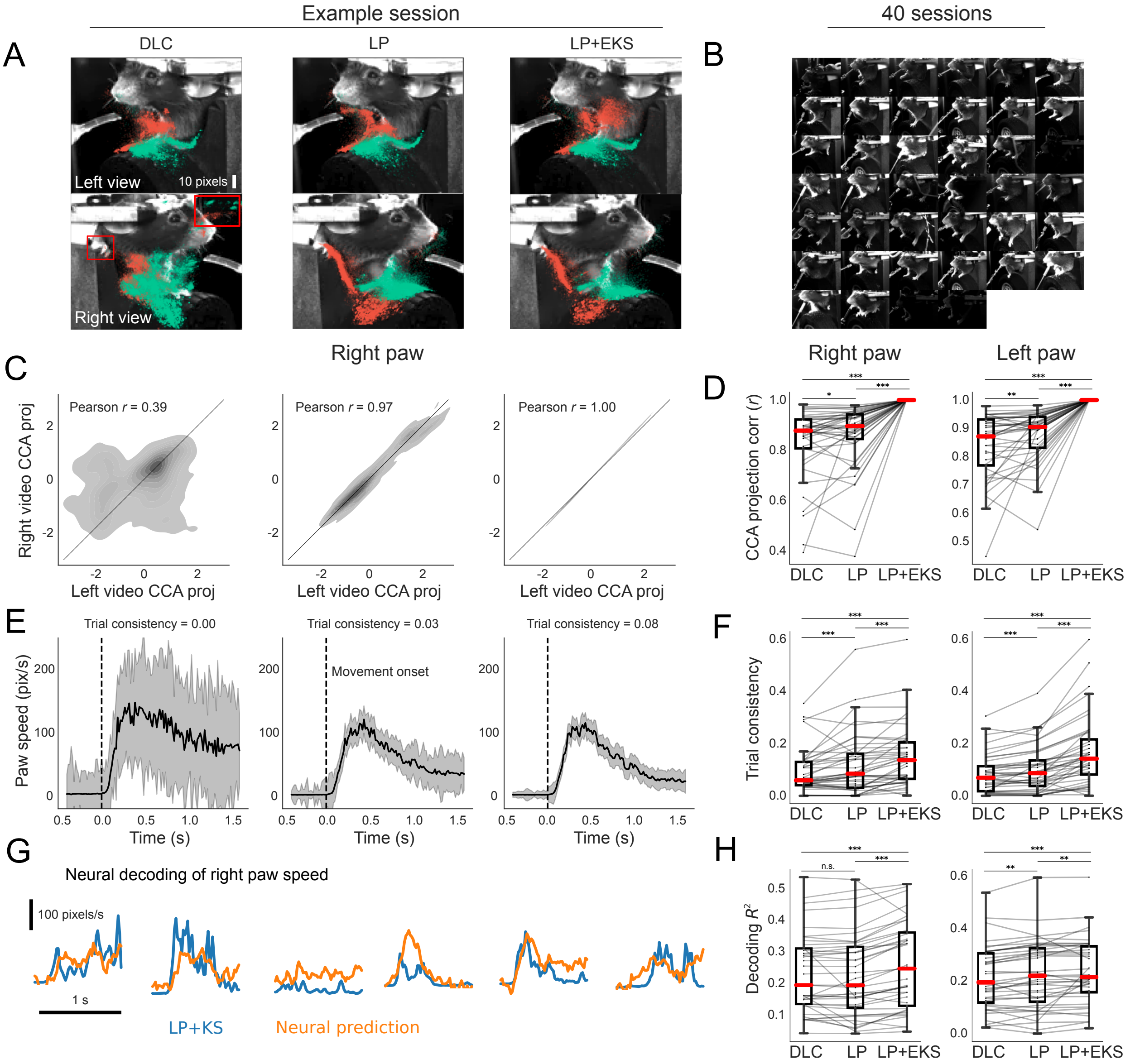


Figure S7: **Lightning Pose models and ensemble smoothing improve pose estimation on IBL paw data.** **A**. Sample frames from each camera view overlaid with a subset of paw markers estimated from DLC (*left*), semi-supervised context Lightning Pose model (*center*), and ensembled and Kalman-smoothed Lightning Pose (*right*); note erroneous DLC predictions near the top of the frame, and lack of predictions on the paw (red boxes). **B**. Example left view frames from 40 IBL sessions, illustrating the diversity of imaging conditions in the dataset. **C**. As discussed in Fig. S6, the right paw position in the right view should be highly correlated with right paw position in the left view; the 1D subspace of maximal correlation is found with canonical correlation analysis (CCA). Panel shows the empirical distribution of the right paw position projected onto this dimension from each view. Column arrangement as in A. Note the correlation is greatly increased in LP compared to DLC; the LP+EKS model imposes a low-dimensional model that enforces perfectly correlated projections, by construction. **D**. Correlation in the CCA subspace is computed across 40 sessions for each model and paw. The right paw correlation is generally high for DLC (r=0.83±0.02, mean±sem) and is slightly improved by the LP model (0.86±0.02). Again, the LP+EKS model has a correlation of 1.0 (±0.00) by construction. Similar results hold for left paw (DLC: 0.84±0.02; LP: 0.88±0.01). **E**. Median