Real-time, low-latency closed-loop feedback using markerless posture tracking

Gary Kane[1], **Gonçalo Lopes**[2], **Jonny L. Saunders**[3], **Alexander Mathis**[1,4], **& Mackenzie W. Mathis**[1,4,*]

[1]The Rowland Institute at Harvard, Harvard University, Cambridge, MA USA
[2]NeuroGEARS Ltd, London, UK
[3]Institute of Neuroscience, Department of Psychology, University of Oregon, Eugene, OR USA
[4]Center for Neuroprosthetics, Center for Intelligent Systems, & Brain Mind Institute, School of Life Sciences, Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland
[*]mackenzie@post.harvard.edu

The ability to control a behavioral task or stimulate neural activity based on animal behavior in real-time is an important tool for experimental neuroscientists. Ideally, such tools are (1) noninvasive, (2) low-latency, and (3) provide interfaces to trigger external hardware based on posture (i.e., not just object-based-tracking). Recent advances in pose estimation with deep learning allows researchers to train deep neural networks to accurately quantify a wide variety of animal behaviors. In extending our efforts towards the animal pose estimation toolbox DeepLabCut, here, we provide a new `DeepLabCut-Live!` package that achieves low-latency real-time pose estimation (within 15 ms, at >100 FPS), with an additional forward-prediction module that achieves zero-latency feedback. We also provide three options for using this tool with ease: a stand-alone GUI (called `DLC-Live! GUI`), integration into `Bonsai` and into `AutoPilot`. Lastly, we benchmarked performance on a wide range of systems so that experimentalists can easily decide what hardware is required for their needs.

Highlights:

1. The DeepLabCut-Live! package is available via `pip install deeplabcut-live`
2. The Bonsai-DLC plugin is available
3. The AutoPilot-DLC plugin is available
4. The DeepLabCut-Live! GUI package is available via `pip install deeplabcut-live-gui`

Introduction

In recent years, advances in deep learning have fueled sophisticated behavioral analysis tools (1–7). Specifically, advances in animal pose estimation have ushered in an era of high-throughput quantitative analysis of movements (8). One such state-of-the-art animal pose estimation package, DeepLabCut (DLC, 4), provides tailored networks that predict the posture of animals of interest based on video frames, and can run swiftly in offline batch processing modes (up to 2,500 FPS on standard GPUs; 9, 10). This high throughput analysis has proven to be an invaluable tool to probe the neural mechanisms of behavior (8, 11).

The ability to apply these behavioral analysis tools to provide feedback to animals in real time is crucial for causally testing the behavioral functions of specific neural circuits.

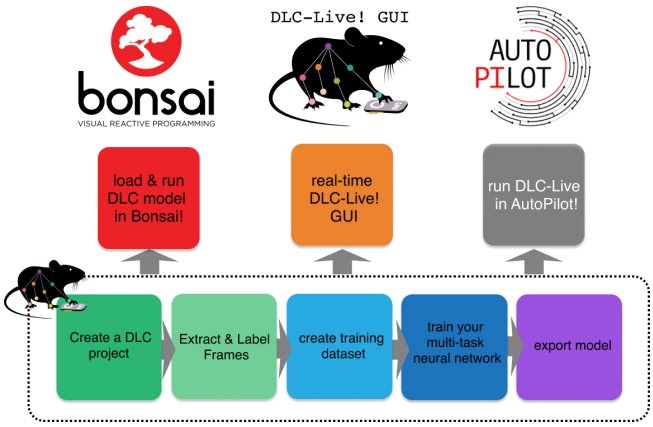

Figure. 1. Overview of using DLC networks in real-time experiments within Bonsai, the DLC-Live! GUI, and AutoPilot.

This paper describes a series of new software tools that can achieve low-latency closed-loop feedback based on animal pose estimation with DeepLabCut. Additionally, these tools make it easier for experimentalists to design and conduct experiments with little to no additional programming, to integrate real-time pose estimation with DeepLabCut into custom software applications, and to share previously trained DeepLabCut neural networks with other users. First, we provide a marked speed and latency improvement from existing real-time pose estimation software (12–14) by optimizing inference code, using lightweight DeepLabCut models that perform well not only on GPUs, but also on CPUs and affordable embedded systems such as the NVIDIA Jetson platform. Second, we introduce a module to export trained neural network models and load them into other platforms with ease, improving the ability to transfer trained models between machines, to share trained models with other users, and to load trained models in other software packages. Easy loading of trained models into other software packages enabled integration of DeepLabCut into another popular systems neuroscience software, Bonsai (15). Third, we provide a new lightweight `DeepLabCut-Live!` package to run DeepLabCut inference online (or offline). This package has minimal software dependencies and can easily be installed on integrated systems, such as the NVIDIA Jetson platform. Furthermore, it is designed to enable easy integration of real-time pose estimation using DeepLabCut into custom software applications. We demonstrate this ability via

integration of DeepLabCut-Live! into the new AutoPilot framework (16).

Using these new software tools, we achieve low latency real-time pose estimation, with delays as low as 10 ms using GPUs and 30 ms using CPUs. Furthermore, we introduce a forward-prediction module that counteracts these delays by predicting the animal's future pose. Using this forward-prediction module, we were able to provide ultra-low latency feedback to an animal (even down to sub-zero ms delay). Such short latencies have only been approachable in marked animals (17), but have not been achieved to the best of our knowledge previously with markerless pose estimation (12–14, 18, 19).

Lastly, we developed a benchmarking suite to test the performance of these new tools on multiple hardware and software platforms. We provide performance metrics for 10 different GPUs, 2 integrated systems and 5 CPUs across different operation systems. We openly share this benchmarking suite[1], so that users can look up expected inference speeds and run the benchmark on their system. We believe that with more user contributions this will allow the community to comprehensively summarize system performance for different hardware options and can thus guide users in choosing GPUs, integrated systems, and other options of interest for their particular use case.

Results

Exporting DLC models.

DeepLabCut enables the creating of tailored neuronal networks for pose estimation of user-defined bodyparts (4, 20). We sought to make these neural networks, which are TensorFlow graphs, easily deployable by developing a model-export functionality. These customized DeepLabCut models can be created from standard trained DeepLabCut models by running the export_model function within DeepLabCut (see Methods), or models can be downloaded from the new DeepLabCut Model Zoo.

The model export module utilizes the protocol buffer format (.pb file). Protocol buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data[2], which makes sharing models simple. Sharing a whole (DeepLabCut) project is not necessary, and an end-user can simply point to the protocol buffer file of a model to run inference on novel videos (online or offline). The flexibility offered by the protocol buffer format allowed us to integrate DeepLabCut into applications written in different languages: a new python package DeepLabCut-Live!, which facilitates loading DeepLabCut networks to run inference; and into Bonsai, which is written in C# and runs DeepLabCut inference using TensorFlowSharp[3].

[1] https://github.com/DeepLabCut/DeepLabCut-live
[2] https://developers.google.com/protocol-buffers
[3] https://github.com/migueldeicaza/TensorFlowSharp

A new python package to develop real-time pose estimation applications.

The DeepLabCut-Live! package provides a simple programming interface to load trained/exported DeepLabCut models and perform pose estimation on single images (i.e., from a camera feed). By design this package has minimal dependencies and can be easily installed even on small integrated systems.

To use the DeepLabCut-Live! package to perform pose estimation, experimenters must simply start with a trained DeepLabCut model in the exported protocol buffer format (.pb file) and instantiate a DLCLive object. This object can be used to load the DeepLabCut network and perform pose estimation on single images:

```
from dlclive import DLCLive
my_live_object =
    DLCLive("/exportedmodel/directory")
my_live_object.init_inference(my_image)
pose = my_live_object.get_pose(my_image)
```

On its own, the DLCLive class only enables experimenters to perform real-time pose estimation. To use poses estimated by DeepLabCut to provide closed-loop feedback, the DeepLabCut-live package uses a Processor class. A Processor class must contain two methods, process and save. The process method takes a pose as an argument, performs some operation, such as giving a command to control external hardware (e.g. to give reward or to turn on a laser for optogenetic stimulation), and returns a *processed* pose. The save method allows the user to save data recorded by the Processor in any format the user desires. By imposing few constraints on the Processor object, this tool is very flexible; for example, it can be used to read and write from a variety of commonly used data acquisition and input/output devices, including National Instruments devices, Arduino and Teensy micro-controllers, as well as Raspberry Pis and similar embedded systems. An example Processor object that uses a Teensy micro-controller to control a laser for optogenetics is provided in the DeepLabCut-live package.

We also provide functionality within this package to test inference speed of DeepLabCut networks. This serves to find the bounds for inference speeds an end user can expect given their hardware and pose estimation requirements. Additionally, there is a method to display the DeepLabCut estimated pose on top of images to visually inspect the accuracy of DeeplabCut networks.

Ultimately, this package is meant to serve as a software development kit (SDK): to be used to easily integrate real-time pose estimation and closed-loop feedback into other software, either that we provide (described below), or integrated into other existing camera capture packages.

Inference speed using the DeepLabCut-live package.

Maximizing inference speed is of utmost importance to experiments that require real-time pose estimation. Some

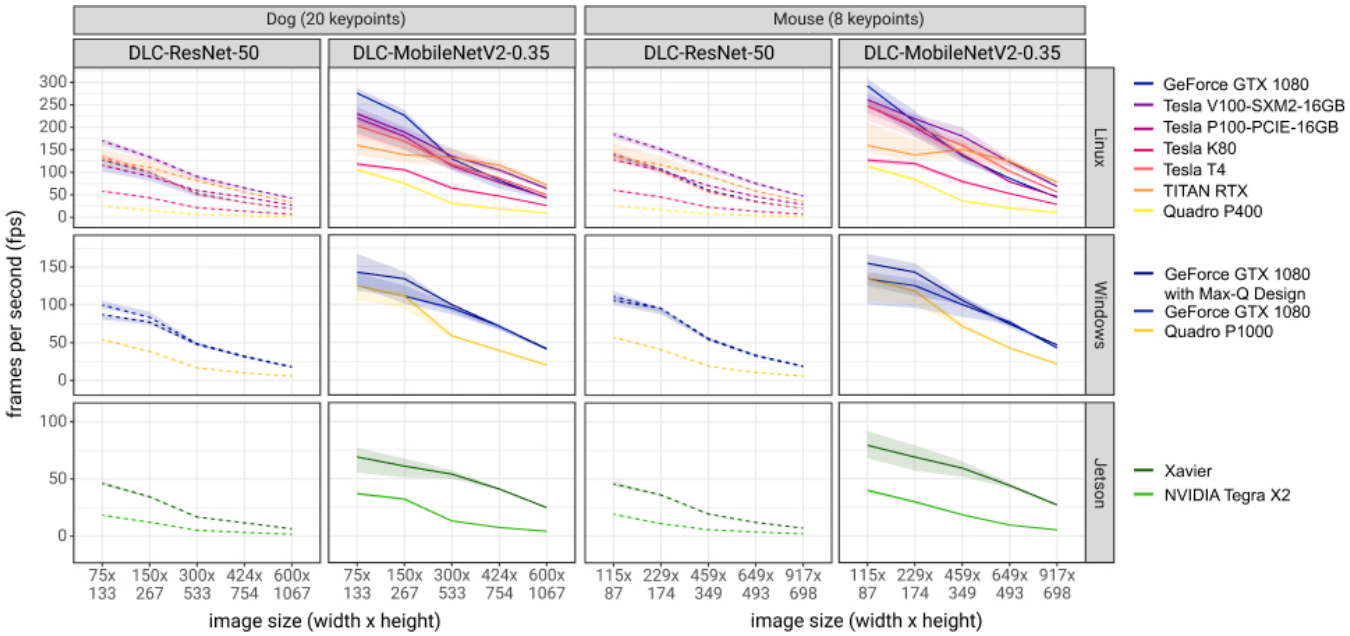

Figure. 2. Inference speed of different networks with different image sizes on different system configurations. Solid lines are tests using DLC-MobileNetV2-0.35 networks and dashed lines using DLC-ResNet-50 networks. Shaded regions represent 10th-90th percentile of the distributions. N = 1000-10000 observations per point. See Suppl. Figure 7 for CPU values.

known factors that influence inference speed of DeepLabCut networks include the size of the network, the size of images, and the computational power of the hardware (9, 10). Accordingly, we tested the inference speed of the `DeepLabCut-Live!` package using different networks, different image sizes, and on multiple systems with different computational power.

