Neuroscience Cloud Analysis As a Service

Taiga Abe[124], Ian Kinsella[125], Shreya Saxena[1235], Liam Paninski[12345], John P. Cunningham[1235]

[1]Mortimer B. Zuckerman Mind Brain Behavior Institute
[2]Center for Theoretical Neuroscience, Columbia University
[3]Grossman Center for the Statistics of Mind, Columbia University
[4]Department of Neuroscience, Columbia University
[5]Department of Statistics, Columbia University
June 11, 2020

Abstract

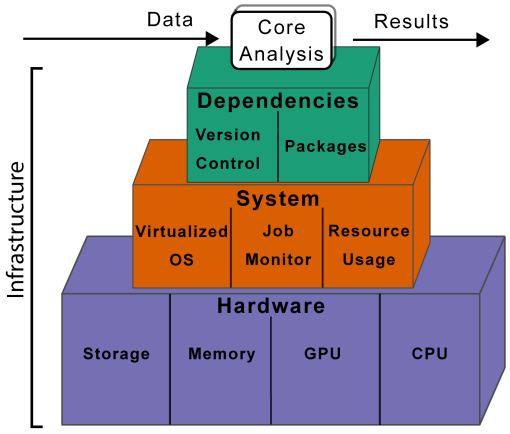
A major goal of computational neuroscience is to develop powerful analysis tools that operate on large datasets. These methods provide an essential toolset to unlock scientific insights from new experiments. Unfortunately, a major obstacle currently impedes progress: while existing analysis methods are frequently shared as open source software, the infrastructure needed to deploy these methods – at scale, reproducibly, cheaply, and quickly – remains totally inaccessible to all but a minority of expert users. As a result, many users can not fully exploit these tools, due to constrained computational resources (limited or costly compute hardware) and/or mismatches in expertise (experimentalists vs. large-scale computing experts). In this work we develop Neuroscience Cloud Analysis As a Service (NeuroCAAS): a fully-managed infrastructure platform, based on modern large-scale computing advances, that makes state-of-the-art data analysis tools accessible to the neuroscience community. We offer NeuroCAAS as an open source service with a drag-and-drop interface, entirely removing the burden of infrastructure expertise, purchasing, maintenance, and deployment. NeuroCAAS is enabled by three key contributions. First, NeuroCAAS cleanly separates tool implementation from usage, allowing cutting-edge methods to be served directly to the end user with no need to read or install any analysis software. Second, NeuroCAAS automatically scales as needed, providing reliable, highly elastic computational resources that are more efficient than personal or lab-supported hardware, without management overhead. Finally, we show that many popular data analysis tools offered through NeuroCAAS outperform typical analysis solutions (in terms of speed and cost) while improving ease of use and maintenance, dispelling the myth that cloud compute is prohibitively expensive and technically inaccessible. By removing barriers to fast, efficient cloud computation, NeuroCAAS can dramatically accelerate both the dissemination and the effective use of cutting-edge analysis tools for neuroscientific discovery.

1 Introduction

In recent years, our field has developed a remarkable variety of recording technologies that measure behavior and neural activity at previously unimaginable scale, producing vast quantities of large and complex data. In parallel, neural data analysis — which aims to build the path from these datasets to scientific insight — has grown into a centrally important and necessary component of modern neuroscience (Paninski and Cunningham, 2018).

The result of this growth is an explosion in the complexity of neural data analysis techniques. Historically, scientific analysis algorithms were typically implemented as isolated code scripts that performed straightforward computations on moderately sized data and were often run on standard desktops or locally networked servers. These code scripts were typically shared with the research community in an ad hoc fashion (printed in appendices or as pseudocode, shared via email, etc.). In stark contrast, modern neural data analysis routinely involves video processing algorithms (Pnevmatikakis et al., 2016, Pachitariu et al., 2017, Mathis et al., 2018, Zhou et al., 2018, Giovannucci et al., 2019) , deep neural networks (Gao et al., 2016, Batty et al., 2016, Lee et al., 2017, Parthasarathy et al., 2017, Mathis et al., 2018, Pandarinath et al., 2018, Giovannucci et al., 2019), sophisticated graphical models (Yu et al., 2009, Wiltschko et al., 2015, Gao et al., 2016), and/or other machine learning tools (Pachitariu et al., 2016, Lee et al., 2017). Thus, modern core analyses depend extensively upon underlying *infrastructure*, including

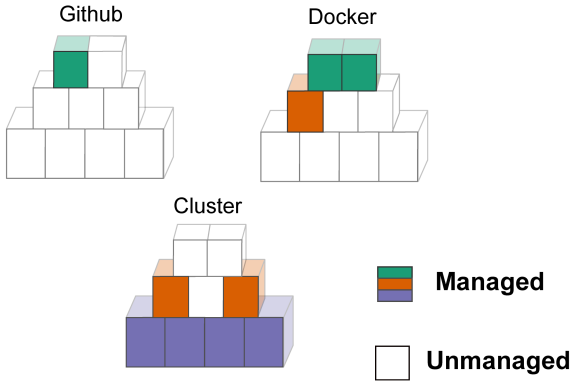
A. Structure of a data pipeline



B. Common problems in neuroscience data analysis

User-Side	Developer-Side
• Pipeline restructuring around software updates • Time and knowledge cost of installation • Manual scaling to multiple machines	• Volatile dependencies • Coordinated development across multiple operating systems
• Interruptions from system updates • Data loss due to human error	• Stability/efficiency at scale • Difficult to test in relevant environment
• Compatibility issues with local hardware • Insufficient resources for dataset volume • Delays due to resource sharing with other users	• Accomodation to hardware limitations of end user

C. Generic management tools



D. Field management standards

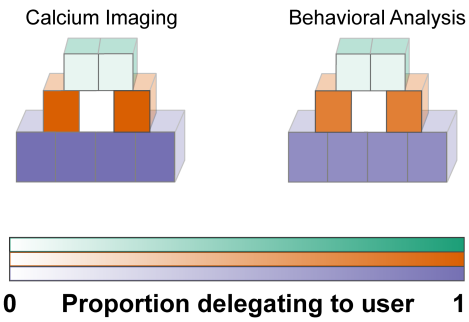


Figure 1: **Data analysis infrastucture.** A. Core analysis code sits atop a pyramid of infrastructure (including both software and hardware) that must be installed and maintained to make analysis viable. B. A number of common problems arise at each layer of this infrastructure pyramid; some of these issues are most visible to users, while some are more visible to software developers. C. Many common management tools deal only with one or two layers in the infrastructure pyramid, leaving gaps that users and developers must fill manually or by cobbling systems together. D. Our surveys of common neural data analysis tools for calcium imaging and behavioral analysis indicate that many layers of the infrastructure pyramid are currently not managed by analysis developers, and implicitly delegated to the user's responsibility (see **Methods** for full details).

software libraries such as pytorch and tensorflow (Abadi et al., 2016, Paszke et al., 2019), systems necessary to connect this software to underlying hardware (Merkel, 2014), and precisely built and configured CPU and GPU hardware. Figure 1A details this infrastructure as the necessary – but largely ignored – foundation on which all analyses run (Demchenko et al., 2013, Jararweh et al., 2016, Zhou et al., 2016).

Major efforts have been made by journals (Donoho, 2010, Hanson et al., 2011, `https://www.nature.com/news/code-share-1.16232`) and funding agencies (Carver et al., 2018) to encourage the sharing of core analysis code. However, these important efforts still ignore the vast majority of the infrastructure (Figure 1A) required to complete modern neural data analysis. As a first example, though many algorithms require a specific operating system for their proper use, only the rarest offer an operating system bundled with their release of code (as is possible via containerization). It thus falls entirely on the user to match analysis technique to operating system. As another example, many neural data analysis algorithms require meaningful use of graphics processing units (GPUs) for their operation, but none to our knowledge offer a bundled GPU with freely available analysis code. (As we will see, this example is quite a bit less absurd than it may appear; indeed this paper offers a solution.) Despite calls to improve standards of practice in the field (Vogelstein et al., 2016), and some progress in other

fields such as astronomy, genomics, and high energy physics (Hoffa et al., 2008, Zhou et al., 2016, Chen et al., 2019), there has been little concrete progress in our field towards a solution that provisions or even documents infrastructure at a scientifically acceptable standard.

The implication of this missing infrastructure is a set of problems all too familiar to experimental and computational neuroscientists alike (Figure 1B): both human and financial resources must be spent on hardware setup, configuration and troubleshooting of software and libraries, unexpected interruptions to processing from unreliable systems, constraints due to limited "on-premises" computational resources, and more. Some tools – notably compute clusters (whether in a public or university-owned cloud), versioning websites like github (`https://github.com`), and more recently containerization websites like Docker (Merkel, 2014) – offer various infrastructure components (Figure 1C), but each of these tools requires some expertise to use, and it is nontrivial to combine these components into a complete working infrastructure, sharply limiting the accessibility and preventing the universal adoption of these tools. The unfortunate result of these problems and these partially used toolsets is a hodgepodge of often slipshod infrastructure practices across the literature (Figure 1D; see supporting data in Tables S1, S2).

Far from being simply a headache, these infrastructure issues can have a serious negative impact on the data analysis process. For example, limitations on hardware capacity have real consequences for the performance of analysis algorithms (Radiuk, 2017), and can interfere with analysis performance in ways that preclude the most careful attempts to reproduce the appropriate infrastructure. Further, the difficulty of configuring analysis infrastructure drives yet more divergence (Demchenko et al., 2013, Monajemi et al., 2016), and can make errors extremely difficult to detect, let alone repair. This challenge persists even if the original developer provides clear and comprehensive instructions for use (which is rarely the case and an ongoing challenge across science (Zhao and Deek, 2005, Stodden et al., 2018, Raff, 2019). Emphasizing this challenge, a recent study of the machine learning literature observed that although local compute clusters address the issue of practical resource availability, *none* of the studies that made use of a compute cluster were reproducible. That paper indicated two causes: the lack of unifying infrastructure frameworks for distributed computing, and lack of details in the description of infrastructure resources (Raff, 2019).