right paw speed plotted across correct trials aligned to first movement onset of the wheel; error bars show 95% confidence interval. The same trial consistency metric from Fig. 5 is computed. See Fig. V3 for a link to the corresponding video. **F**. DLC has the most variable traces across sessions per the trial consistency metric for right paw (0.09±0.01), which is improved by the LP model (0.12±0.02) and again by LP+EKS (0.15±0.02). Similar conclusions hold for left paw (DLC: 0.08±0.01; LP: 0.10±0.01; LP+EKS: 0.16±0.02) **G**. Example traces of Kalman smoothed right paw speed (blue) and predictions from neural activity (orange) for several trials using cross-validated, regularized linear regression (Methods). **H**. DLC decoding performance of the right paw across sessions (R^2=0.23±0.02) is similar to that of the LP model (0.22±0.02) and slightly improved by LP+EKS (0.25±0.02). Similar conclusions hold for left paw (DLC: 0.21±0.02; LP: 0.23±0.02; LP+EKS: 0.24±0.02).

7 Supplemental videos

The figures in this work are accompanied by various videos demonstrating the performance of the pose estimators and the downstream analyses. All videos can be found at this link. In detail, these videos include:

- Baseline vs semi-supervised context model predictions (Fig. V1)
- Ensemble Kalman model predictions (Fig. V2)
- Trial-by-trial markers and traces for IBL datasets (Fig. V3)

The following figures present a still frame from each of these video types, as well as links to each of the videos.