Specifically, we tested two different network architectures available within DeepLabCut: DLC-ResNet-50v1 (1, 4, 9, 21) and DLC-MobileNetV2-0.35 (9, 22). We tested different image sizes using image pre-processing methods built in to the `DeepLabCut-Live!` package. There are three methods to reduce the size of images to increase inference speed: static image cropping, dynamic cropping around keypoints, and downsizing images (see Methods). These methods are especially important tools, as experimenters may want to capture a higher resolution, larger "full frame" view, but can increase inference speed by either performing inference only on the portion of the image in which the animal is present (i.e., dynamically crop the image around the animal), or if the entire image is needed, by performing inference on an image with reduced resolution (i.e. a smaller image). In particular, we used the downsizing function to parametrically change image sizes for this test. Lastly, we performed these tests on a variety of hardware configurations, ranging from NVIDIA GPUs to Intel CPUs on Linux, Windows, and MacOS computers, as well as NVIDIA Jetson developer kits–inexpensive embedded systems with on-board GPUs (Figure 2).

For each test, we measured the inference speed on 1,000-10,000 images on CPUs and GPUs. As expected, inference speeds were faster for the smaller MobileNetV2-0.35 encoder than the larger ResNet-50 encoder (cf. 9), faster with smaller images, and faster when using more powerful NVIDIA GPUs compared to smaller GPUs or CPUs (Figure 2). For example, with the NVIDIA Titan RTX GPU (24 GB) and the NVIDIA GeForce GTX 1080 GPU (8 GB), we achieved inference speeds of 152 $\pm$ 15 and 134 $\pm$ 9 frames per second on medium sized images (459 x 349 pixels) using the MobileNetV2-0.35 DeepLabCut network. Full results are presented in Figure 2 and Table 1.

Moreover, we created a website that we aim to continuously update with user input: one can simply export the results of these tests (which capture information about the hardware automatically), and submit the results on GitHub[4]. These results, in addition to the extensive testing we provide below, then become a community resource for considerations with regard to GPUs and experimental design[5].

User-interfaces for real-time feedback.

In addition to the DeepLabCut-live package that serves as a SDK for experimenters to write custom real-time pose estimation applications, we provide three methods for conducting experiments that use DeepLabCut to provide closed-loop feedback that do not require users to write any additional code: a standalone user interface called the DLC-Live! GUI (DLG), and by integrating DeepLabCut into popular existing experimental control softwares Autopilot (16) and Bonsai (15).

DLC-Live! GUI.

The DLG provides a graphical user interface that simultaneously controls capturing data from a camera (many cam-

[4] `https://github.com/DeepLabCut/DLC-inferencespeed-benchmark`

[5] `https://deeplabcut.github.io/DLC-inferencespeed-benchmark/`

Table 1. Inference speed using the DeepLabCut-live package. All values are mean ± standard deviation; Frames Per Second (FPS).

	Dog Video		**image size (pixels, w*h)**				
	GPU Type	DLC Model	75 x 133	150 x 267	300 x 533	424 x 754	600 x 1067
Linux	Intel Xeon CPU	MobileNetV2-0.35	62 ± 5	31 ± 1	14 ± 0	8 ± 0	4 ± 0
	(3.1 GHz)	ResNet-50	24 ± 1	11 ± 0	3 ± 0	2 ± 0	1 ± 0
	GeForce GTX 1080	MobileNetV2-0.35	**256 ± 51**	**208 ± 46**	124 ± 19	80 ± 9	43 ± 1
		ResNet-50	121 ± 15	95 ± 13	52 ± 3	33 ± 1	19 ± 0
	TITAN RTX	MobileNetV2-0.35	168 ± 29	144 ± 20	133 ± 14	**115 ± 9**	**71 ± 3**
		ResNet-50	132 ± 14	113 ± 11	82 ± 4	56 ± 2	33 ± 1
	Tesla V100-SXM2-16GB	MobileNetV2-0.35	229 ± 13	189 ± 10	**138 ± 11**	105 ± 6	64 ± 3
		ResNet-50	169 ± 7	133 ± 4	90 ± 4	65 ± 2	42 ± 1
	Tesla P100-PCIE-16GB	MobileNetV2-0.35	220 ± 12	179 ± 9	114 ± 7	77 ± 3	44 ± 1
		ResNet-50	115 ± 3	91 ± 2	59 ± 2	45 ± 1	26 ± 1
	Tesla K80	MobileNetV2-0.35	118 ± 4	105 ± 3	64 ± 4	47 ± 2	26 ± 1
		ResNet-50	58 ± 2	43 ± 1	21 ± 1	13 ± 0	7 ± 0
	Tesla T4	MobileNetV2-0.35	200 ± 17	166 ± 13	117 ± 10	86 ± 5	49 ± 2
		ResNet-50	134 ± 8	99 ± 5	51 ± 3	33 ± 1	18 ± 0
	Quadro P400	MobileNetV2-0.35	105 ± 4	76 ± 2	31 ± 1	18 ± 0	10 ± 0
		ResNet-50	24 ± 0	16 ± 0	6 ± 0	4 ± 0	2 ± 0
Windows	Intel Xeon Silver CPU	MobileNetV2-0.35	28 ± 1	13 ± 0	6 ± 0	3 ± 0	1 ± 0
	(2.1 GHz)	ResNet-50	22 ± 1	9 ± 0	3 ± 0	2 ± 0	1 ± 0
	GeForce GTX 1080	MobileNetV2-0.35	142 ± 17	132 ± 14	98 ± 5	69 ± 3	41 ± 1
	with Max-Q Design	ResNet-50	87 ± 3	77 ± 3	48 ± 1	31 ± 1	18 ± 0
	GeForce GTX 1080	MobileNetV2-0.35	128 ± 11	115 ± 10	94 ± 7	72 ± 3	41 ± 1
		ResNet-50	101 ± 5	86 ± 4	49 ± 1	32 ± 1	18 ± 0
	Quadro P1000	MobileNetV2-0.35	120 ± 11	108 ± 10	58 ± 2	39 ± 1	20 ± 0
		ResNet-50	54 ± 2	38 ± 1	17 ± 0	10 ± 0	5 ± 0
MacOS	Intel Core i5 CPU	MobileNetV2-0.35	39 ± 5	20 ± 2	11 ± 1	7 ± 1	4 ± 0
	(2.4 GHz)	ResNet-50	8 ± 1	4 ± 0	1 ± 0	1 ± 0	0 ± 0
	Intel Core i7 CPU	MobileNetV2-0.35	117 ± 8	47 ± 3	15 ± 1	8 ± 0	4 ± 0
	(3.5 GHz)	ResNet-50	29 ± 2	11 ± 1	3 ± 0	2 ± 0	1 ± 0
	Intel Core i9 CPU	MobileNetV2-0.35	126 ± 25	66 ± 13	19 ± 3	11 ± 1	6 ± 0
	(2.4 GHz)	ResNet-50	31 ± 6	16 ± 2	6 ± 1	4 ± 0	2 ± 0
Jetson	Xavier	MobileNetV2-0.35	68 ± 8	60 ± 7	54 ± 4	41 ± 1	25 ± 1
		ResNet-50	46 ± 1	34 ± 1	17 ± 0	12 ± 0	6 ± 0
	Tegra X2	MobileNetV2-0.35	37 ± 2	32 ± 2	13 ± 0	7 ± 0	4 ± 0
		ResNet-50	18 ± 1	12 ± 0	5 ± 0	3 ± 0	2 ± 0
	Mouse Video		**image size (pixels, w*h)**				
	GPU Type	DLC Model	115 x 87	229 x 174	459 x 349	649 x 493	917 x 698
Linux	Intel Xeon CPU	MobileNetV2-0.35	61 ± 4	32 ± 1	15 ± 0	8 ± 0	4 ± 0
	(3.1 GHz)	ResNet-50	28 ± 1	11 ± 0	4 ± 0	2 ± 0	1 ± 0
	GeForce GTX 1080	MobileNetV2-0.35	**285 ± 24**	**209 ± 23**	134 ± 9	86 ± 2	44 ± 1
		ResNet-50	136 ± 8	106 ± 3	60 ± 1	35 ± 0	19 ± 0
	TITAN RTX	MobileNetV2-0.35	169 ± 28	145 ± 19	152 ± 15	**124 ± 9**	**78 ± 3**
		ResNet-50	140 ± 16	119 ± 11	92 ± 3	58 ± 2	35 ± 1
	Tesla V100-SXM2-16GB	MobileNetV2-0.35	260 ± 12	218 ± 9	**180 ± 18**	121 ± 8	68 ± 3
		ResNet-50	184 ± 6	151 ± 5	111 ± 6	75 ± 3	47 ± 2
	Tesla P100-PCIE-16GB	MobileNetV2-0.35	246 ± 12	198 ± 7	138 ± 8	79 ± 3	46 ± 1
		ResNet-50	128 ± 3	103 ± 2	70 ± 3	46 ± 1	28 ± 1
	Tesla K80	MobileNetV2-0.35	127 ± 6	119 ± 5	79 ± 4	52 ± 2	28 ± 1
		ResNet-50	60 ± 2	45 ± 2	23 ± 1	13 ± 0	7 ± 0
	Tesla T4	MobileNetV2-0.35	242 ± 21	197 ± 16	156 ± 14	101 ± 6	56 ± 2
		ResNet-50	141 ± 7	102 ± 5	57 ± 3	34 ± 1	20 ± 0
	Quadro P400	MobileNetV2-0.35	114 ± 5	84 ± 3	37 ± 1	20 ± 0	10 ± 0
		ResNet-50	25 ± 0	16 ± 0	7 ± 0	4 ± 0	2 ± 0
Windows	Intel Xeon Silver CPU	MobileNetV2-0.35	28 ± 1	14 ± 0	6 ± 0	3 ± 0	1 ± 0
	(2.1 GHz)	ResNet-50	23 ± 1	9 ± 0	3 ± 0	2 ± 0	1 ± 0
	GeForce GTX 1080	MobileNetV2-0.35	147 ± 17	136 ± 15	108 ± 6	73 ± 3	46 ± 1
	with Max-Q Design	ResNet-50	107 ± 5	93 ± 4	54 ± 2	32 ± 1	18 ± 0
	GeForce GTX 1080	MobileNetV2-0.35	133 ± 15	119 ± 14	100 ± 9	77 ± 3	43 ± 1
		ResNet-50	110 ± 5	94 ± 3	55 ± 1	34 ± 1	19 ± 0
	Quadro P1000	MobileNetV2-0.35	129 ± 13	115 ± 11	72 ± 3	43 ± 1	22 ± 0
		ResNet-50	57 ± 2	41 ± 1	19 ± 0	10 ± 0	6 ± 0
MacOS	Intel Core i5 CPU	MobileNetV2-0.35	42 ± 5	22 ± 2	12 ± 1	7 ± 1	4 ± 0
	(2.4 GHz)	ResNet-50	9 ± 1	4 ± 0	2 ± 0	1 ± 0	0 ± 0
	Intel Core i7 CPU	MobileNetV2-0.35	127 ± 10	48 ± 3	16 ± 1	9 ± 1	4 ± 0
	(3.5 GHz)	ResNet-50	30 ± 3	10 ± 2	3 ± 0	2 ± 0	1 ± 0
	Intel Core i9 CPU	MobileNetV2-0.35	178 ± 15	74 ± 16	19 ± 3	10 ± 1	6 ± 0
	(2.4 GHz)	ResNet-50	35 ± 8	14 ± 2	6 ± 1	4 ± 0	2 ± 0
Jetson	Xavier	MobileNetV2-0.35	79 ± 9	68 ± 9	59 ± 5	44 ± 2	27 ± 1
		ResNet-50	46 ± 2	36 ± 1	19 ± 0	12 ± 0	7 ± 0
	TX2	MobileNetV2-0.35	39 ± 2	30 ± 2	18 ± 1	9 ± 0	5 ± 0
		ResNet-50	19 ± 1	11 ± 0	6 ± 0	4 ± 0	2 ± 0