Suboptimal infrastructure has meaningful and urgent scientific consequences. First, scientific progress suffers when researchers must make a choice between a cutting-edge data analysis (with some nontrivial cost and difficulty of use) versus a simpler (though less insightful) traditional technique. Second, reproducibility of results, one hallmark of rigorous scientific discovery, is extremely difficult when the infrastructure components needed to reproduce a result are neither described fully nor readily recreated (Crook et al., 2013, Stodden et al., 2018, Krafczyk et al., 2019, Raff, 2019). Third, scientific results become less reliable as opportunities for errors creep into poorly understood analysis pipelines. Taken together, the current standards for (re)creating and then maintaining data analysis infrastructure is an unaddressed bottleneck in the field. Furthermore, this burden often falls on trainees who are neither scientifically rewarded (`https://chanzuckerberg.com/rfa/essential-open-source-software-for-science/`) nor specifically instructed on how to build, configure, and install infrastructure. We term this problematic conventional model of creating analysis infrastructure *Infrastructure-as-Grad-Student* (IaGS). The IaGS status quo fails any reasonable standard of scientific rigor and impedes both scientific training and the broad use of valuable analytical tools.

Of course, the infrastructure issues described here are not specific to neuroscience. Looking to sectors that deploy software as a service at industrial scale, there has been in the last few years remarkable progress in the *Infrastructure-as-Code* (IaC) paradigm (Morris, 2016): a practice and emerging toolset that endows infrastructure creation and deployment with the replicability and determinism that we have come to expect from software. IaC first details, in code, the various infrastructure resources (each box in Figure 1A) that constitute a project, and the ways that they should be coordinated. This code is then automatically deployed into the equivalent functioning software, system, and hardware resources, analogous to how standard software code is compiled into a program and executed. IaC offers tremendous benefits to scientific data analysis, and neuroscience in particular, but there has been no previous effort to extend this new technology for our field.

In response, we developed Neuroscience Cloud Analysis as a Service (NeuroCAAS), a platform that hosts neural data analysis algorithms and builds all the infrastructure (via IaC) upon which they depend. The result is a drag-and-drop interface for neural data analysis: experimentalists and computational neuroscientists can log on to the NeuroCAAS website, set some parameters for an analysis, and drop their neural or behavioral data onto the platform. Infrastructure is then automatically provisioned, deployed on the cloud, and used to produce

analysis results which are returned to the user, after which the infrastructure is automatically dissolved.

In this work we combine the modern concepts of resource virtualization, immutable analysis environments, and analysis blueprints to automate the data analysis process, building a platform that fully dissociates science from engineering. With NeuroCAAS, as elsewhere, the scientific user bears the responsibility of using analyses thoughtfully. Thus, critically, NeuroCAAS neither removes nor attempts to lessen the scientific rigor needed for neural data analysis; instead, it abstracts away all nonscientific infrastructure.

Next, we manifest NeuroCAAS as a fully hosted, secure service on the publicly accessible cloud. This point warrants emphasis, as it diverges starkly from the traditional scientific practice: NeuroCAAS is *not* a platform design or suggestion that the reader can attempt to recreate on their own; instead, NeuroCAAS is offered as an open source, fully hosted platform available for immediate use, via a website that users can sign up to and then drag and drop their data for analysis (`www.neurocaas.org`). NeuroCAAS offers a growing set of popular but challenging-to-implement data analysis algorithms, and has been heavily tested by users in the computational and experimental neuroscience community. As natural consequences of its design, data analyses hosted on NeuroCAAS are automatically reproducible end-to-end, and seamlessly scalable across datasets.

Finally, we conducted experiments that compare NeuroCAAS's implementation to conservative "on-premises" benchmarks, measuring time and financial cost. Because it uses state-of-the-art compute resources, but only when needed, NeuroCAAS is both cheaper and faster than local computing across a wide range of common data analyses, dispelling a common misconception about the viability of such a platform (§2.3). Indeed, processing on NeuroCAAS is sufficiently inexpensive that we can offer this platform as a free service for many users, sharply increasing the accessibility and reach of modern analysis tools (§2.4). By addressing these critical and growing issues in neuroscience research, NeuroCAAS paves the way to the next generation of rigorous and powerful data analysis in our field.

2 Results

2.1 NeuroCAAS replaces Infrastructure-as-Grad-Student with Infrastructure-as-Code

The challenges described above make IaGS difficult to address through policy changes or enforcement of particular programming practices, as has been suggested for infrastructure in other contexts (Buckheit and Donoho, 1995, Crook et al., 2013, Stodden et al., 2018). In response, we designed an infrastructure *service* (*not* a policy or new software library) that uses ideas from modern computing research to entirely remove infrastructure development and management from a user's data analysis process. This service – NeuroCAAS– offers a qualitatively different analysis model (Figure 2) to what is standard in the field.

To analyze data on NeuroCAAS, users simply choose from a list of supported methods (see Table 1, or better the continuously updating list at `www.neurocaas.org`), select a corresponding configuration file and modify it as needed, and then drag-and-drop all datasets to be processed into the appropriate box on the website (Figure 2, top left). No further user input is needed: NeuroCAAS first detects the submission event (dataset and configuration file submission), then recruits a job manager to programatically create and manage all infrastructure and autonomously execute analysis, thereby providing highly scalable and completely reproducible computational processing via IaC (Figure 2, right). Analysis outputs (including live status logs and a complete description of analysis parameters) are then delivered back to the user, and finally the provisioned infrastructure is dissolved automatically when data processing is complete (Figure 2, bottom). We reiterate that throughout the whole process, the user never has to maintain any sophisticated hardware, read or install any complicated software, or troubleshoot any software dependency or operating system compatibility issues.

NeuroCAAS simultaneously resolves all of the challenges underlying IaGS, uniformly across all potential users of a given algorithm. First, to solve the challenge of practical resource availability (§2.1.1), we use virtualized public cloud resources that are accessible at scale and effectively limitless. Second, for reproducibility (§2.1.2), NeuroCAAS saves analyses into a blueprint that can be versioned, copied, and redeployed back into fully functioning analyses. Finally, to address configuration difficulty and analysis at scale, we autonomously configure all resources into an *immutable* analysis environment (§2.1.3). These three innovations introduce the Infrastructure as Code, or IaC (Morris, 2016) paradigm to neuroscience.