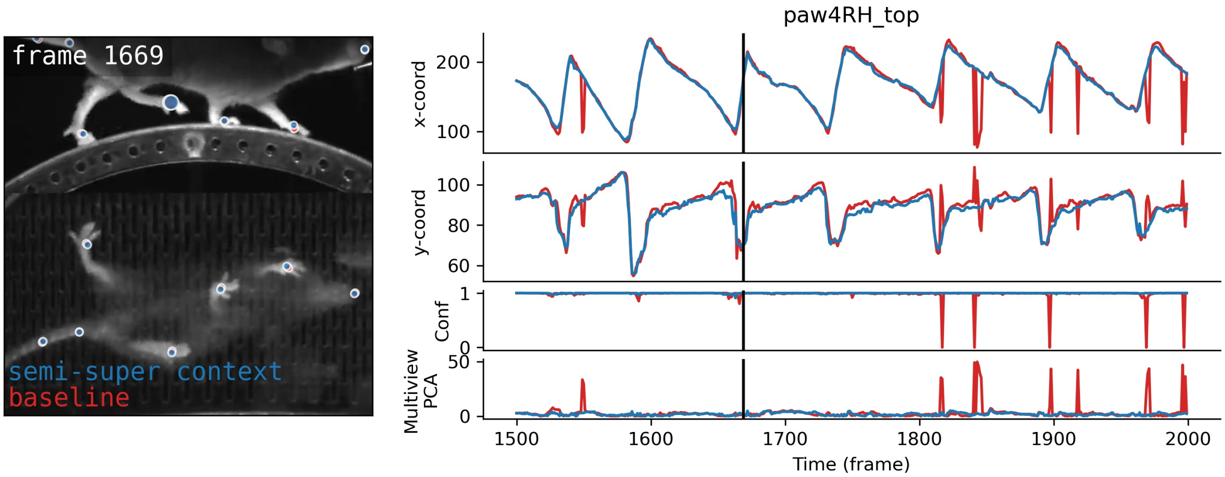


Figure V1: **Baseline vs semi-supervised context model predictions.** *Left*: Model predictions (with no confidence filtering) for a snippet of OOD video, for both the fully supervised baseline model (red) and the semi-supervised context model (blue). The larger marker is the one depicted in the traces. *Right*: x- and y-coordinates of the larger marker, along with confidence values and Multi-view PCA loss. Vertical black bar marks the current frame in the video.

7.1 Model predictions

- mirror-mouse dataset, DLC model trained with 631 labeled frames, corresponding to traces in Fig. 1B [link]
- mirror-mouse dataset, baseline/semi-supervised context models trained with 75 labeled frames, corresponding to traces in Fig. 4A [link]
- mirror-fish dataset, baseline/semi-supervised context models trained with 75 labeled frames, corresponding to traces in Fig. S2A [link]
- mirror-fish dataset, baseline/semi-supervised context models trained with 268 labeled frames [link]