era types are supported, see Methods), recording videos, and performing pose estimation and closed-loop feedback using the `DeepLabCut-Live!` package. To allow users to both record video and perform pose estimation at the fastest possible rates, these processes run in parallel. Thus, video data can be acquired from a camera without delays imposed by pose estimation, and pose estimation will not be delayed by the time spent acquiring images and saving video data to the hard drive. However, running image acquisition and pose estimation asynchronously can create additional delays: if pose

Table 2. Performance of the DeepLabCut-live-GUI (DLG). F-P = delay from image acquisition to pose estimation; F-L = delay from image acquisition to turning on the LED; FPS (DLG) = Rate of pose estimation (in frames per second) in the DeepLabCut-live-GUI; FPS (DLCLive) = Rate of pose estimation for the same exact configuration directly tested using the DeepLabCut-live benchmarking tool. All values are mean ± standard deviation.

			176 x 137 pixels				352 x 274 pixels			
	GPU Type	Mode	F-P (ms)	F-L (ms)	FPS (DLG)	FPS (DLCLive)	F-P (ms)	F-L (ms)	FPS (DLG)	FPS (DLCLive)
Windows	GeForce GTX 1080	Latency	10 ± 1	11 ± 1	56 ± 16	123 ± 16	12 ± 2	14 ± 73	48 ± 6	112 ± 12
		Rate	15 ± 3	16 ± 3	91 ± 11		16 ± 3	17 ± 3	84 ± 9	
Linux	Quadro P400	Latency	11 ± 1	11 ± 1	63 ± 19	105 ± 4	20 ± 1	21 ± 1	42 ± 7	52 ± 1
		Rate	15 ± 3	16 ± 3	93 ± 9		27 ± 4	27 ± 4	47 ± 4	
Jetson	Xavier	Latency	13 ± 1	14 ± 1	48 ± 3	84 ± 7	13 ± 1	14 ± 1	48 ± 3	73 ± 9
		Rate	18 ± 3	18 ± 3	75 ± 5		18 ± 3	18 ± 3	74 ± 5	
MacOS	CPU	Latency	29 ± 5	29 ± 4	29 ± 5	62 ± 6	79 ± 19	79 ± 22	12 ± 2	21 ± 3
		Rate	34 ± 7	35 ± 6	35 ± 4		91 ± 19	92 ± 22	12 ± 2	

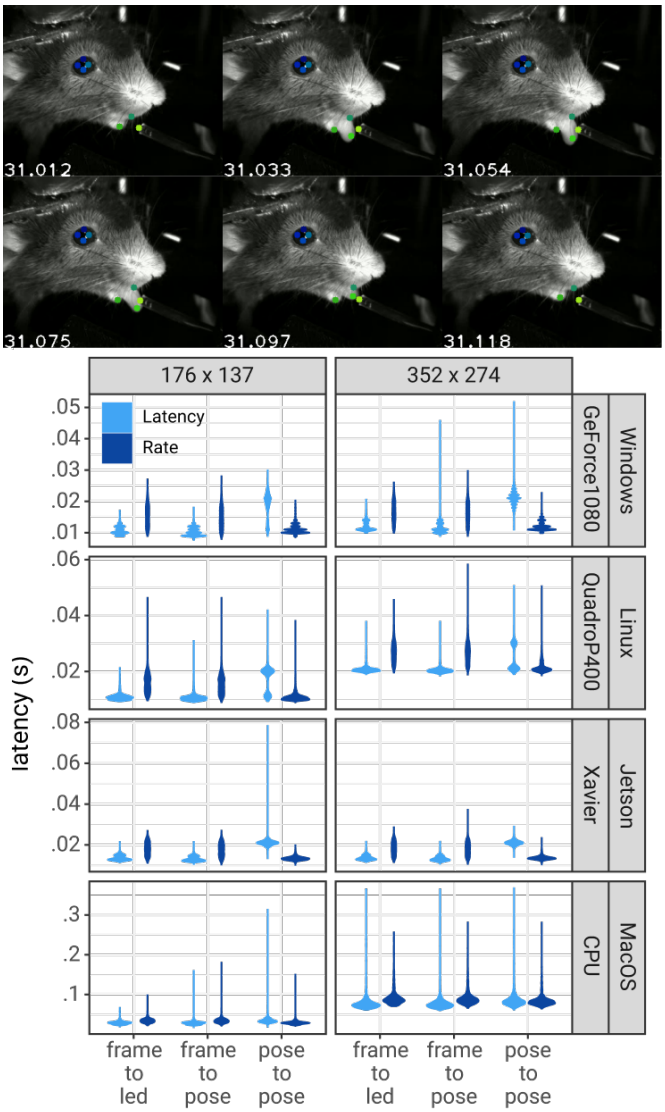

Figure. 3. Pose estimation latency using the DLC-Live!-GUI Top: A lick sequence from the test video with 8 keypoints measured using the DLC-Live!-GUI (using Windows/GeForce GTX 1080 GPU). Image timestamps are presented in the bottom-left corner of each image. **Bottom:** Latency from image acquisition to obtaining the pose, from the last pose to the current pose, and from image acquisition when the mouse's tongue was detected to turning on an LED. The width of the violin plots indicate the probability density – the likelihood of observing a given value on the y-axis.

estimation is run completely independent of image acquisition, there may be delays from the time an image was acquired to the time pose estimation on that image begins. To reduce this delay, the pose estimation process can wait for a new image to be acquired. This strategy reduces the latency from the time an image was acquired to the time a pose is returned, but pose estimation is performed less frequently. DLG gives users a choice of which mode they prefer: an "Optimize Rate" mode, in which pose estimation is performed at the maximum possible rate, but there may be delays from the time an image was captured to the time pose estimation begins, and an "Optimize Latency" mode, in which the pose estimation process waits for a new image to be acquired, minimizing the delay from the time an image was acquired to when the pose becomes available.

To measure the performance of DLG, we used a video of a head-fixed mouse performing a task that required licking to receive a liquid reward. To simulate a camera feed from an animal in real-time, single frames from the video were loaded (i.e., acquired) at the rate that the video was initially recorded– 100 frames per second. Here, we measured three latency periods: i) for each measured pose the delay from image acquisition to obtaining the pose; ii) the delay from one measured pose to the next measured pose; and iii) the delay from detecting an action (the mouse's tongue was detected) to turn on an LED. To measure the time from lick detection to turning on an LED, we used a `Processor` object that, when the tongue was detected, sent a command to a Teensy microcontroller to turn on an infrared LED. To determine that the LED had been activated, the status of the LED was read using an infrared photodetector. When the photodetector was activated, the Teensy reported this back to the `Processor`. The delay from image acquisition to turning on the LED was measured as the difference between the time the frame was acquired and the time that the photodetector had been activated.

This procedure was run under four configurations: pose estimation performed on full-size images (352 x 274 pixels) and images downsized by 50% in both width and height (176 x 137 pixels); both image sizes were run in "Optimize Rate" and "Optimize Latency" modes. These four configurations were run on four different computers to span a range of op-

tions (to see how they generally perform, please see Table 1): a Windows desktop with NVIDIA GeForce 1080 GPU, a Linux desktop with NVIDIA Quadro P400 GPU, a NVIDIA Jetson Xavier, and a MacBook Pro laptop with Intel Core-i7 CPU.

On a Windows system with GeForce GTX 1080 GPU, DLG achieved delays from frame acquisition to obtaining the pose as fast as 10 $\pm$ 1 ms (mean $\pm$ sd) in the "Optimize Latency" mode. Compared to the "Optimize Latency" mode, this delay was, on average, 4.38 ms longer (95% CI: 4.32-4.44 ms) with smaller images and 4.6 ms longer (95% CI: 4.51-4.63 ms) with the larger images in the "Optimize Rate" mode. As suggested above, the longer delay from frame acquisition to pose in the "Optimize Rate" mode can be attributed to delays from when images are acquired until pose estimation begins. With a frame acquisition rate of 100 FPS, this delay would be expected to be 5 ms with a range from 0-10 ms, as observed.

Running DLG In the "Optimize Rate" mode on this Windows system, the delay from obtaining one pose to the next was 11 $\pm$ 2 ms (rate of 91 $\pm$ 11 poses per second) for smaller images and 12 $\pm$ 1 ms (rate of 84 $\pm$ 9 poses per second) for larger images. Compared to the "Optimize Rate" mode, the "Optimize Latency" mode was 7.7 ms (95% CI: 7.6-7.8 ms) slower for smaller images and 9.1 ms (95% CI: 9.0-9.1 ms) for larger images. This increased delay from one pose to another can be attributed to time waiting for acquisition of the next image in the "Optimize Latency" mode.

Lastly, the delay from acquiring an image in which the tongue was detected until the LED could turned on/off includes the time needed to obtain the pose, plus additional time to determine if the tongue is present and to execute the control signal (send a TTL pulse to the LED). To determine the additional delay caused by detection of the tongue and sending a TTL signal to the LED, we compared the delay from image acquisition to turning on the LED with the delay from image acquisition to obtaining a pose in which the LED was not triggered. Detecting the tongue and sending a TTL pulse only took an additional 0.4 ms (95% CI: 0.3-0.6 ms). Thus, the delay from image acquisition to turn on the LED can be almost entirely attributed to pose estimation. Full results from all four tested systems can be found in Figure 3 and Table 2.

DeepLabCut models in Bonsai.

Bonsai is a widely used visual language for reactive programming, real-time behavior tracking, synchronization of multiple data streams and closed-loop experiments (15). It is written in C#, thus provides an alternative environment for running real-time DeepLabCut and also test the performance of native TensorFlow inference outside of a Python environment. We developed equivalent performance benchmarks for testing our newly developed Bonsai-DLC plugin. This plugin allows loading of the DeepLabCut exported .pb files directly in Bonsai.