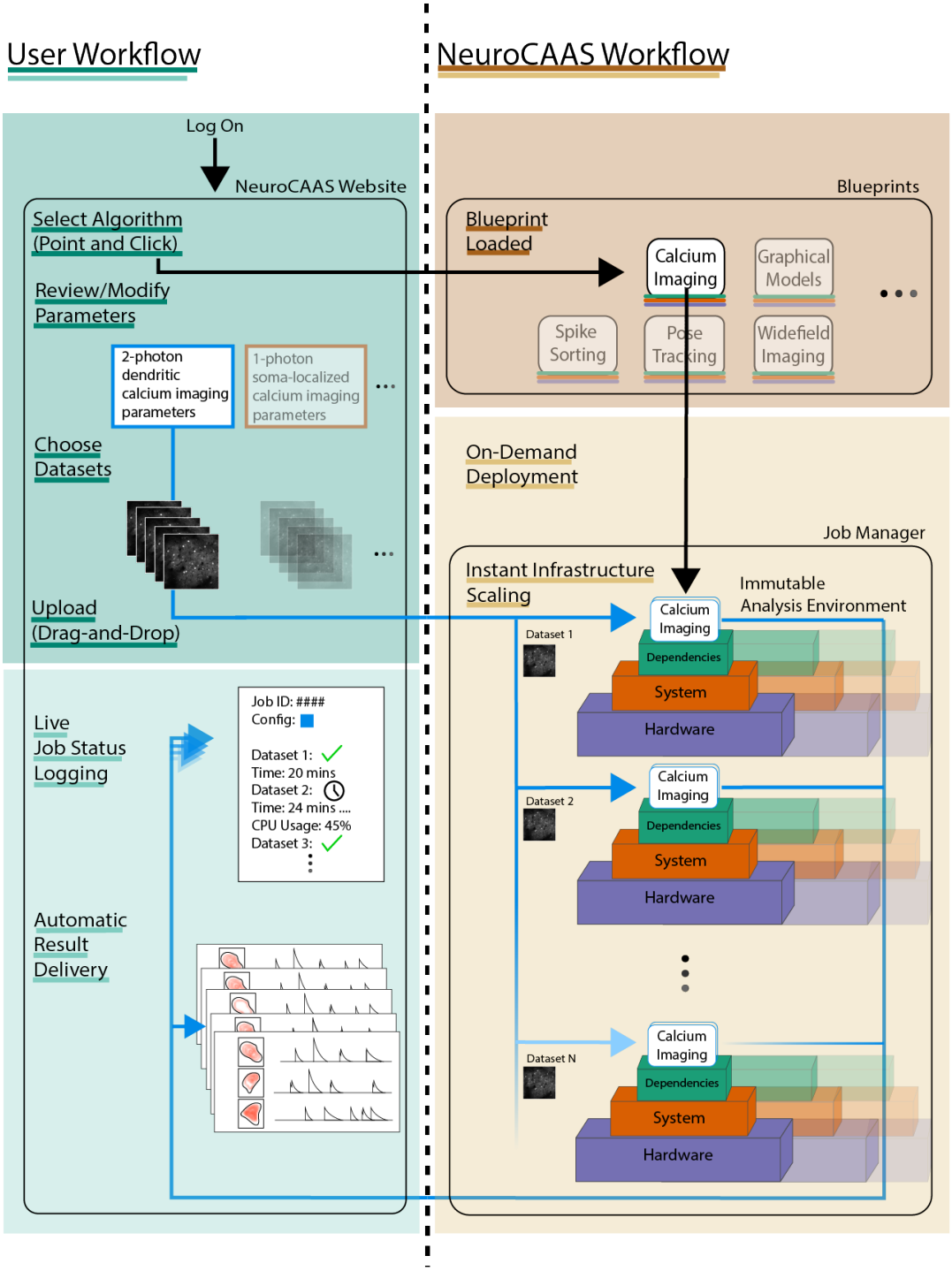


Figure 2: **Overview of NeuroCAAS**. Left indicates the user's experience; right indicates the work that NeuroCAAS performs. The user begins by choosing from the algorithms hosted on the NeuroCAAS website, then choosing a corresponding configuration file (and modifying this file as needed). The user then uploads dataset(s) and the configuration file. NeuroCAAS detects this upload event, reads the configuration file, and deploys the requested algorithm using an infrastructure blueprint (including the necessary software, compute environment resources, and hardware) that the developer previously specified as optimal for the user's needs. The deployed blueprint is read by NeuroCAAS's job manager, and built into an immutable analysis environment. Multiple analysis environments may be deployed in parallel if the user uploads multiple datasets. The deployed resources persist only as necessary, and results, as well as diagnostic information, are automatically routed back to the user. Still image of neural data adapted from `http://neurofinder.codeneuro.org`.

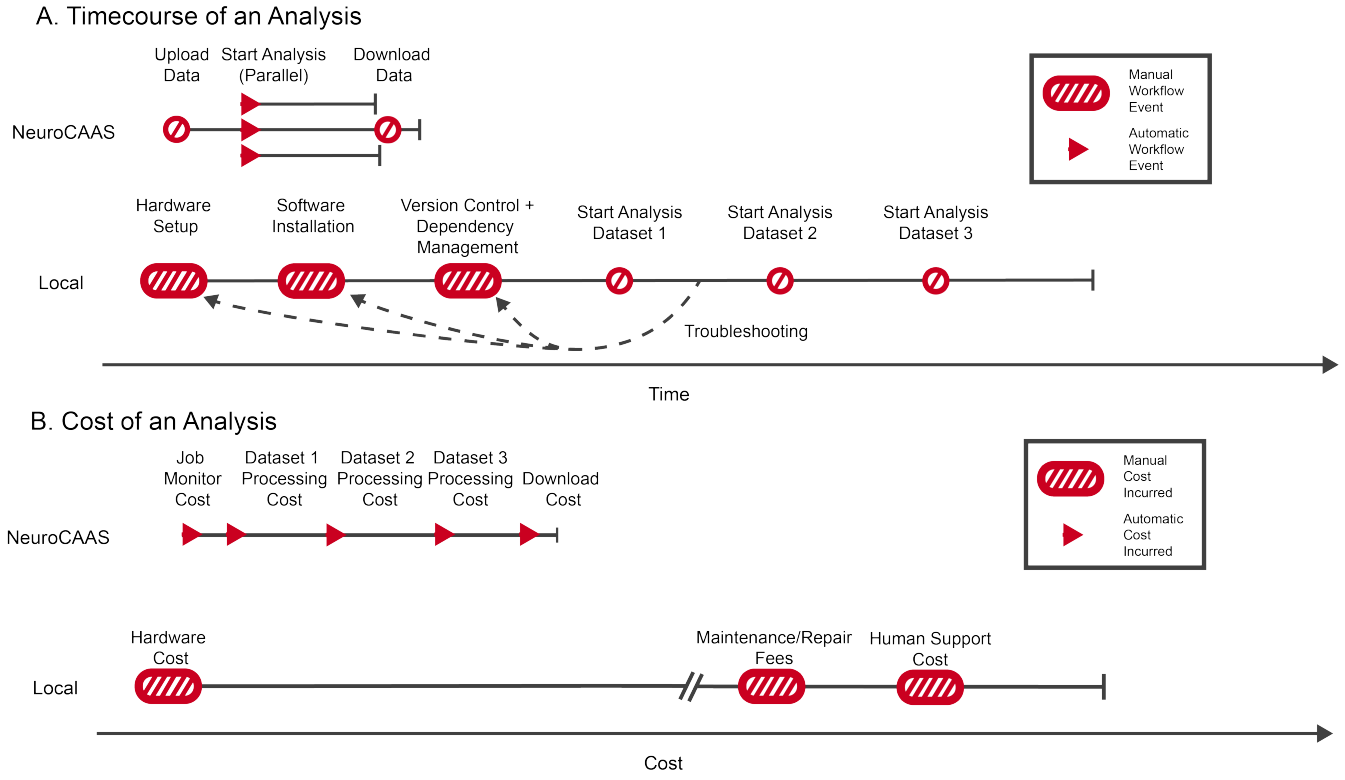


Figure 3: **Infrastructure-as-grad-student vs infrastructure-as-code.** A. Local processing requires a number of time-consuming steps from the user (hardware setup, software installation and maintenance, etc.) before any analyses are run. Then typically analyses of large datasets are run serially (due to resource constraints), leading to longer processing times. On NeuroCAAS, user interaction is only required at the beginning of the analysis (to upload the data), then NeuroCAAS processes the data using large-scale parallel compute resources, leading to faster overall processing times. B. On NeuroCAAS, some costs are incurred with each analysis run: the user must upload the datasets (incurring a small job monitor cost), and then each dataset incurs some compute cost. For local processing, the bulk of the costs are paid upfront, in purchasing hardware; then additional labor costs are incurred for maintenance, support, and usage of limited local resources. If the per-dataset costs are low and the total number of datasets to be processed is limited then NeuroCAAS can lead to significantly smaller total costs than local processing.

2.1.1 Virtualization

For data analyses provided on NeuroCAAS, users never provision their own infrastructure. By utilizing fully virtualized infrastructure, NeuroCAAS makes the exact infrastructure resources used by the developer of an analysis available to all users. Furthermore, this access is achieved without sacrificing the freedom of developers to allocate the resources most appropriate for their needs. NeuroCAAS uses virtualized resources on a commercial public cloud (currently Amazon Web Services).

The public cloud is the right choice for three reasons. First, the public cloud provides a vast array of virtualization options that can support the diverse needs of different analyses. Second, the cloud operates at a scale and level of accessibility that supports many global, industrial-grade applications, meaning that the volume of virtualized resources that can be simultaneously deployed is practically unlimited by physical resource availability. Finally, we made use of cloud tools and frameworks to build a high-level job manager that coordinates automatic assembly, destruction, and communication with analysis-relevant computing infrastructure.

2.1.2 Reproducibility via Blueprints

NeuroCAAS makes analyses immediately reproducible by specifying the identity and configuration of all relevant infrastructure resources in a *blueprint* that captures all aspects of analysis-related infrastructure, describing resources as disparate as virtualized hardware, job managers, and relevant user permissions in a comprehensive

document. The NeuroCAAS blueprint is a dictionary of key-value pairs, implemented as a JSON object, with the keys representing different virtualized infrastructure components, and values representing their specification.

NeuroCAAS blueprints have immediate value for standards of reproducible research in our field. A great deal of ink has been spilled over the last couple decades on the importance of reproducible research (Buckheit and Donoho (1995), others) — but in many cases research remains frustratingly non-reproducible (Crook et al., 2013, Raff, 2019, Stodden et al., 2018). NeuroCAAS sidesteps all of the typical barriers to reproducible research by tightly coupling resources to documentation via blueprints: virtualized resources are provisioned and configured *directly* into functioning analyses from these blueprints without any further manual input (Figure 2, top right into bottom right), meaning that any working analysis is by necessity reproducible. What follows is extreme portability: when provided with a blueprint, the exact analysis employed by a particular user to analyze their data is instantly available to anyone else.

Developers can easily extend analyses already on NeuroCAAS by instantiating them from blueprints, adjusting them as required, and saving the relevant changes. All changes to blueprints are version controlled via git, and generate a unique id that can be referenced within jobs (see Figure 2), ensuring that the infrastructure used to analyze a particular data set is never lost, regardless of ongoing development to the analysis itself.

2.1.3 Immutable Analysis Environments

NeuroCAAS serves analyses to users by means of *immutable analysis environments*. In an immutable analysis environment, all infrastructure is configured independently of the user, and cannot be altered during the process of analyzing data. The requirement of immutability means that the hardware and operating system of an analysis are set up without user input, but more strictly that installation, package management, and critically analysis workflow are dictated by *developer* choices. We provide developers an interface to create the immutable analysis environment without requiring additional knowledge of cloud architecture (see `neurocaas.org` for details).

To stabilize data analysis workflow, immutable analysis environments are isolated from user interaction mid-analysis. Instead, users simply input data and analysis parameters to the relevant NeuroCAAS job manager, which parses these inputs and initiates processing in the immutable analysis environment, autonomously executing the necessary computations to generate a result (Figure 2, bottom right panel). The NeuroCAAS job manager collects status messages from analysis output and environment diagnostics throughout processing, and returns them to the user live, allowing them to monitor ongoing progress as they wait for their analysis results. At the end of processing, users receive with their results a certificate detailing the blueprint, data, and analysis parameters that fully specify the analyses that were run (Figure 2, bottom left panel).