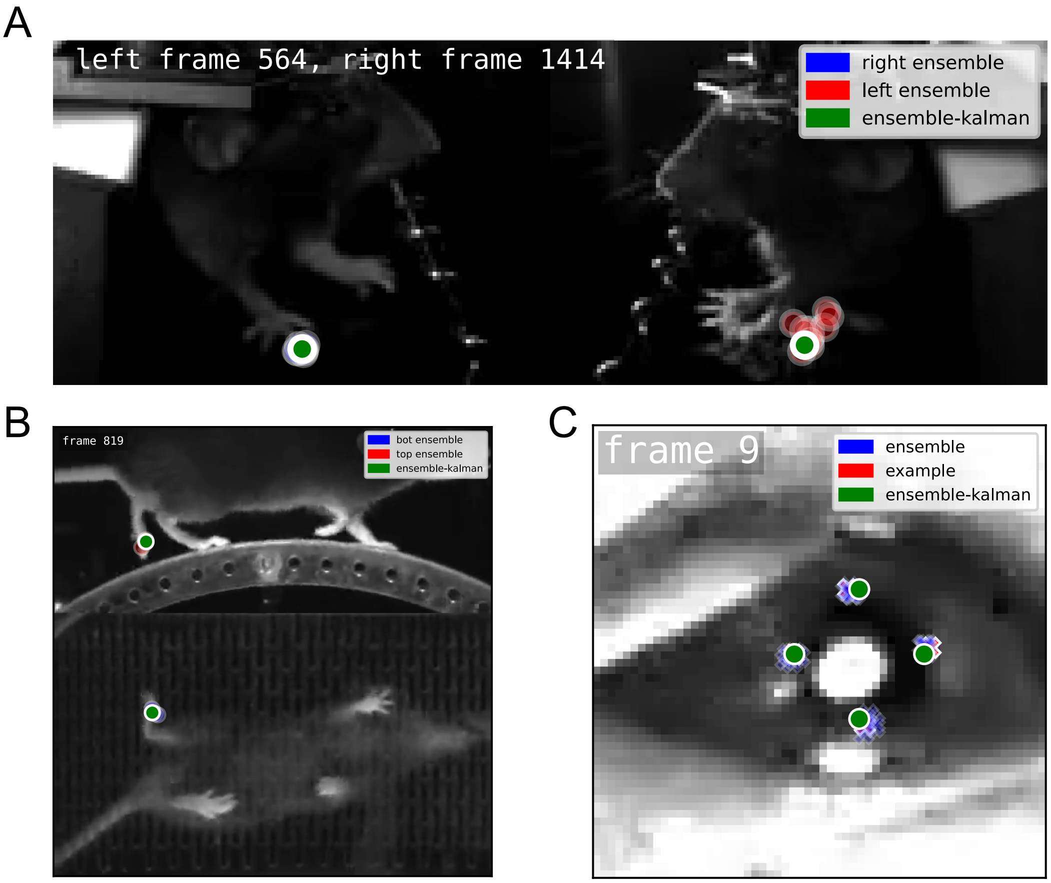


Figure V2: **Ensemble kalman model predictions.** **A**: Video frames for IBL paw dataset from two cameras (right and left) overlaid with predicted markers from ten supervised baseline models (blue and red) and the ensemble Kalman smoother (green). The cameras for this dataset have different sampling frequencies so the closest frames are visualized together for this video. All blue points are closely packed together behind the green point. **B**: Video frames for 75 frame mirror-mouse dataset with two camera views (top and bottom) overlaid with predicted markers from five semi-supervsied baseline models (blue and red) and the ensemble Kalman smoother (green). **C**: Video frame for IBL pupil dataset with predicted markers from ten supervised baseline models (blue), one example supervised baseline model (red), and the ensemble Kalman smoother (green).

7.2 Ensemble kalman model predictions

- mirror-mouse video markers, corresponding to session in Fig. S4 [link]
- pupil video markers, corresponding to session in Fig. S5 [link]
- paw video markers, corresponding to session in Fig. S6 [link]

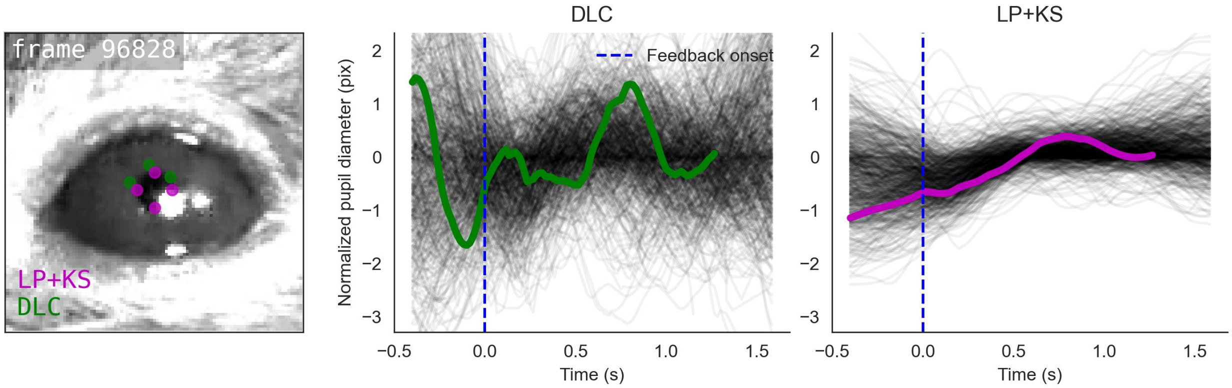


Figure V3: **Trial-by-trial markers and traces for IBL datasets. Left**: Video frame overlaid with predicted markers from the DLC model (green) and the ensemble Kalman smoother (LP+KS; magenta). Markers with confidence <0.9 are not displayed. **Center**: Black traces show smoothed DLC pupil diameter from a subset of correct trials, aligned to feedback onset (reward delivery). The green trace highlights the pupil diameter of the current trial displayed on the left. **Right**: Pupil diameters from the LP+KS model.

7.3 Trial-by-trial markers and traces for IBL datasets

- pupil diameter, corresponding to session in Fig. 5C,E,G [link]
- paw speed, corresponding to session in Supplementary Fig. S7A,C,E [link]