We compared the performance of Bonsai-DLC and `DeepLabCut-Live!` on a Windows 10 computer with

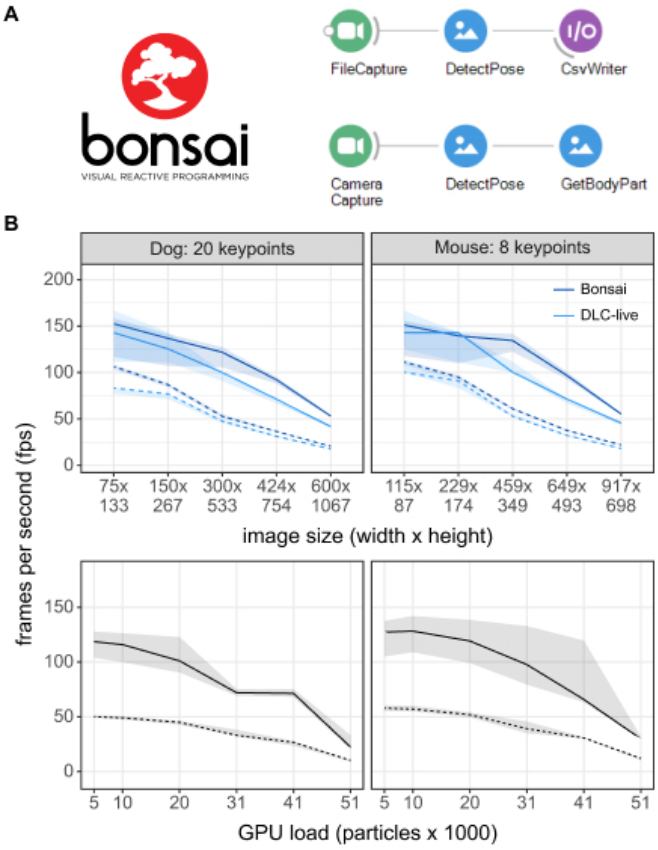

Figure. 4. Inference speed using the Bonsai-DLC Plugin. Dashed Lines are ResNet-50, solid lines are MobileNetV2-0.35 **(A)** Overview of using DeepLabCut within Bonsai. Both capture-to-write or capture to detect for real-time feedback is possible. **(B)** Top: Direct comparison of the inference speeds using Bonsai-DLC plugin vs. the DeepLabCut-Live! package across image sizes from the same computer (OS: Windows 10, GPU: NVIDIA Ge-Force 1080 with Max-Q Design). Bottom: Inference speeds using the Bonsai-DLC plugin while the GPU was engaged in a particle simulation. More particles indicates greater competition for GPU resources.

GeForce GTX 1080 with Max-Q design GPU, and found that the performance of running inference through Bonsai-DLC was comparable to `DeepLabCut-Live!` inference (Figure 4), suggesting that, as expected, inference speed is limited primarily by available CPU/GPU computational resources rather than by any native language interface optimizations. Moreover, we found the latency to be 34 ms $\pm$ 9.5 ms (median, IQR, n=500) tested at 30Hz with 384 x 307 pixels, which is equivalent to what was found with DLG above.

We then took advantage of the built-in OpenGL shader support in Bonsai to assess how external load on the GPU would impact DLC inference performance, as would happen when running closed-loop virtual reality simulations in parallel with video inference. To do this, we implemented a simulation of N-body particle interactions using OpenGL compute shaders in Bonsai, where we were able to vary the load on the GPU by changing the number of particles in the simulation, from 5,120 up to 51,200 particles. This is a quadratic problem as each particle interacts with every other particle, so it allows us to easily probe the limits of GPU load and its effects on competing processes.

Overall, we found that as the number of particle interactions increased the GPU load, there was a corresponding drop in DLC inference speeds (Figure-4. The effects of load on inference were non-linear and mostly negligible until the load on the GPU approached 50%, then started to drop as more and more compute banks were scheduled and used, likely due to on-chip memory bottlenecks in the GPU compute units (23). Nevertheless, as long as GPU load remained balanced, there was no obvious effect on inference speeds, suggesting that in many cases it would be possible to combine closed-loop accelerated DeepLabCut-live inference with real-time visual environments running on the same GPU (24).

Distributed DeepLabCut with Autopilot.

Autopilot is a Python framework designed to overcome problems of simultaneously collecting multiple streams of data by distributing different data streams over a swarm of networked computers (16). Its distributed design could be highly advantageous for naturalistic experiments that require large numbers of cameras and GPUs operating in parallel.

Thus, we integrated `DeepLabCut-Live!` into Autopilot in a new module of composable data transformation objects. As a proof of concept, we implemented the minimal distributed case of two computers: one Raspberry Pi capturing images and one NVIDIA Jetson TX2, an affordable embedded system with an onboard GPU, processing them (see Methods, Table 5).

We tested the performance of this system by measuring the end-to-end latency of a simple light detection task (Figure 5A). The Raspberry Pi lit an LED while capturing and streaming frames to the Jetson. Autopilot's networking modules stream arrays by compressing them on-the-fly with blosc(25) and routing them through a series of "nodes"– in this case each frame passed through four networking nodes in each direction. The Jetson then processed the frames in a chain of `Transform`s that extracted poses from frames using `DLC-Live!` (DLC-MobileNetV2-0.35) and returned a Boolean flag indicating whether the LED was illuminated. `True` triggers were sent back to the Raspberry Pi which emitted a TTL voltage pulse to the LED on receipt.

Experiments were performed with differing acquisition frame rates (30, 60, 90 FPS), and image sizes (128 x 128, 256 x 256, 512 x 416 pixels; Figure 5B). Frame rate had a little effect with smaller images, but at 512 x 416, latencies at 30 FPS (median = 161.3 ms, IQR = [145.6-164.4], n = 500) were 38.6 and 52.6 ms longer than at 60 FPS (median = 122.7 ms, IQR = [109.4 - 159.3], n = 500) and 90 FPS (median = 113.7 ms, IQR = [106.7 - 118.5], n = 500), respectively.

Frame rate imposes intrinsic latency due to asynchrony between acquired events and the camera's acquisition interval: e.g. if an event happens at the beginning of a 30 FPS exposure, the frame will not be available to process until 1/30s=33.3ms later. If this asynchrony is distributed uniformly then a latency of half the inter-frame interval is expected for a given frame rate. This quantization of frame

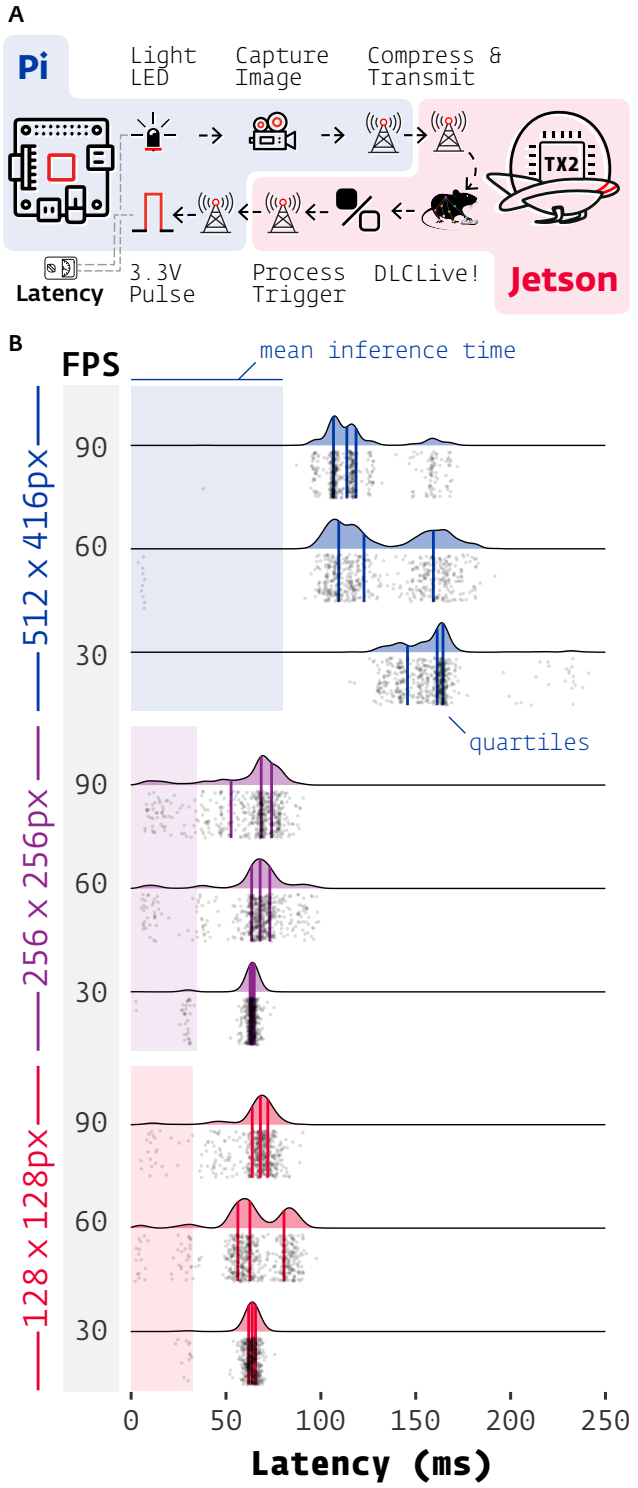

Figure. 5. Latency of DeepLabCut-Live with Autopilot. **(A)** Images of an LED were captured on a Raspberry Pi 4 (shaded blue) with Autopilot, sent to an Nvidia Jetson TX2 (shaded red) to process with DeepLabCut-Live and Autopilot Transforms, which triggered a TTL pulse from the Pi when the LED was lit. **(B)** Latencies from LED illumination to TTL pulse (software timestamps shown, points and densities) varied by resolution (color groups) and acquisition frame rate (shaded FPS bar). Processing time at each stage of the chain in **(A)** contributes to latency, but pose inference on the relatively slow TX2 (shaded color areas, from Figure 2) had the largest effect. Individual measurements (n=500 each condition) cluster around multiples of the inter-frame interval (eg. 1/30 FPS = 33.3ms).

rate latency can be seen in the multimodal distributions in Figure 5B, with peaks separated by multiples of their inter-frame interval. The inter-frame interval of inference with `DeepLabCut-Live!` imposes similar intrinsic latency. The combination of these two sources of periodic latency and occasional false-negatives in inference gives a parsimonious, though untested, account of the latency distribution for the 512x416 experiments.

Latency at different image sizes were primarily influenced by the relatively slow frame processing of the Jetson TX2 (See Figure 2). WIth smaller images (128 x 128 and 256 x 256), inference time (shaded areas in 5B) was the source of roughly half of the total latency (inference/median total latency, n=1500 each, pooled across frame rates. 128 x 128: 32.2/64.6 ms = 49.7%. 256 x 256: 34.5/65.1 ms = 53.0%). At 512 x 416, inference time accounted for between 50 and 70% of total latency (512 x 416, 30 FPS: 79.9/161.3ms = 49.5%, 90 FPS: 79.9/113.7ms = 70.3%). Minimizing instrument cost for experimental reproducibility and scientific equity is a primary design principle of Autopilot, so while noting that it would be trivial to reduce latency by using a faster GPU or the lower-latency Jetson Xavier (i.e., see above sections), we emphasize that `DeepLabCut-Live!` in Autopilot is very usable with $350 of computing power (TX2 with education discount: $299, 26; Raspberry Pi 4 2GB RAM: $35, 27).