Serving immutable analysis environments has several benefits. First, immutable analysis environments provide scalability: the NeuroCAAS job manager uses the information in a blueprint to rapidly instantiate immutable analysis environments on-demand. NeuroCAAS's job manager automatically detects input batching and instantiates the corresponding number of identical immutable analysis compute environments (Figure 2, bottom right), distributing batched data across these environments, aggregating results for the user afterwards, and destroying the immutable analysis environments once they are no longer needed. In the era of big data and powerful analyses it is common to use the same configuration parameters to analyze many datasets simultaneously, providing an ideal use case for NeuroCAAS. On typical local infrastructure, this type of large-scale analysis would be very time consuming (or not feasible at all) and would often require a great deal of laborious manual oversight and book-keeping.

Second, immutable analysis environments minimize the potential for analysis errors. By closing the immutable analysis environment from direct input, we eliminate the possibility of concurrent and irrelevant processes interfering with analysis stability, and minimize the effects of software updates and spontaneous hardware failure-long-standing frustrations in the scientific computing literature that can have considerable effects upon the results of an analysis (Crook et al., 2013, Stodden et al., 2018) (see **Methods** for details). Additionally, the degree of predictability offered by immutable design makes developer bugs much more easy to find and fix, as they must be a result of incorrect input management (and not, for example, due to the user running a different version of some software package).

The principle of immutability has a long history in computer science (Bloch, 2008) and is readily achievable on existing popular algorithms in neuroscience (§2.3). Moreover, immutable analysis environments support a long line of work arguing for increasing automation in data analysis as opposed to interactivity. Advocates have argued that

automation provides scientists the opportunity to better study their data and methods (Tukey, 1962), and that automation is fundamental to allow modern data analysis to achieve reliable and reproducible results at massive scale (Waltz and Buchanan, 2009, Monajemi et al., 2016).

2.2 NeuroCAAS: hammer, screwdriver, or what?

The modern computing era brings neuroscientists a large and increasingly powerful toolkit for improved data analysis. What sort of tool then is NeuroCAAS, and what problems does it (and does it not) address?

What NeuroCAAS *does.* As discussed throughout, we emphasize that NeuroCAAS offers automated *infrastructure.* Its purpose is to remove the burden of setting up and maintaining software, hardware, operating system, etc. from the process of neural data analysis. The singular goal of NeuroCAAS is to remove Infrastructure-as-Grad-Student from neuroscience, freeing users from having to think about infrastructure choices, from having to purchase and set up infrastructure, and from having to troubleshoot infrastructure; all users of those methods then are provided that same infrastructure via NeuroCAAS.

What NeuroCAAS *does not do.* First, NeuroCAAS does not aim to improve the scientific use of neural data analysis algorithms. For example, if a user has data that is incorrectly formatted for a particular algorithm, the same error will happen with NeuroCAAS as it would with conventional usage. Garbage-in-garbage-out is as true with NeuroCAAS as with any other software platform.

Second, NeuroCAAS is not a scientific workflow management system. Workflow management systems such as Datajoint (Yatsenko et al., 2015) and the Common Workflow Language (Amstutz et al., 2016) codify the sequential steps (or workflow) that make up a data analysis, ensuring data integrity and provenance. While the design of NeuroCAAS incorporates some level of data and task management, our main goal is not to schematize the workflow within a given algorithm, or to ensure its compatibility within other data pipelines. Instead, our goal is to organize, document, and provide the work that must be done to make a *given* data analysis functional, efficient, and accessible. This goal is orthogonal and complementary to applications that explicitly provide tools to make rigorous the data pipeline from experiment through to database and data processing.

Third, NeuroCAAS is not unstructured access to cloud computational resources. The concept of immutable analysis environments should clarify this fact: NeuroCAAS serves a set of analyses that are configured to a particular specification, as established by the algorithm developer. If a user is looking for computational resources to run an algorithm that is not yet supported on NeuroCAAS, they can either request to add the algorithm to NeuroCAAS, or adapt existing cloud services to their needs (Jupyter et al., 2018, Chen et al., 2019).

2.3 NeuroCAAS is faster and cheaper than local "on-premises" processing

NeuroCAAS offers a number of major advantages over local processing, in terms of ease of use, reproducibility, and scalability whether we consider local infrastructure to be a personal workstation or a locally available cluster. However, since NeuroCAAS is based on a cloud computing architecture, one might worry that data transfer times (i.e., uploading and downloading data to and from the cloud) could potentially lead to slower overall results compared to various forms of local processing. Second, one might worry that cloud computing might be more expensive than local processing.

Figure 3 qualitatively illustrates the accumulating inefficiencies of time and cost in every IaGS pipeline. IaGS begins with a number of time-consuming steps from the user, including hardware acquisition, hardware setup, and software installation and maintenance (Figure 3A). Another manual user event must then follow, namely the uploading and analysis of a new dataset. While parallel processing is possible, in many cases datasets are run serially. The user sometimes acts as job monitor, waiting for an analysis to complete before (after some delay) starting another analysis. What results from IaGS is gross inefficiency of time and resources. Consider as an alternative IaC-based analyses as delivered by NeuroCAAS: once data is uploaded, parallel infrastructure is instantly created, data is analyzed, and results are served, without any steps required from the user, and without any artificial IaGS delays (Figure 3A, top). Indeed, this gap between IaGS and IaC is mirrored from a cost perspective as well (Figure 3B).

Figure 4 considers this same question quantitatively, comparing NeuroCAAS to processing on local infrastructure resources, priced as a personal workstation (see **Methods**). For the analogous comparisons pricing these resources as a local cluster, see Figure 9. To be (extremely) conservative, we assume local infrastructure is set up already, neglecting all of the time associated with installing and maintaining software and hardware. We use a total

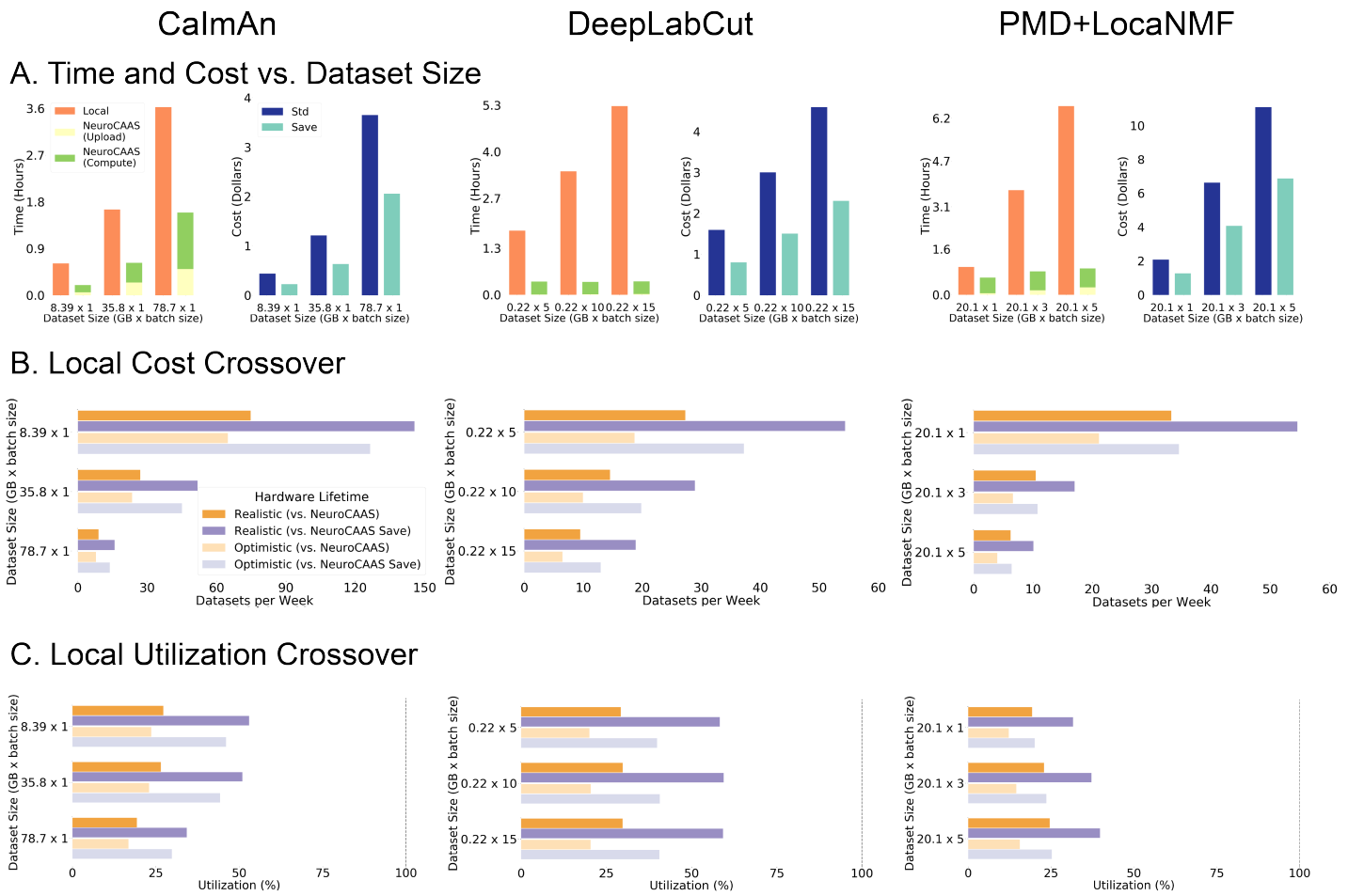


Figure 4: **Quantitative comparison of NeuroCAAS versus local processing for three different analysis pipelines.** A. Simple quantifications of NeuroCAAS performance. Left graphs compare total processing time on NeuroCAAS vs. local infrastructure (orange). NeuroCAAS processing time is broken into two parts: Upload (yellow) and Compute (green). Repeated analyses of data already in NeuroCAAS need only consider Compute times (see main text for details). Right graphs quantify cost of analyzing data on NeuroCAAS with two different pricing schemes: Standard (dark blue) or Save (light blue), offering the same analyses at a reduced price if the approximate duration of an analysis job is known beforehand. B. Cost comparison with local infrastructure. Local Cost Crossover gives the minimum per-week data analysis rate required to justify the cost of a local infrastructure compared to NeuroCAAS. We consider local pricing against both Standard and Save prices, and with Realistic (2 year) and Optimistic (4 year) lifecycle times for local hardware. C. Achieving Crossover Analysis Rates. Local Utilization Crossover gives the minimum utilization required to achieve crossover rates shown in B. Dashed vertical line indicates maximum feasible utilization rate at 100%.