Autopilot, of course, imposes its own latency in distributing frame capture and processing across multiple computers. Subtracting latency that is intrinsic to the experiment (frame acquisition asynchrony and GPU speed), Autopilot has approximately 25 ms of overhead (median of latency - mean DeepLabCut inference time - 1/2 inter-frame interval = 25.4 ms, IQR = [14.7 - 37.9], n = 4500). Autopilot's networking modules take 6.9 ms on average to route a message one-way (16), and have been designed for high throughput rather than low latency (message batching, on-the-fly compression).

Real-time feedback based on posture.

Lastly, to demonstrate practical usability of triggering a TTL signal based on posture, we performed an experiment using DLG on a Jetson Xavier in which an LED was turned on when a dog performed a "rearing" movement (raised forelimbs in the air, standing on only hindlimbs; Figure 6). First, a DeepLabCut network based on the ResNet-50 architecture was trained to track 20 keypoints on the face, body, forelimbs, and hindlimbs of a dog (see Methods). Next, the Jetson Xavier running DLG was used to record the dog as she performed a series of "rearing" movements in response to verbal commands, with treats given periodically by an experimenter. Video was recorded using a Logitech C270 webcam, with 640 x 480 pixel images at 30 FPS. Inference was run on images downsized by 50% (320 x 240 pixels), using the "Optimize Rate" mode in DLG.

The dog was considered to be in a "rearing" posture if the vertical position of at least one of the elbows was above the vertical position of the withers (between the shoulder blades). Similar to the mouse experiment, a `Processor` was used to detect "rearing" postures and control an LED via communication with a Teensy micro-controller (Figure 6A). The LED was turned on upon the first image in which a "rearing" posture was detected, and subsequently turned off upon the first image in which the dog was not in a "rearing" posture (for a fully closed-loop stimulus).

This setup achieved a rate of pose estimation of 22.417 $\pm$ 0.928 frames per second, with an image to pose latency of 61 $\pm$ 10 ms (N = 1848 frames) and, on images for which the LED was turned on or off, an image to LED latency of 59 $\pm$ 11 ms (N = 9 "rearing" movements). However, using DLG, if the rate of pose estimation is slower than video acquisition, not all images will be used for pose estimation (N = 2433 total frames, 1848 poses recorded). To accurately calculate the delay from the ideal time to turn the LED on or off, we must compare the time of the first frame in which a "rearing" posture was detected from all images recorded, not only from images used for pose estimation. To do so, we estimated the pose on all frames recorded offline using the same exported DeepLabCut model and calculated the ideal times that the LED would have been turned on or off all available images. According to this analysis, there was a delay of 70 $\pm$ 23 ms to turn the LED on or off (consistent with estimates on Jetson systems shown above).

As shown above, there are three methods that we have implemented that could reduce these delays: i) training a DeepLabCut model based on the MobileNetV2 architecture (vs ResNet-50), ii) using more computationally powerful GPU-accelerated hardware for pose estimation (see 2), iii) or acquiring video at a higher frame rate, which reduces potential delays between the time an image was acquired and the time pose estimation begins in the "Optimize Rate" mode, or reducing the time the pose estimation process would have to wait for a newly acquired frame in the "Optimize Latency" mode. However, no matter how fast the hardware system, there will be some delay from acquiring images, estimating pose, and providing feedback.

To overcome these delays, we developed another method to reduce latency for highly sensitive applications. Namely, we can perform a forward prediction– to predict the animal's future pose before the next image is acquired and processed. Depending on the forward prediction model, this could potentially reach zero-latency feedback levels (or below)– a dream for experimentalists who aim to study timing of causal manipulations in biological systems.

To reduce the delay to turn on the LED when the dog exhibited a "rearing" movement, we implemented a Kalman filter that estimated the position, velocity, and acceleration of each keypoint, and used this information to predict the position of each keypoint at the current point in time (i.e., not the time at which the image was acquired, but after time had been taken to process the image). For example, if the image was acquired 60 ms ago, the Kalman filter "Processor" predicted keypoints 60 ms into the future.

Thus, in another set of experiments, we recorded as the dog

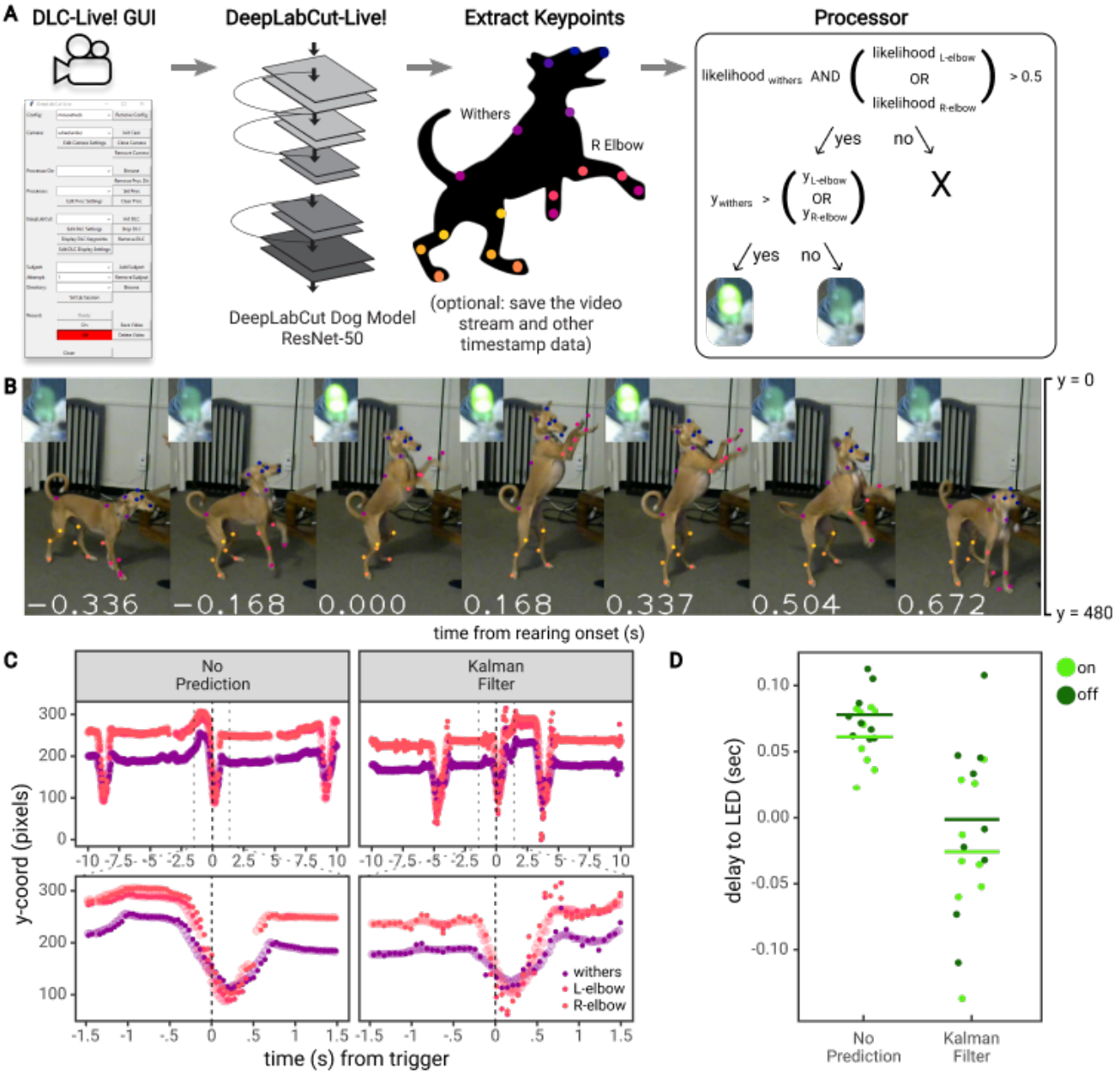

Figure. 6. Real-time feedback based on posture. A) Diagram of the workflow for the dog feedback experiment. Image acquisition and pose estimation were controlled by the DLC-Live! GUI. A DeepLabCut-live Processor was used to detect "rearing" postures based on the likelihood and y-coordinate of the withers and elbows and turn an LED on or off accordingly. B) An example jump sequence with LED status, labeled with keypoints measured offline using the DeepLabCut-live benchmarking tool. The images are cropped for visibility. C) Example trajectories of the withers and elbows, locked to one jump sequence. Larger, transparent points represent the true trajectories – trajectories measured offline, from each image in the video. The smaller opaque points represent trajectories measured in real-time, in which the time of each point reflects the delay from image acquisition to pose estimation, with and without the Kalman filter forward prediction. Without forward prediction, estimated trajectories are somewhat delayed from the true trajectories. With the Kalman filter forward prediction, trajectories are less accurate but less delayed when keypoints exhibit rapid changes in position, such as during a rearing movement. D) The delay from the time the dog first exhibited a rearing posture (from postures measured offline) to the time the LED was turned on or off. Each point represents a single instance of the detection of a transition to a rearing posture or out of a rearing posture.

performed a series of "rearing" movements, this time using a `Processor` that first forward-predicted the dog's pose, then detected "rearing" movements and controlled an LED accordingly. Using the `Processor` with a Kalman filter reduced inference speed compared to the `Processor` without the Kalman filter, due to the time required to compute Kalman filter updates(15.616 $\pm$ 0.332 frames per second). The image to pose latency was 81 $\pm$ 10 ms (N = 1187 frames), and image to LED latency of 82 $\pm$ 11 ms (N = 9 "rearing" movements). However, compared to the ideal times to turn the LED on or off calculated from pose estimation performed on all available images, the Kalman filter "Processor" achieved a delay of -13 $\pm$ 61 ms. In this case, the Kalman filter turned the LED on or off 13 ms *prior* to the point at which the first rearing posture was detected. These results indicate the potential to perform zero or negative latency feedback based on posture.

Discussion

Providing event-triggered feedback in real-time is one of the strongest tools in neuroscience. From closed-loop optogenetic feedback to behaviorally-triggered virtual reality, some of the largest insights from systems neuroscience have come through causally testing the relationship of behavior to the brain (28–30). Although tools for probing both the brain and for measuring behavior have become more advanced, there is still the need for such tools to be able to seamlessly interact. Here, we aimed to provide a system that can provide real-time feedback based on advances in deep learning-based pose estimation. We provide new computational tools to do so with high speeds and low latency, as well as a a full benchmarking test suite (and related website: `https://deeplabcut.github.io/DLC-inferencespeed-benchmark/`), which we hope enables ever more sophisticated experimental science.