cost of ownership (TCO) metric (Morey and Nambiar, 2009) that includes the purchase cost of local hardware, plus reasonable maintenance costs; see **Methods** for full details. Figure 4 presents time and cost benchmark results on four modern data analysis algorithms hosted on NeuroCAAS: CaImAn, a toolbox for analysis of calcium imaging data (Giovannucci et al., 2019); DeepLabCut (DLC), a method for markerless tracking of pose from behavioral video data (Mathis et al., 2018); Penalized Matrix Decomposition (PMD), a method for denoising and compressing functional imaging data (Buchanan et al., 2018); and Localized Non-negative Matrix Factorization (LocaNMF), a method for demixing widefield calcium imaging data (Saxena et al., 2020). Each analysis presented in Figure 4 highlights a different strength of NeuroCAAS compared to local infrastructure (see Methods for implementation details). Our implementation of CaImAn shows that NeuroCAAS is more efficient than a standard local workstation even without parallelizing data into batches by using state-of-the-art virtualized computing resources. Our implementation of DeepLabCut highlights the scalability benefits of our approach, simultaneously deploying 15 separate GPU-equipped immutable analysis environments to efficiently analyze a batch of data in parallel.

Finally, we used NeuroCAAS to compose PMD and LocaNMF into a single analysis, passing the data sequentially through two separate, independently optimized immutable analysis environments. The resulting analysis shows that multiple analysis pipelines hosted on NeuroCAAS can be chained together without compromising infrastructure towards one or the other, and maintaining efficiency in both cost and time.

Across all algorithms and datasets considered in Figure 4, we found that analyses run on NeuroCAAS were significantly faster than those run on the selected local infrastructure, even accounting for the time taken to stage data to the cloud (Figure 4A, left panes). For DLC and PMD+LocaNMF, the NeuroCAAS compute time was effectively constant across increasing total dataset size, as data was evenly batched into subsets of approximately equal size and each batch was analyzed in its own independent instance; this is a concrete example of the scalability benefits discussed in section 2.1.3. Furthermore, note that NeuroCAAS upload time can be ignored if analyzing data that has already been staged for processing — for example if there is a need to re-process data with an updated algorithm or parameter setting — leading to potential speedups. Finally, in the future processing can be made even more efficient by scheduling the construction of immutable analysis environments ahead of data upload, saving a precious few minutes for the cases of small datasets and time-sensitive analysis paradigms.

Next we turn to cost analyses. Over the range of algorithms and datasets analyzed here, we found that the overall NeuroCAAS analysis cost was on the order of a few US dollars per dataset (Figure 4A, right panels). In addition to our baseline implementation , we also offer an option to run analyses at a significantly lower price (indicated as "Std" and "Save" respectively in the cost barplots in Figure 4), if the user knows ahead of time how long their job is expected to take (i.e. from previous runs of similar data).

We also provided a cost metric that faciliates direct comparison to the cost of an on-premise hardware implementation in each panel of Figure 4B. The Local Cost Crossover is a fair metric to compare NeuroCAAS cost to the TCO of an analogous local infrastructure. Assuming reasonable lifetimes for local hardware, the Local Cost Crossover provides the threshold rate at which a user would have to analyze data in order to merit purchase of a local machine instead of using NeuroCAAS (e.g., in order for a local machine to be cost effective for CaImAn, one must analyze ∼100 datasets of 8.39 GB per week, every week for several years according to the top two bars of Figure 4B, left).

The threshold rates given in Figure 4B do not consider the time taken to process datasets on local infrastructure. To integrate the time and cost of analyzing data locally, we introduce the Local Utilization Crossover: this measures the utilization (proportion of all possible compute time devoted to a given data analysis) required to achieve the rates depicted in Figure 4B. It assumes the analysis times that are given in Figure 4A to calculate these utilization percentages (see **Methods** for details). In practice, maximum attainable utilization is bounded well below 100% by the constraints of the user's time, or the needs of other users on shared cluster resources (see Figure 9). These calculations demonstrate that even without considering all of the IaGS issues that our solution avoids, it is difficult to use local infrastructure more efficiently than NeuroCAAS. For users who would like to benchmark their own infrastructure against NeuroCAAS, we provide a tool to do so (see the instructions at `https://github.com/cunningham-lab/neurocaas`).

2.4 NeuroCAAS is offered as a free service for many users

From Figure 4 we conclude that running analyses on NeuroCAAS is fairly inexpensive: on the order of a few US dollars per dataset for the analysis pipelines considered here. Given this low per-dataset cost, and the major advantages summarized above of NeuroCAAS compared to the current status quo of local processing, we have decided to subsidize the use of NeuroCAAS, as a service to the community. Users do not need to set up any billing information or worry about incurring any costs when trying out NeuroCAAS; in the interest of protecting NeuroCAAS as a non-commercial open-source effort, we cover all costs up to a per-user cap. This removes one final friction point that might slow adoption of NeuroCAAS. Since NeuroCAAS is relatively inexpensive, many users will not hit the cap; thus, for these users, NeuroCAAS is offered as a free service.

3 Discussion

NeuroCAAS integrates scientifically rigorous infrastructure practices via IaC into neuroscience data analysis. By drastically reducing the technical barriers to usage, NeuroCAAS promotes a new standard of replicable, efficient, cutting-edge data analysis for the neuroscience community that smoothly integrates with current development and use practices. Further, NeuroCAAS introduces robust automation methods via our job manager to reliably handle the increasingly large and complex data that have become characteristic of modern approaches to neuroscience, and seamlessly parallelizes analyses over multiple datasets. Finally, we have shown that the scientific virtues of NeuroCAAS are accompanied by increases in efficiency, reducing both the time and cost required to run neuroscience data analyses.

The fundamental choice made by NeuroCAAS is to provide infrastructure as a *service*, such that neither analysis users nor analysis developers have to introduce a new library or framework into their coding practices; rather, NeuroCAAS removes the infrastructure burden entirely. Such a choice is a tradeoff worth making explicit. First, a service provides the user an interface to use the analysis method exactly as intended by the developer, without possibility for the introduction of additional errors in the pipeline, and without setup inefficiencies on the user side. For the developer, NeuroCAAS provides a user-independent approach to analysis configuration that alleviates the burden of maintaining an open source project across diverse computing environments, and simultaneously provides developers freedom to design infrastructure to be as powerful or complicated as is optimal without being constrained by accessibility concerns. These benefits to analysis users and developers will collectively tighten the feedback loop between experimental and computational neuroscientists.

This commitment to a service architecture also has one notable consequence: if an analysis method is difficult to use – for example if data needs to be preprocessed, or reconfigured in some non-obvious way for proper use – NeuroCAAS does not intervene, and simply passes this analysis onto the user. Explicitly, NeuroCAAS is an infrastructure service, *not* a workflow management system. Many excellent works have and continue to develop such tooling (Perkel, 2019), breaking down complicated data analysis workflows into reusable, organized, well documented parts. NeuroCAAS does not seek to improve the use of analysis directly as these systems do; it seeks only to remove the burden of the infrastructure upon which that analysis sits. Workflow management systems have been specifically designed for neuroscience (Gorgolewski et al., 2011, Yatsenko et al., 2015) as well as more general tools and frameworks (Köster and Rahmann, 2012, Stojanovic, 2016). These tools can be integrated with NeuroCAAS to control workflow in our immutable analysis environments, or replace our current data storage implementation, as well as developing pipelines that easily chain together multiple individual analyses hosted on NeuroCAAS. Integrating workflow management tools more thoroughly with NeuroCAAS is an important future direction to extend the benefits of NeuroCAAS across the entire data lifecycle.

Many different related analysis platforms have gained prominence in our community and others, including models that focus on a set of streamlined core methods (Carpenter et al., 2006, Sommer et al., 2011), as well as those that consolidate a broad selection of community-contributed packages/plugins (De Chaumont et al., 2012, Schindelin et al., 2012, Amezquita et al., 2019). In contrast to these platforms that are installed on a researcher's hardware, NeuroCAAS is totally independent of local infrastructure resources to emphasize standardized, reliable performance for cutting-edge data analyses across a variety of experimental media and analysis methods. Other tools have brought large-scale distributed computing to neuroscience (Freeman, 2015, Rocklin, 2015). It is entirely feasible to extend analyses on NeuroCAAS with more sophisticated methods of parallelization as offered by these tools on a per-analysis basis, as an immutable analysis environment contains all of the infrastructure available in typical computing environments, spanning infrastructure ranging from workstation to compute cluster. Lastly, other tools offer reproducible analyses to researchers as a service (Monajemi et al., 2016, Šimko et al., 2019, Brinckman et al., 2019, `https://flywheel.io`). NeuroCAAS is again complementary to these methods and services because it is designed explicitly for a heterogeneous community of neuroscientists, giving developers access to all the resources required to create powerful, general-purpose analyses, while simultaneously removing all major barriers of entry to these analyses for a diverse population of users.

By virtue of its open source code and public cloud construction, NeuroCAAS will naturally continue to evolve. First, we and community developers will add more analysis algorithms to NeuroCAAS, with an emphasis on subfields of computational analysis that we do not yet support (e.g. Pachitariu et al. (2016) and Lee et al. (2017) for spike sorting) and also add more support for real-time processing (e.g., Giovannucci et al. (2017) for calcium imaging, or Schweihoff et al. (2019) for closed-loop experiments, or Lopes et al. (2015) for the coordination of

multiple data streams). Second, many existing methods can be upgraded given the NeuroCAAS service: for example, on NeuroCAAS we or a developer can trivially take advantage of many more cores or GPUs than are typically available locally on a university compute cluster, and this will likely lead developers to different algorithmic choices and tradeoffs. Third, to facilitate more interactive workflows, we plan further integration with database systems such as Datajoint (Yatsenko et al., 2015) to enable more routine hierarchical multi-step analyses of large-scale datasets.

Finally, and perhaps most importantly, another opportunity for future work is the integration of NeuroCAAS with modern visualization tools. We have emphasized above that NeuroCAAS 's immutable analysis environments are designed with the ideal of fully automated data analyses in mind, because of the virtues that automation brings to data analyses. However, we recognize that for some of the algorithms on NeuroCAAS, and indeed most of those popular in the field, some user interaction is required to optimize results. Nothing fundamentally excludes user interaction (UI) design from NeuroCAAS, so we will establish a configuration path by which analysis developers can also serve a UI to the user in an interactive mode, without sacrificing the benefits of cost efficiency, scalability, and reproducibility that distinguish NeuroCAAS in its current form.

Longer term, we hope to build a sustainable and open-source user and developer community around NeuroCAAS. We welcome suggestions for improvements from users, and new analyses as well as ideas for extensions from interested developers, with the goal of creating a sustainable community-driven resource that will enable new large-scale neural data science in the decade to come.

4 Acknowledgements

We thank Ioana Carcea and Robert C. Froemke for the use of benchmarking data for DeepLabCut, Simon Musall and Anne Churchland for the use of benchmarking data for Penalized Matrix Decomposition and LocaNMF, Jian Wang and Yakov Stern for helpful discussions on NeuroCAAS's architecture and extensions, Peter Lee for development of additional algorithms, Zahra Adahman, Ioana Carcea, Claire Everett, Andres Bendesky, Andres Villegas, Franck Polleux, Vivek Athalye, and Darcy Peterka for discussion and feedback on the use of NeuroCAAS during development, and Dan Biderman and Danil Tyulmankov for useful references and discussions of benchmarking. T.A. is supported by NIH training grant 2T32NS064929-11. S.S. is supported by the Swiss National Science Foundation P2SKP2_178197 and 5U19NS104649. J.P.C. is supported by Simons 542963. L.P. is funded by IARPA MICRONS D16PC00003, NIH 5U01NS103489, 5U19NS104649, 5U19NS107613, 1UF1NS107696, 1UF1NS108213, 1RF1MH120680, DARPA NESD N66001-17-C-4002 and Simons Foundation 543023. L.P. and J.P.C. are supported by NSF Neuronex Award DBI-1707398.

5 Author Contributions

T.A. and J.P.C. conceived of the project. T.A. designed the infrastructure platform with input from all authors. S.S., I.K., and T.A. developed analyses to work on the platform (with supervision from L.P. and J.P.C.), and collected the data shown in Figure 4. T.A., L.P. and J.P.C. wrote the paper.

6 Competing Interests

The authors declare no competing interests.

7 Data Availability Statement

The time and cost quantification data supporting the findings in this study (specifically Figures 4 and 9) are available in the NeuroCAAS repository in the "experiments" directory, at `https://github.com/cunningham-lab/neurocaas/tree/master/experiments`. Although this data will not change with repo development, the git commit at time of submission is b12627eb1db4941a488231a42ecc9cff94f9caff. All other data are included in the manuscript.

8 Code Availability Statement

We make our data collection, analysis, and platform code available at `https://github.com/cunningham-lab/neurocaas/tree/master`.

References

Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al.
2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, Pp. 265–283.

Amezquita, R. A., A. T. Lun, E. Becht, V. J. Carey, L. N. Carpp, L. Geistlinger, F. Martini, K. Rue-Albrecht, D. Risso, C. Soneson, et al.
2019. Orchestrating single-cell analysis with bioconductor. *Nature methods*, Pp. 1–9.

Batty, E., J. Merel, N. Brackbill, A. Heitman, A. Sher, A. Litke, E. Chichilnisky, and L. Paninski
2016. Multilayer recurrent network models of primate retinal ganglion cell responses. *International Conference on Learning Representations*.

Bloch, J.
2008. *Effective java (the java series)*. Prentice Hall PTR.

Brinckman, A., K. Chard, N. Gaffney, M. Hategan, M. B. Jones, K. Kowalik, S. Kulasekaran, B. Ludäscher, B. D. Mecum, J. Nabrzyski, et al.
2019. Computing environments for reproducibility: Capturing the "whole tale". *Future Generation Computer Systems*, 94:854–867.

Buchanan, E. K., I. Kinsella, D. Zhou, R. Zhu, P. Zhou, F. Gerhard, J. Ferrante, Y. Ma, S. Kim, M. Shaik, et al.
2018. Penalized matrix decomposition for denoising, compression, and improved demixing of functional imaging data. *arXiv preprint arXiv:1807.06203*.

Buckheit, J. B. and D. L. Donoho
1995. Wavelab and reproducible research. In *Wavelets and statistics*, Pp. 55–81. Springer.

Business Intelligence
2004. Pilot study: Optimum refresh cycle and method for desktop outsourcing. Technical report, Intel Business Center.

Carcea, I., N. L. Caraballo, B. J. Marlin, R. Ooyama, J. M. M. Navarro, M. Opendak, V. E. Diaz, L. Schuster, M. I. A. Torres, H. Lethin, et al.
2019. Oxytocin neurons enable social transmission of maternal behavior. *bioRxiv*, P. 845495.

Carpenter, A. E., T. R. Jones, M. R. Lamprecht, C. Clarke, I. H. Kang, O. Friman, D. A. Guertin, J. H. Chang, R. A. Lindquist, J. Moffat, et al.
2006. Cellprofiler: image analysis software for identifying and quantifying cell phenotypes. *Genome biology*, 7(10):R100.

Carver, J. C., S. Gesing, D. S. Katz, K. Ram, and N. Weber
2018. Conceptualization of a us research software sustainability institute (urssi). *Computing in Science & Engineering*, 20(3):4–9.

Chen, X., S. Dallmeier-Tiessen, R. Dasler, S. Feger, P. Fokianos, J. B. Gonzalez, H. Hirvonsalo, D. Kousidis, A. Lavasa, S. Mele, et al.
2019. Open is not enough. *Nature Physics*, 15(2):113–119.

Cheng, J. and J. L. Guo
2018. How do the open source communities address usability and ux issues?: An exploratory study. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*, P. LBW523. ACM.

Crook, S. M., A. P. Davison, and H. E. Plesser
2013. Learning from the past: approaches for reproducibility in computational neuroscience. In *20 Years of Computational Neuroscience*, Pp. 73–102. Springer.

De Chaumont, F., S. Dallongeville, N. Chenouard, N. Hervé, S. Pop, T. Provoost, V. Meas-Yedid, P. Pankajakshan, T. Lecomte, Y. Le Montagner, et al.
2012. Icy: an open bioimage informatics platform for extended reproducible research. *Nature methods*, 9(7):690.

Demchenko, Y., P. Grosso, C. De Laat, and P. Membrey
2013. Addressing big data issues in scientific data infrastructure. In *2013 International Conference on Collaboration Technologies and Systems (CTS)*, Pp. 48–55. IEEE.

Donoho, D. L.
2010. An invitation to reproducible computational research. *Biostatistics*, 11(3):385–388.

Freeman, J.
2015. Open source tools for large-scale neuroscience. *Current opinion in neurobiology*, 32:156–163.

Gao, Y., E. W. Archer, L. Paninski, and J. P. Cunningham
2016. Linear dynamical neural population models through nonlinear embeddings. In *Advances in neural information processing systems*, Pp. 163–171.

Giovannucci, A., J. Friedrich, P. Gunn, J. Kalfon, B. L. Brown, S. A. Koay, J. Taxidis, F. Najafi, J. L. Gauthier, P. Zhou, et al.
2019. Caiman an open source tool for scalable calcium imaging data analysis. *Elife*, 8:e38173.

Giovannucci, A., J. Friedrich, M. Kaufman, A. Churchland, D. Chklovskii, L. Paninski, and E. A. Pnevmatikakis
2017. Onacid: Online analysis of calcium imaging data in real time. In *Advances in neural information processing systems*, Pp. 2381–2391.

Gorgolewski, K., C. D. Burns, C. Madison, D. Clark, Y. O. Halchenko, M. L. Waskom, and S. S. Ghosh
2011. Nipype: a flexible, lightweight and extensible neuroimaging data processing framework in python. *Frontiers in neuroinformatics*, 5:13.

Hanson, B., A. Sugden, and B. Alberts
2011. Making data maximally available. *Science*, 331(6018):649.

Hoffa, C., G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good
2008. On the use of cloud computing for scientific workflows. In *2008 IEEE fourth international conference on eScience*, Pp. 640–645. IEEE.

Jararweh, Y., M. Al-Ayyoub, E. Benkhelifa, M. Vouk, A. Rindos, et al.
2016. Software defined cloud: Survey, system and evaluation. *Future Generation Computer Systems*, 58:56–74.

J.Gold Associates LLC
2014. Replacing enterprise pcs: The fallacy of the 3-4 year upgrade cycle. Technical report, J.Gold Associates LLC.

Jupyter, P., M. Bussonnier, J. Forde, J. Freeman, B. Granger, T. Head, C. Holdgraf, K. Kelley, G. Nalvarte, A. Osheroff, and et al.
2018. Binder 2.0 - reproducible, interactive, sharable environments for science at scale. *Proceedings of the 17th Python in Science Conference*.