Related Works. DeepLabCut and related animal pose estimation tools (reviewed in 8) have become available starting in early 2018, and two groups have built tools around real-time applications with DeepLabCut. However, the reported speed and latencies are slower than what we were able to achieve here: Forys et al. 2020 (12) latencies of 30 ms using top-end GPUs, but this delay increases if the frame acquisition rate is increased beyond 100 frames per second. Schweihoff et al. 2019 (14) also achieve latencies of 30 ms from frame acquisition to detecting a behavior of interest (round-trip frame to LED equivalent was not reported). We report a 2-3x reduction in latency (11 ms/16 ms from frame to LED in the "Optimize Latency"/"Optimize Rate" mode of DLG) on a system that uses a less powerful GPU (Windows/GeForce GTX 1080) compared to these studies, and equivalent performance (29 ms/35 ms from frame to LED in the "Optimize Latency"/"Optimize Rate" mode of DLG) on a conventional laptop (MacBook Pro with Intel Core-i7 CPU). Although we believe such tools can use the advances presented in this work to achieve higher frame rates and lower latencies, our new real-time approach provides an improvement in portability, speed, and latency.

Animal pose estimation toolboxes, like DeepLabCut, have all benefited from advances in human pose estimation research. Although the goals do diverge (reviewed in 8) in terms of required speed, the ability to create tailored networks, and accuracy requirements, competitions on human pose estimation benchmarks such as PoseTrack (31) and COCO (32) have advanced computer vision. Several human pose estimation systems have real-time options: OpenPose (2) has a real-time hand/face pose tracker available, and PifPaf (33) reaches about 10Hz on COCO (32, depending on the backbone). On the challenging multi-human PoseTrack (31) benchmark, LightTrack (34) reaches less than 1Hz. However, recent work achieves 3D multi-human pose estimation at remarkable frame rates (35), in particular they report an astonishing 154 FPS for 12 cameras with 4 people in the frame. State of the art face detection frameworks, based on optimized architectures such as BlazeFace can achieve remarkable speeds of >500 FPS on GPUs of cell phones (36). The novel (currently unpublished) multi-animal version of DeepLabCut can also be used for feedback, and depending on the situation, tens of FPS for real-time applications should be possible. Inference speed can also be improved by various techniques such as network pruning, layer decomposition, weight discretization or feed-forward efficient convolutions (37). Plus, the ability to forward predict postures, as we show here, can be used to compensate for hardware delays.

Scalability, affordability, and integration into existing pipelines.

If neuroscience's embrace of studying the brain in its natural context of complex, contingent, and open-ended behavior (8, 38, 39) is smashing the champagne on a long-delayed voyage, the technical complexity of the experiments is the grim spectre of the sea. Markerless tracking has already enabled a qualitatively new class of data-dense behavioral experiments, but the heroism required to simultaneously record natural behavior from 62 cameras (40) or electrophysiology from 65,000 electrodes (41), or integrate dozens of heterogeneous custom-built components (42) hints that a central challenge facing neuroscience is *scale*.

Hardware-intensive experiments typically come at a significant cost, even if the pose estimation tools are "free" (developed in laboratories at a non-significant expense, but provided open source). Current commercial systems are expensive–up to $10,000–and they have limited functionality; these systems track the location of animals but not postures or movements of experimenter-defined points of the animal, and few to none offer any advanced deep learning-based solutions. Thus, being able to track posture with state-of-the-art computer vision at scale is a highly attractive goal.

`DeepLabCut-Live!` experiments are a many-to-many computing problem: many cameras to many GPUs (and coordinated with many other hardware components). The Autopilot experiment we described is the simplest 2-computer case of distributed applications of `DeepLabCut-Live!`, but Autopilot provides a framework for its use with arbitrary numbers of cameras and GPUs in parallel. Autopilot is not prescriptive about hardware configuration, so for example if lower latencies were needed users could capture frames on the same computer that processes them, use more expensive GPUs, or use the forward-prediction mode. Along with the rest of its hardware, experimental design, and data management infrastructure, integration of DeepLabCut in Autopilot makes complex experiments scalable and affordable.

Thus, here we presented options that span ultra high-performance (at GPU cost) to usable, affordable solutions that will work very well for most all applications (i.e., up to 90 FPS with zero to no latency if using our forward-prediction mode). Indeed, the Jetson experiments that we performed used simple hardware (inexpensive webcam, simple LED circuit) and either the open source DLC-Live! GUI or AutoPilot.

In addition to integration with Autopilot, we introduce integration of real-time DeepLabCut into Bonsai, a popular framework that is already integrated into many popular neuroscience tools such as OpenEphys (43), BonVision (24), and Bpod[6]. The merger of DeepLabCut and Bonsai could therefore allow for real-time posture tracking with sophisticated neural feedback with hardware such as NeuroPixels, Miniscopes, and beyond. For example, Bonsai and the newly released BonVision toolkit (24) are tools for providing real-time virtual reality (VR) feedback to animals. Here, we tested the capacity for a single GPU laptop system to run Bonsai-DLC with another computational load akin to what is needed for VR, making this an accessible tool for systems neuroscientists wanting to drive stimuli based on potentially sophisticated postures or movements. Furthermore, in our real-time dog-feedback utilizing the forward-prediction mode we utilized both posture and kinematics (velocity) to be able to achieve sub-zero latency.

Sharing DLC Models.

With this paper we also introduce three new features within the core DeepLabCut ecosystem. One, the ability to easily export trained models without the need to share project folders (as previously); two, the ability to load these models into other frameworks aside from DLC-specific tools; and three, we modified the code-base to allow for frozen-networks. These three features are not only useful for real-time applications, but if users want to share models more globally (as we are doing with the DeepLabCut Model-Zoo Project), or have a easy-install lightweight DeepLabCut package on dedicated machines for running inference, this is an attractive option. For example, the protocol buffer files are system and framework agnostic: they are easy to load into TensorFlow (44) wrappers based on C++, Python, etc. This is exactly the path we pursued for Bonsai's plugin via a C#-TensorFlow wrapper (`https://github.com/migueldeicaza/TensorFlowSharp`). Moreover, this package can be utilized even in offline modes where batch processing is desirable for very large speed gains (9, 10).

Feedback on posture and beyond.

To demonstrate the capabilities of `DeepLabCut-Live!` we performed a set of experiments where an LED was triggered based on the confidence of the DeepLabCut network and the posture of the animal (here a dog, but as is DeepLabCut, this package is animal and object agnostic). We also provide a forward-prediction mode that utilizes temporal information via kinematics to predict future postural states. But, the software is not limited to this. For example, one can build `Processor` objects to trigger on joint angles, or more abstract targets such as being in a particular high dimensional state space. We believe the flexibility of this feedback tool, plus the ability to record long-time scale videos for "standard" DeepLabCut analysis makes this broadly applicable to many applications.

[6]Developed by Sanworks: `https://www.sanworks.io/index.php`

Conclusions.

We report the development of a new light-weight Python pose estimation package based on DeepLabCut, that can be integrated with behavioral control systems (such as Bonsai and AutoPilot) or used within a new DLC-Live! GUI. This toolkit allows users to do real-time, low-latency tracking of animals (or objects) on high-performance GPU cards or on low cost, affordable and scalable systems. We envision this being useful for precise behavioral feedback in a myriad of paradigms.

Methods

Alongside this publication we developed several software packages that are available on GitHub. Links are listed in table 3 and details provided throughout the paper.

Name	URL
DeepLabCut-Live! SDK	GitHub Link
Benchmarking Submission	GitHub Link
Benchmarking Results	Website Link
DLC-Live! GUI	GitHub Link
Bonsai - DLC Plugin	GitHub Link
AutoPilot - DLC	GitHub Link

Table 3. Software packages presented with this paper

Animals.

All mouse work was carried out under the permission of the IACUC at Harvard University (#17-07-309). Dog videos and feedback was exempt from IACUC approval (with conformation of this with IACUC).

Mice were surgically implanted with a headplate as in (45). In brief, using aseptic technique mice were anesthetized to the surgical plane, a small incision in the skin was made, the skull was cleaned and dried and a titanium headplate was placed with Metabond. Mice were allowed 7 days to recover, and given burphrenorphine for 48 hours post-operatively.

The dog used in this paper was previously trained to perform rearing actions for positive reinforcement, and therefore no direct behavioral manipulation was done.

DeepLabCut.

The mouse DeepLabCut model was previously trained according to the protocol in (9, 20). Briefly, the DeepLabCut toolbox (version 2.1.6.4) was used to i) extract frames from selected videos, ii) manually annotate keypoints on selected frames, iii) create a training dataset to train the convolutional neural network, iv) train the neural network, v) evaluate the performance of the network, and vi) refine the network. This network was trained on a total of 120 labeled frames.

The dog model was initially created based on the "full_dog" model available from the DeepLabCut Model Zoo (ResNet-50, with ADAM optimization and imgaug augmentation (46); currently unpublished, more details will be provided elsewhere). Prior to running the DLG feedback experiments,

initial training videos were taken, frames from these videos were extracted and labeled, and the model was retrained using imgaug with the built in scaling set to 0.1-0.5 to optimize network accuracy on smaller images. This network was retrained with 157 labelled frames.

After training, DeepLabCut models were exported to a protocol buffer format (.pb) using the new export model feature in the main DeepLabCut package (2.1.8). This can be performed using the command line:

```
dlc model-export /path/to/config.yaml
```

or in python:

```
import deeplabcut as dlc
dlc.export_model("/path/to/config.yaml")
```

Table 4. Relevant DLC updates

Feature	DLC version	Pub. Link
DLC-ResNets	1, 2.0+	4, 20
DLC-MobileNetV2s	2.1+	9
Model Export Fxn	2.1.8+	this paper
DeepLabCut-live	new package	this paper

DeepLabCut-Live! package.

The DeepLabCut-Live code was written in Python 3 (www.python.org), and distributed as open source code on GitHub and on PyPi. It utilizes TensorFlow (44), numpy (47), scipy (48), openCV (49), and others. Please see GitHub for complete, platform-specific installation instructions and description of the package.

The DeepLabCut-live package provides a DLCLive class that facilitates loading DeepLabCut models and performing inference on single images. The DLC-Live class also has built in image pre-processing methods to reduce the size of images for faster inference: image cropping, dynamic image cropping around detected keypoints, and image downsizing. DLC-Live objects can be instantiated in the following manner:

```
from dlclive import DLCLive
my_live_object =
    DLCLive("/path/to/exported/model/directory")

# base instantiation
my_live_object =
    DLCLive("/path/to/exported/model/directory")

# use only the first 200 pixels in both
    width and height dimensions of image
my_live_object =
    DLCLive("/path/to/exported/model/directory",
            cropping=[0, 200, 0, 200])

# dynamically crop image around detected
    keypoints, with 20 pixel buffer
my_live_object =
    DLCLive("/path/to/exported/model/directory",
            dynamic=(True, 0.5, 20))

# resize height and width of image to 1/2
    its original size
my_live_object =
    DLCLive("/path/to/exported/model/directory",
            resize=0.5)
```

Inference speed tests were run using a benchmarking tool built into the DeepLabCut-Live package. Different image sizes were tested by adjusting the *pixels* parameter, which specifies the total number of pixels in the image while maintaining the aspect ratio of the full-size image. For all options of the benchmarking tool, please see GitHub or function documentation. Briefly, this tool can be used from the command line:

```
dlc-live-benchmark /path/to/model/directory
    /path/to/video/file -o
    /path/to/output/directory
```

or from python:

```
from dlclive import benchmark_videos
benchmark_videos("/path/to/model/directory",
                 "/path/to/video",
          output="/path/to/output/directory")
```

DLC-Live! GUI software.

The DLC-Live! GUI (DLG) code was also written in Python 3 (www.python.org), and distributed as open source code on GitHub and on PyPi. DLG utilizes Tkinter for the graphical user interface. Please see Github for complete installation instructions and a detailed description of the package.