Köster, J. and S. Rahmann
2012. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522.

Krafczyk, M., A. Shi, A. Bhaskar, D. Marinov, and V. Stodden
2019. Scientific tests and continuous integration strategies to enhance reproducibility in the scientific software context. In *Proceedings of the 2nd International Workshop on Practical Reproducible Evaluation of Computer Systems*, Pp. 23–28. ACM.

Lee, J. H., D. E. Carlson, H. S. Razaghi, W. Yao, G. A. Goetz, E. Hagen, E. Batty, E. Chichilnisky, G. T. Einevoll, and L. Paninski
2017. Yass: Yet another spike sorter. In *Advances in neural information processing systems*, Pp. 4002–4012.

Lopes, G., N. Bonacchi, J. Frazão, J. P. Neto, B. V. Atallah, S. Soares, L. Moreira, S. Matias, P. M. Itskov, P. A. Correia, et al.
2015. Bonsai: an event-based framework for processing and controlling data streams. *Frontiers in neuroinformatics*, 9:7.

Mahvi, J. and A. Zarfaty
2009. Using tco to determine pc upgrade cycles. *Corporation, Intel*.

Mathis, A., P. Mamidanna, K. M. Cury, T. Abe, V. N. Murthy, M. W. Mathis, and M. Bethge
2018. Deeplabcut: markerless pose estimation of user-defined body parts with deep learning. *Nature Neuroscience*, 21(9).

Merkel, D.
2014. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2.

Monajemi, H., D. L. Donoho, and V. Stodden
2016. Making massive computational experiments painless. In *2016 IEEE International Conference on Big Data (Big Data)*, Pp. 2368–2373. IEEE.

Morey, T. and R. Nambiar
2009. Using total cost of owner-ship to determine optimal pc refresh lifecycles.

Morris, K.
2016. *Infrastructure as code: managing servers in the cloud*. " O'Reilly Media, Inc.".

Musall, S., M. T. Kaufman, A. L. Juavinett, S. Gluf, and A. K. Churchland
2019. Single-trial neural dynamics are dominated by richly varied movements. *Nature neuroscience*, 22(10):1677–1686.

Nichols, D. M., K. Thomson, and S. A. Yeates
2001. Usability and open-source software development. In *CHINZ'01*, Pp. 49–54. ACM SIGCHI NZ.

Pachitariu, M., N. A. Steinmetz, S. N. Kadir, M. Carandini, and K. D. Harris
2016. Fast and accurate spike sorting of high-channel count probes with kilosort. In *Advances in Neural Information Processing Systems*, Pp. 4448–4456.

Pachitariu, M., C. Stringer, M. Dipoppa, S. Schröder, L. F. Rossi, H. Dalgleish, M. Carandini, and K. D. Harris
2017. Suite2p: beyond 10,000 neurons with standard two-photon microscopy. *Bioarxiv*, P. 061507.

Pandarinath, C., D. J. O'Shea, J. Collins, R. Jozefowicz, S. D. Stavisky, J. C. Kao, E. M. Trautmann, M. T. Kaufman, S. I. Ryu, L. R. Hochberg, et al.
2018. Inferring single-trial neural population dynamics using sequential auto-encoders. *Nature methods*, 15(10):805–815.

Paninski, L. and J. Cunningham
2018. Neural data science: accelerating the experiment-analysis-theory cycle in large-scale neuroscience. *Current opinion in neurobiology*, 50:232–241.

Parthasarathy, N., E. Batty, W. Falcon, T. Rutten, M. Rajpal, E. Chichilnisky, and L. Paninski
2017. Neural networks for efficient bayesian decoding of natural images from retinal neurons. In *Advances in Neural Information Processing Systems*, Pp. 6434–6445.

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al.
2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, Pp. 8024–8035.

Perkel, J. M.
2019. Workflow systems turn raw data into scientific knowledge. *Nature*, 573:149–150.

Pnevmatikakis, E. A., D. Soudry, Y. Gao, T. A. Machado, J. Merel, D. Pfau, T. Reardon, Y. Mu, C. Lacefield, W. Yang, et al.
2016. Simultaneous denoising, deconvolution, and demixing of calcium imaging data. *Neuron*, 89(2):285–299.

Radiuk, P. M.
2017. Impact of training set batch size on the performance of convolutional neural networks for diverse datasets. *Information Technology and Management Science*, 20(1):20–24.

Raff, E.
2019. A step toward quantifying independently reproducible machine learning research. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc.

Rocklin, M.
2015. Dask: Parallel computation with blocked algorithms and task scheduling. In *Proceedings of the 14th Python in Science Conference*, K. Huff and J. Bergstra, eds., Pp. 130 – 136.

Saxena, S., I. Kinsella, S. Musall, S. H. Kim, J. Meszaros, D. N. Thibodeaux, C. Kim, J. Cunningham, E. M. Hillman, A. Churchland, et al.
2020. Localized semi-nonnegative matrix factorization (locanmf) of widefield calcium imaging data. *PLOS Computational Biology*, 16(4):e1007791.

Schindelin, J., I. Arganda-Carreras, E. Frise, V. Kaynig, M. Longair, T. Pietzsch, S. Preibisch, C. Rueden, S. Saalfeld, B. Schmid, et al.
2012. Fiji: an open-source platform for biological-image analysis. *Nature methods*, 9(7):676–682.

Schweihoff, J. F., M. Loshakov, I. Pavlova, L. Kück, L. A. Ewell, and M. K. Schwarz
2019. Deeplabstream: Closing the loop using deep learning-based markerless, real-time posture detec-tion. *bioRxiv*.

Šimko, T., L. Heinrich, H. Hirvonsalo, D. Kousidis, and D. Rodríguez
2019. Reana: A system for reusable research data analyses. In *EPJ Web of Conferences*, volume 214, P. 06034. EDP Sciences.

Sommer, C., C. Straehle, U. Koethe, and F. A. Hamprecht
2011. Ilastik: Interactive learning and segmentation toolkit. In *2011 IEEE international symposium on biomedical imaging: From nano to macro*, Pp. 230–233. IEEE.

Stodden, V., J. Seiler, and Z. Ma
2018. An empirical analysis of journal policy effectiveness for computational reproducibility. *Proceedings of the National Academy of Sciences*, 115(11):2584–2589.

Stojanovic, P. A. M. R. C. N. T. B. C. J. C. M. H. A. K. J. K. D. L. H. M. M. N. M. S. S. S.-R. L.
2016. Common workflow language, v1.0.

Terry, M., M. Kay, and B. Lafreniere
2010. Perceptions and practices of usability in the free/open source software (foss) community. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Pp. 999–1008. ACM.

Tukey, J. W.
1962. The future of data analysis. *The annals of mathematical statistics*, 33(1):1–67.

Vogelstein, J. T., B. Mensh, M. Häusser, N. Spruston, A. C. Evans, K. Kording, K. Amunts, C. Ebell, J. Muller, M. Telefont, et al.
2016. To the cloud! a grassroots proposal to accelerate brain science discovery. *Neuron*, 92(3):622–627.

Waltz, D. and B. G. Buchanan
2009. Automating science. *Science*, 324(5923):43–44.

Wiltschko, A. B., M. J. Johnson, G. Iurilli, R. E. Peterson, J. M. Katon, S. L. Pashkovski, V. E. Abraira, R. P. Adams, and S. R. Datta
2015. Mapping sub-second structure in mouse behavior. *Neuron*, 88(6):1121–1135.

Yatsenko, D., J. Reimer, A. S. Ecker, E. Y. Walker, F. Sinz, P. Berens, A. Hoenselaar, R. James Cotton, A. S. Siapas, and A. S. Tolias
2015. Datajoint: managing big scientific data using matlab or python. *bioRxiv*.

Yu, B. M., J. P. Cunningham, G. Santhanam, S. I. Ryu, K. V. Shenoy, and M. Sahani
2009. Gaussian-process factor analysis for low-dimensional single-trial analysis of neural population activity. *J Neurophysiol*, 102(1):614–35.

Zhao, L. and F. P. Deek
2005. Improving open source software usability. *AMCIS 2005 Proceedings*, P. 430.

Zhou, A., B. He, S. Ibrahim, R. Buyya, R. Calheiros, and A. Dastjerdi
2016. escience and big data workflow in clouds: A taxonomy and survey. *Big data: Principles and paradigms*, Pp. 431–456.

Zhou, P., S. L. Resendez, J. Rodriguez-Romaguera, J. C. Jimenez, S. Q. Neufeld, A. Giovannucci, J. Friedrich, E. A. Pnevmatikakis, G. D. Stuber, R. Hen, et al.
2018. Efficient and accurate extraction of in vivo calcium signals from microendoscopic video data. *Elife*, 7:e28728.

9 Methods

9.1 NeuroCAAS architecture specifics

The NeuroCAAS platform is comprised of two complementary components: the infrastructure resources that run data analyses (i.e., handle data, launch the immutable analysis environment, and manage the analysis jobs) and the code schematic that defines these infrastructure resources. Together these two components (Figure 5) constitute our Infrastructure-as-Code (IaC) architecture.

The code schematic (Figure 5, top) contains a comprehensive and programmatic design specification for all resources used in NeuroCAAS. It includes template code for NeuroCAAS blueprints, analogous templates for user profiles, and protocols determining the behavior of NeuroCAAS's per-analysis job managers. First, for each NeuroCAAS analysis, its corresponding blueprint contains the details necessary to fully describe analysis configuration and usage via an immutable analysis environment (§9.2, 9.3). An example of the NeuroCAAS blueprint template is included in the supplementary text; all other components of the NeuroCAAS code schematic are available at `https://github.com/cunningham-lab/neurocaas`). Second, for each NeuroCAAS user, their user profile contains user-specific data storage and security information, as described in section (§10.1). The front end `neurocaas.org` automates the process of filling in user profile templates from a web interface.