DLG currently supports a wide variety of cameras across platforms. On Windows, DLG supports The Imaging Source USB cameras and OpenCV compatible webcams. On MacOS, DLG supports OpenCV webcams, PlayStation Eye cameras using the https://github.com/bensondaled/pseyepy, and USB3 Vision and GigE Vision cameras using the https://github.com/AravisProject/aravis. On Linux, DLG supports any device compatible with Video4Linux drivers using OpenCV, and USB3 Vision and GigE Vision devices using the Aravis Project.

DLG uses the multiprocess package (50) to run image acquisition, writing images to disk and pose estimation in separate processes from the main user-interface. Running these processes in parallel enables users to record higher frame rate videos with minimal sacrifice to pose-estimation speed. However, there are still some delays when running image acquisition and pose-estimation asynchronously: if these processes are run completely independently, the image may not have been acquired immediately before pose-estimation begins. For example, if images are acquired at 100 frames per second, the image will have been acquired with a range of 0-10 ms prior to running pose-estimation on the image. If the pose-estimation process waits for a new image to be acquired, then there will be a delay between completing pose-estimation on one image and beginning pose-estimation on

the next one. Accordingly, DLG allows users to choose between two modes: i) the Latency mode, in which the pose-estimation process waits for an a new image to reduce the latency between image acquisition and pose-estimation, and ii) the Rate mode, in which the pose-estimation process runs independently of image acquisition. In this mode, there will be longer latencies from image acquisition to pose-estimation but the rate of pose-estimation will be faster than in the Latency mode.

To test the performance of DLG, we used a video of a mouse performing a task that required licking to receive reward in the form of a drop of water. Video was collected at 100 frames per second using a The Imaging Source USB3 camera (model number: DMK 37AUX287) and Camera_Control software (51).

We tested the performance of DLG under four conditions– on full-size images (352 x 274 pixels) and downsized images (176 x 137 pixels), both image sizes in Latency mode and Rate mode. All four conditions were tested on four different computers (see Table 2 for specifications).

Postural feedback experiment.

Prior to conducting the dog feedback experiment, the dog was extensively trained to "rear" upon the verbal command "jump" and the visual cue of a human's arm raised at shoulder height. "Rearing" was reinforced by manually providing treats for successful execution. Prior to recording videos for tracking and feedback, the dog routinely participated in daily training sessions with her owners. There was no change to the dog's routine when implementing sessions in which feedback was provided.

To conduct the feedback experiment, we used the DLG software on a Jetson Xavier developer kit. Pose estimation was performed using an exported DeepLabCut model (details regarding training provided above). Feedback was provided using a custom DeepLabCut-Live `Processor` that detected "rearing" movements and controlled an LED via serial communication to a Teensy microcontroller.

The dog was considered to be in a "rearing" posture if a) the likelihood of the withers and at least one elbow was greater than 0.5 and b) the vertical position of at least one elbow, whose likelihood was greater than 0.5, was above the vertical position of the withers (i.e., $y_{\text{elbow}} < y_{\text{withers}}$, with the top of the image as $y = 0$ and bottom of image as $y =$ image height. For each pose, the `Processor` determined whether the dog was in a "rearing" posture, queried the current status of the LED from the Teensy microcontroller, and if the current status did not match the dog's posture (i.e., if the LED was off and the dog was in a "rearing" posture), sent a command to the Teensy to turn the LED on or off.

In this experiment, we recorded the time at which images were accessed by DLG, the time at which poses were obtained by DLG after processing, and the times that the LED was turned on or off by the `Processor`. We calculated the pose estimation rate as the inverse of the delay from obtaining one pose to the next, the latency from image acquisition to obtaining the pose from that image, and, for poses in which the `Processor` turned the LED on or off, the latency from image acquisition to sending the command to turn the LED on or off.

As not all images will be run through pose estimation using DLG, to assess the delay from behavior to feedback, we performed offline analyses to determine the ideal time to turn the LED on or off given all the acquired images. Using the DeepLabCut-live benchmarking tool, we obtained the pose for all frames in the acquired videos by setting the `save_poses` flag from the command line:

```
dlc-live-benchmark /path/to/model/directory
    /path/to/video/file --save-poses -n 0
```

This command can also be run from python:

```
from dlclive import benchmark_videos
benchmark_videos("/path/to/model/directory",
                 "/path/to/video/file",
                 n_frames=0,
                 save_poses=True)
```

We then ran this full sequence of poses through the "rearing" detection `Processor`, and compared these times – for each "rearing" movement, the time at which the first frame that showed a transition to a "rearing" posture and out of a "rearing" posture was acquired – with the times that the LED was turned on or off during real-time feedback.

To implement the forward-predicting Kalman filter, we used a `Processor` object that first used a Kalman filter to estimate the position, velocity, and acceleration of each keypoint; used the position, velocity, and acceleration to predict the position of the limb into the future– the amount of time it predicted into the future depended on the delay for that image (prediction time $=$ current time $-$ image acquisition time) – and finally, checked if the dog was in a "rearing" posture and controlled the LED accordingly. For the exact implementation, please see the source code on Github for the Kalman filter Processor and the dog rearing Processor.

Details of AutoPilot setup.

Latencies were measured using software timestamps and confirmed by oscilloscope. Software measurements could be gathered in greater quantity but were reliably longer than the oscilloscope latency measurements by 2.8 ms (median of difference, n=75, IQR=[2.4 - 3.4]), thus we use the software measurements noting they are a slightly conservative estimate of functional latency.

Autopilot experiments were performed using the `DLC_Latency` Task and the `Transformer` "Child" (see (16) for terminology).

Separate DLC-MobileNetV2-0.35 models tracking a single point were trained for each capture resolution (128 x 128, 256 x 256, 512 x 416 pixels). Training data was labeled such that the point was touching the LED when it was on, and in the

corner of the frame when the LED was off. Frames were processed with a chain of Autopilot `Transform` objects like:

```
from autopilot import transform as t

# create transformation object
tfm = t.image.DLC("/model/path") + \
      t.selection.DLCSLice("part_name") + \
      t.logical.Condition(
         minimum = [min_x, min_y],
         maximum = [max_x, max_y]
      )
# process a frame, yielding a bool
# true/false == LED on/off
led_on = tfm.process(frame)
```

where `min_x, min_y`, etc. defined a bounding box around the LED.

Tool	Version
Raspberry Pi	4, 2GB
Autopilot	0.3.0-2f31e78
Jetson	TX2 Developer Kit
Camera	FLIR CM3-U3-13Y3M-CS
Spinnaker SDK	2.0.0.147
Oscilloscope	Tektronix TDS 2004B

Table 5. Materials for Autopilot Tests

Data Analysis and Visualization.

Autopilot data were analyzed with Pandas (1.0.5) (52) in Python and Tidyverse (1.3.0) (53) in R. Data were visualized with ggplot2 (3.3.0) (54) and ggridges (0.5.2) (55).

Acknowledgments:

We thank Sébastien Hausmann for testing the DLC-Live GUI, and to Jessy Lauer for optimization assistance. We thank Jessica Schwab and Izzy Schwane for assistance with the dog feedback experiment. We greatly thank Michael Wehr for support of AutoPilot-DLC, and the Mathis Lab for comments. Funding was provided by the Rowland Institute at Harvard University and the Chan Zuckerberg Initiative DAF, an advised fund of Silicon Valley Community Foundation to AM and MWM, a Harvard Mind, Brain, Behavior Award to GK, MWM; and NSF Graduate Research Fellowship No. 1309047 to JLS. The authors declare no conflicts of interest.

Contributions:

G.K., A.M., M.W.M conceptualized the project. G.K., J.S., G.L, A.M., M.W.M designed experiments. G.K. and J.S. developed Jetson integration. G.K. developed and performed experiments for DLC-Live, & the DLC-Live GUI. J.S. developed and performed experiments for AutoPilot-DLC. G.L. developed and performed experiments for Bonsai-DLC. A.M. contributed to Bonsai-DLC and DLC-Live. All authors performed benchmarking tests. G.K. and M.W.M. original draft, all authors contributed to writing. M.W.M., A.M. acquired funding and supervised the project.

References

1. Eldar Insafutdinov, Leonid Pishchulin, Bjoern Andres, Mykhaylo Andriluka, and Bernt Schiele. DeeperCut: A deeper, stronger, and faster multi-person pose estimation model. In *European Conference on Computer Vision*, pages 34–50. Springer, 2016.
2. Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017.
3. Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *European Conference on Computer Vision*, pages 483–499. Springer, 2016.
4. Alexander Mathis, Pranav Mamidanna, Kevin M Cury, Taiga Abe, Venkatesh N Murthy, Mackenzie Weygandt Mathis, and Matthias Bethge. Deeplabcut: markerless pose estimation of user-defined body parts with deep learning. *Nature neuroscience*, 21:1281–1289, 2018.
5. Talmo D Pereira, Diego E Aldarondo, Lindsay Willmore, Mikhail Kislin, Samuel S-H Wang, Mala Murthy, and Joshua W Shaevitz. Fast animal pose estimation using deep neural networks. *Nature methods*, 16(1):117, 2019.
6. Jacob M Graving, Daniel Chae, Hemal Naik, Liang Li, Benjamin Koger, Blair R Costelloe, and Iain D Couzin. Deepposekit, a software toolkit for fast and robust animal pose estimation using deep learning. *eLife*, 8:e47994, oct 2019. ISSN 2050-084X. doi: 10.7554/eLife.47994.
7. Silvia Zuffi, Angjoo Kanazawa, Tanja Berger-Wolf, and Michael Black. Three-d safari: Learning to estimate zebra pose, shape, and texture from images "in the wild". In *ICCV*. IEEE Computer Society, 08 2019.
8. Mackenzie Weygandt Mathis and Alexander Mathis. Deep learning tools for the measurement of animal behavior in neuroscience. *Current Opinion in Neurobiology*, 60:1 – 11, 2020. ISSN 0959-4388. doi: https://doi.org/10.1016/j.conb.2019.10.008. Neurobiology of Behavior.
9. Alexander Mathis, Mert Yüksekgönül, Byron Rogers, Matthias Bethge, and Mackenzie W Mathis. Pretraining boosts out-of-domain robustness for pose estimation. *arXiv preprint arXiv:1909.11229*, 2019.
10. Alexander Mathis and Richard A. Warren. On the inference speed and video-compression robustness of deeplabcut. *bioRxiv*, 2018. doi: 10.1101/457242.
11. Lukas von Ziegler, Oliver Sturman, and Johannes Bohacek. Big behavior: challenges and opportunities in a new era of deep behavior profiling. *Neuropsychopharmacology*, pages 1–12, 2020.
12. Brandon J. Forys, Dongsheng Xiao, Pankaj Gupta, and Timothy H. Murphy. Real-time selective markerless tracking of forepaws of head fixed mice using deep neural networks. *eNeuro*, 7(3), 2020. doi: 10.1523/ENEURO.0096-20.2020.
13. Vilim Štih, Luigi Petrucco, Andreas M Kist, and Ruben Portugues. Stytra: An open-source, integrated system for stimulation, tracking and closed-loop behavioral experiments. *PLoS computational biology*, 15(4):e1006699, 2019.
14. Jens Schweihoff, Martin K Schwarz, Laura A Ewell, Matvey Loshakov, Laura Kück, and Irina Pavlova. DeepLabStream: Closing the loop using deep learning-based markerless, real-time posture detec-tion. page 15.
15. Gonçalo Lopes, Niccolò Bonacchi, João Frazão, Joana P. Neto, Bassam V. Atallah, Sofia Soares, Luís Moreira, Sara Matias, Pavel M. Itskov, Patrícia A. Correia, Roberto E. Medina, Lorenza Calcaterra, Elena Dreosti, Joseph J. Paton, and Adam R. Kampff. Bonsai: An event-based framework for processing and controlling data streams. *Frontiers in Neuroinformatics*, 9, April 2015. ISSN 1662-5196. doi: 10.3389/fninf.2015.00007.

16. Jonny L. Saunders and Michael Wehr. Autopilot: Automating behavioral experiments with lots of Raspberry Pis. Preprint, Animal Behavior and Cognition, October 2019.
17. Keisuke Sehara, Viktor Bahr, Ben Mitchinson, Martin J. Pearson, Matthew Evan Larkum, and Robert N S Sachdev. Fast, flexible closed-loop feedback: Tracking movement in "real-millisecond-time". *eNeuro*, 6, 2019.
18. Yipu Zhao, Wenkai Ye, and Patricio A. Vela. Low-latency visual slam with appearance-enhanced local map building. *2019 International Conference on Robotics and Automation (ICRA)*, pages 8213–8219, 2019.
19. Mattia Privitera, Kim David Ferrari, Lukas M von Ziegler, Oliver Sturman, Sian N Duss, Amalia Floriou-Servou, Pierre-Luc Germain, Yannick Vermeiren, Matthias T Wyss, Peter P De Deyn, Bruno Weber, and Johannes Bohacek. A complete pupillometry toolbox for real-time monitoring of locus coeruleus activity in rodents. *Nature protocols*, 15(8):2301—2320, August 2020. ISSN 1754-2189. doi: 10.1038/s41596-020-0324-6.
20. Tanmay Nath, Alexander Mathis, An Chi Chen, Amir Patel, Matthias Bethge, and Mackenzie W Mathis. Using deeplabcut for 3d markerless pose estimation across species and behaviors. *Nature protocols*, 14:2152–2176, 2019.
21. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
22. Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2018.
23. Parker Hill, Animesh Jain, Mason Hill, Babak Zamirai, Chang-Hong Hsu, Michael Laurenzano, Scott A. Mahlke, Lingjia Tang, and Jason Mars. Deftnn: Addressing bottlenecks for dnn execution on gpus via synapse vector elimination and near-compute data fission. *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 786–799, 2017.
24. Goncalo Lopes, Karolina Farrell, Edward A. B. Horrocks, Chi-Yu Lee, Mai M. Morimoto, Tomaso Muzzu, Amalia Papanikolaou, Fábio Ribeiro Rodrigues, Thomas Wheatcroft, Stefano Zucca, Samuel G. Solomon, and Aman B. Saleem. Bonvision – an open-source software to create and control visual environments. *bioRxiv*, 2020.
25. Francesc Alted, Valentin Haenel, and Blosc Development Team. Python-blosc.
26. Nvidia. Exclusive Education Discount on the Jetson TX2 Developer Kit.
27. Adafruit. Raspberry Pi 4 Model B - 2 GB RAM.
28. Christina K Kim, Avishek Adhikari, and Karl Deisseroth. Integration of optogenetics with complementary methodologies in systems neuroscience. *Nature Reviews Neuroscience*, 18:222–235, 2017.
29. Selmaan N Chettih and Christopher D. Harvey. Single-neuron perturbations reveal feature-specific competition in v1. *Nature*, 567:334 – 340, 2019.
30. Mehrdad Jazayeri and Arash Afraz. Navigating the neural space in search of the neural code. *Neuron*, 93:1003–1014, 2017.
31. Mykhaylo Andriluka, Umar Iqbal, Eldar Insafutdinov, Leonid Pishchulin, Anton Milan, Juergen Gall, and Bernt Schiele. Posetrack: A benchmark for human pose estimation and tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5167–5176, 2018.
32. Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.
33. Sven Kreiss, Lorenzo Bertoni, and Alexandre Alahi. Pifpaf: Composite fields for human pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11977–11986, 2019.
34. Guanghan Ning, Jian Pei, and Heng Huang. Lighttrack: A generic framework for online top-down human pose tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 1034–1035, 2020.
35. Long Chen, Haizhou Ai, Rui Chen, Zijie Zhuang, and Shuang Liu. Cross-view tracking for multi-human 3d pose estimation at over 100 fps, 2020.
36. Valentin Bazarevsky, Yury Kartynnik, Andrey Vakunov, Karthik Raveendran, and Matthias Grundmann. Blazeface: Sub-millisecond neural face detection on mobile gpus. *arXiv preprint arXiv:1907.05047*, 2019.
37. Qianru Zhang, Meng Zhang, Tinghuan Chen, Zhifei Sun, Yuzhe Ma, and Bei Yu. Recent advances in convolutional neural network acceleration. *Neurocomputing*, 323:37–51, 2019.
38. John W. Krakauer, Asif A. Ghazanfar, Alex Gomez-Marin, Malcolm A. MacIver, and David Poeppel. Neuroscience Needs Behavior: Correcting a Reductionist Bias. 93(3): 480–490. ISSN 0896-6273. doi: 10.1016/j.neuron.2016.12.041.
39. Sandeep Robert Datta, David J. Anderson, Kristin Branson, Pietro Perona, and Andrew Leifer. Computational Neuroethology: A Call to Action. 104(1):11–24. ISSN 0896-6273. doi: 10.1016/j.neuron.2019.09.038.
40. Praneet C. Bala, Benjamin R. Eisenreich, Seng Bum Michael Yoo, Benjamin Y. Hayden, Hyun Soo Park, and Jan Zimmermann. OpenMonkeyStudio: Automated Markerless Pose Estimation in Freely Moving Macaques. page 2020.01.31.928861. doi: 10.1101/2020.01.31.928861.
41. Kunal Sahasrabuddhe, Aamir A Khan, Aditya P Singh, Tyler M Stern, Yeena Ng, Aleksandar Tadić, Peter Orel, Chris LaReau, Daniel Pouzzner, Kurtis Nishimura, Kevin M Boergens, Sashank Shivakumar, Matthew S Hopper, Bryan Kerr, Mina-Elraheb S Hanna, Robert J Edgington, Ingrid McNamara, Devin Fell, Peng Gao, Amir Babaie-Fishani, Sampsa Veijalainen, Alexander V Klekachev, Alison M Stuckey, Bert Luyssaert, Takashi Daniel Yoshida Kozai, Chong Xie, Vikash Gilja, Bart Dierickx, Yifan Kong, Malgorzata Straka, Harbaljit S Sohal, and Matthew R Angle. The Argo: A 65,536 channel recording system for high density neural recording in vivo. page 2020.07.17.209403. doi: 10.1101/2020.07.17.209403.
42. Teresa M Findley, David G Wyrick, Jennifer L Cramer, Morgan A Brown, Blake Holcomb, Robin Attey, Dorian Yeh, Eric Monasevitch, Nelly Nouboussi, Isabelle Cullen, Jeremea Songco, Jared F King, Yashar Ahmadian, and Matthew C Smear. Sniff-synchronized, gradient-guided olfactory search by freely moving mice. page 2020.04.29.069252. doi: 10.1101/2020.04.29.069252.
43. Joshua H. Siegle, Aarón Cuevas López, Yogi A. Patel, Kirill Abramov, Shay Ohayon, and Jakob Voigts. Open ephys: an open-source, plugin-based platform for multichannel electrophysiology. *Journal of neural engineering*, 14 4:045003, 2017.
44. Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.
45. Mackenzie Weygandt Mathis, Alexander Mathis, and Naoshige Uchida. Somatosensory cortex plays an essential role in forelimb motor adaptation in mice. *Neuron*, 93 (6):1493 – 1503.e6, 2017. ISSN 0896-6273. doi: https:

//doi.org/10.1016/j.neuron.2017.02.049.
46. Alexander B. Jung, Kentaro Wada, Jon Crall, Satoshi Tanaka, Jake Graving, Christoph Reinders, Sarthak Yadav, Joy Banerjee, Gábor Vecsei, Adam Kraft, Zheng Rui, Jirka Borovec, Christian Vallentin, Semen Zhydenko, Kilian Pfeiffer, Ben Cook, Ismael Fernández, François-Michel De Rainville, Chi-Hung Weng, Abner Ayala-Acevedo, Raphael Meudec, Matias Laporte, et al. imgaug. https://github.com/aleju/imgaug, 2020. Online; accessed 01-Feb-2020.
47. Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.
48. Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: https://doi.org/10.1038/s41592-019-0686-2.
49. G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.
50. Michael M. McKerns, Leif Strand, Tim Sullivan, Alta Fang, and Michael A. G. Aivazis. Building a framework for predictive science. *CoRR*, abs/1202.1056, 2012.
51. Gary Kane and Mackenzie W. Mathis. Camera control: record video and system timestamps from imaging source usb3 cameras. Aug 2019. doi: 10.5281/zenodo.3360725.
52. Wes McKinney. Data structures for statistical computing in python. 445:51–56.
53. Hadley Wickham, Mara Averick, Jennifer Bryan, Winston Chang, Lucy McGowan, Romain François, Garrett Grolemund, Alex Hayes, Lionel Henry, Jim Hester, Max Kuhn, Thomas Pedersen, Evan Miller, Stephan Bache, Kirill Müller, Jeroen Ooms, David Robinson, Dana Seidel, Vitalie Spinu, Kohske Takahashi, Davis Vaughan, Claus Wilke, Kara Woo, and Hiroaki Yutani. Welcome to the Tidyverse. 4 (43):1686. ISSN 2475-9066. doi: 10.21105/joss.01686.
54. Hadley Wickham. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. ISBN 978-3-319-24277-4.
55. Claus O. Wilke. *Ggridges: Ridgeline Plots in 'Ggplot2'*.

Supplemental Figure

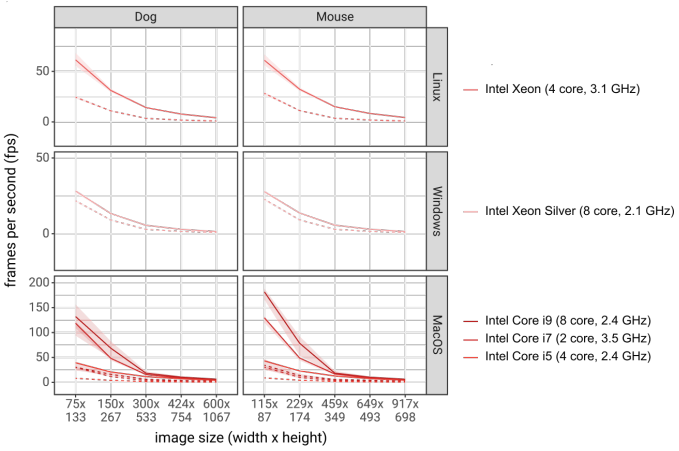

Figure. 7. Inference speed of different networks with different image sizes with different system configurations, using CPUs for inference. Solid lines are tests using DLC-MobileNetV2-0.35 networks and dashed lines using DLC-ResNet-50 networks. Shaded regions represent 10th-90th percentile of the distributions. N = 1000-10000 observations per point.