Finally, to transmit data between analysis based and user based resources, we rely on autonomous, event-triggered *job managers*. Within the code schematic, protocols dictate the duties and behavior of job managers, such as parsing user input, monitoring active analyses for errors and sending key outputs back to the user as well as various resources. Protocols were configured and packaged with each NeuroCAAS blueprint according to the particular needs that each analysis imposed on the job manager.

To materialize this code schematic into NeuroCAAS's infrastructure resources, NeuroCAAS contains tools that read the code schematic and provision the virtualized resources declared within. The code schematic also provides directions to automatically configure and link these resources into a fully functioning platform without further manual setup. We call the conversion of the code schematic into infrastructure resources *deployment* (Figure 5, middle). There is a one-to-one correspondence between NeuroCAAS's code schematic and infrastructure components: deploying the code schematic provides total coverage of all of the functioning resources that together constitute NeuroCAAS (Figure 5, bottom). For our public cloud implementation, we used a variety of AWS cloud resources as building blocks for deployed infrastructure, as described in the following sections. We designed deployment to be modular, so that individual blueprints and user profiles could be deployed to extend the existing NeuroCAAS platform. Notably, our deployment tools interfaced with the AWS Serverless Application Model (an AWS-native IaC management resource), and the python library troposphere `https://troposphere.readthedocs.io/en/latest/` (a set of utilities that make our code schematic more legible and easier to organize).

9.2 Building immutable analysis environments

For each immutable analysis environment, the blueprint specifies hardware as a single virtualized *instance*; i.e., a bundled collection of virtual cpus, memory, and gpus. Some virtualized instances have capacity analogous to personal hardware, while others provide resources comparable to high performance cluster compute nodes (see Table 1 for details). We chose instances for each of our analyses from the list of available Amazon EC2 instance types; these are updated periodically as newer, faster hardware becomes available. Data storage was provisioned on these instances (again, via the blueprint) with virtualized hard disks large enough to accommodate the datasets analyzed as performance benchmarks (see Table §4). Additional storage can be mounted on-demand to immutable analysis environments, increasing storage capacity as required by particularly large datasets. We specified storage as Amazon EBS volumes. All of our analysis environments ran on virtualized Linux operating systems (EC2 base AMIs).

To bring the actual analyses into NeuroCAAS, we followed developer instructions and/or collaborated with the analysis developers. Custom per-analysis scripts handle user inputs (data and parameters) and pass these inputs to the native analysis interface (the code from the developers) over the course of an active analysis's progress; see supplementary material for an example script. Scripts were saved as executables in the analysis environment.

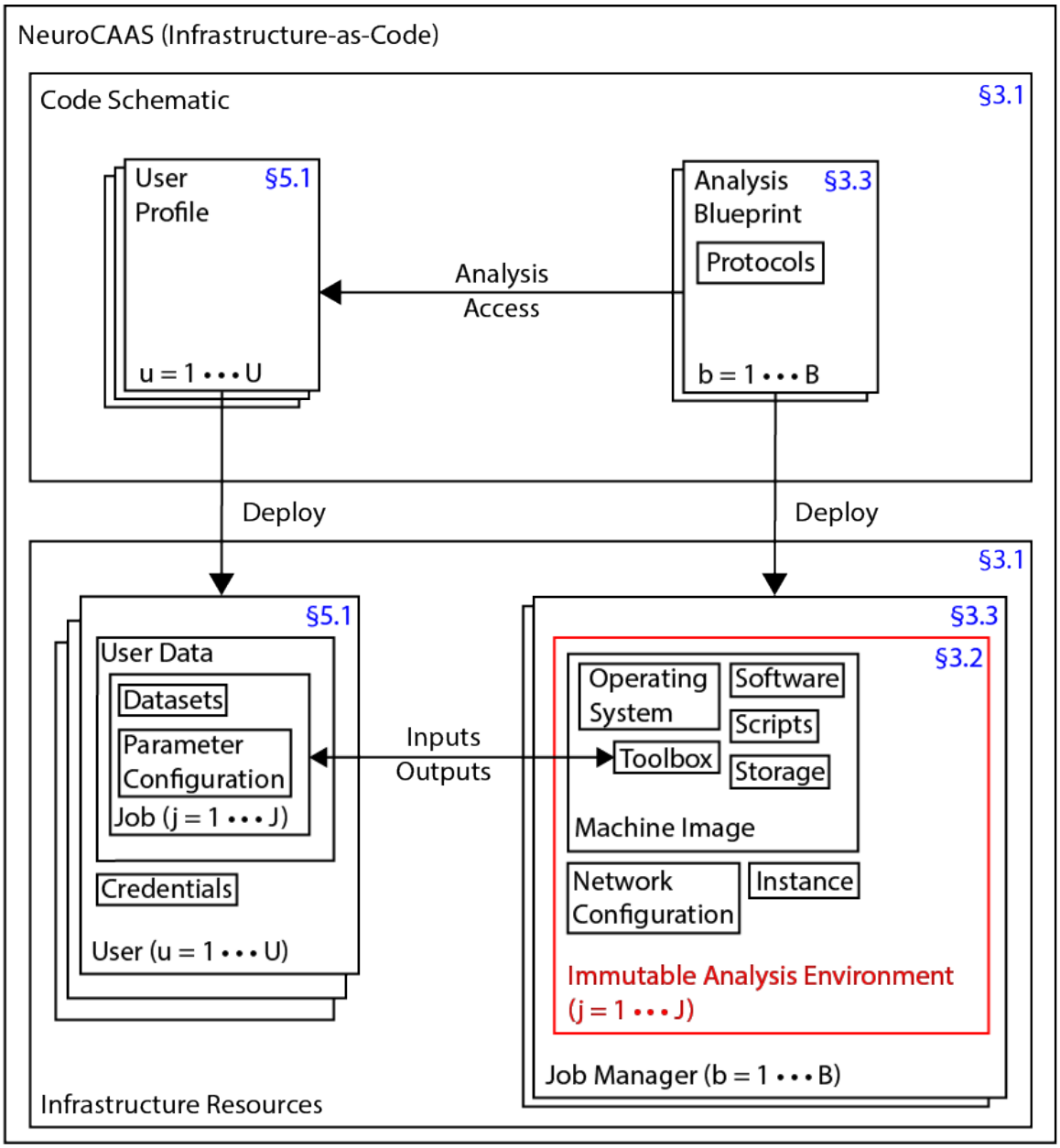


Figure 5: **NeuroCAAS design diagram.** NeuroCAAS is built with an Infrastructure-as-Code design, meaning that we first write a *code schematic* (top) specifying all of the actual resources we will use to carry out data processing (bottom). Section numbers refer to relevant parts of the main text. The code schematic (top) contains three main types of code: User Profiles, specifying relevant user data; Analysis Blueprints, describing individual analyses on NeuroCAAS, and Protocols, giving rules that describe NeuroCAAS job manager function. Each user and each analysis in NeuroCAAS has a dedicated code document, as specified by indices (u, b). All parts of the code schematic can independently be *deployed*, automatically provisioning and configuring the infrastructure resources specified therein. Deployment comprehensively generates the resources necessary to run analyses on NeuroCAAS. Notably, Immutable Analysis Environments (bottom right) are not persistent, but rather are instantiated every time users request an analysis job, specified as a combination of datasets and parameter configurations (bottom left). Job managers deploy one Immutable Analysis Environment for each requested job, as specified by the index j.

To ensure immutability, we defined the job manager to be the only entity capable of starting scripts in an immutable analysis environment in the code schematic and closed analysis environments off from standard network access protocols such as ssh. To allow data upload to the analysis environment without the interactive access provided by standard network protocols, we wrote an additional toolbox that facilitates data transfer between the immutable analysis environment and the user. This toolbox communicates with the job manager and is critical to

deliver status updates mid-analysis to the user. The toolbox consists of a set of shell functions that are analysis-agnostic and shared between all of the analyses currently implemented on NeuroCAAS. It can be found here (`https://github.com/cunningham-lab/neurocaas_remote`), along with a set of example use cases that should be sufficient to implement data transfer (input and output) for most analyses relevant to NeuroCAAS (template in supplement).

Once configured, we saved the state of the analysis environment's operating system, storage, installed applications, and toolbox in a single *machine image* (Figure 5, bottom right), capturing infrastructure details that would otherwise be difficult to document in a code schematic. Once saved, machine images can be easily redeployed with different hardware on other instances, providing immutable analysis environments with portability similar to that provided by Docker images. Together with the instance type and network configuration, the machine image provides a complete account of the state of a configured immutable analysis environment.

For developers, detailed instructions for hosting analyses on NeuroCAAS can be found at `neurocaas.org`.

9.3 Deploying analyses from blueprints

NeuroCAAS blueprints are custom built JSON objects that can describe any analysis deployed on NeuroCAAS. In addition to the machine image,each blueprint contains the instance type and network configuration of the immutable analysis environment. Each blueprint also includes a list of the users approved to use to the corresponding analysis. Storing user access details in the blueprint allows NeuroCAAS to monitor the resource usage of each user, distribute users to minimize wait time, and easily make newly developed analyses available to users. For an example blueprint and more on user management see **supplementary methods**.

Additionally, each blueprint is paired with a *protocol* that configures a dedicated job manager for a particular analysis (Figure 5, top right). Protocols describe the input format that the job manager should expect for a given analysis, the analysis diagnostics that should be routed to the user during analysis, and the specific strategy used to scale processing over many datasets. Protocols are written in python, and job managers were implemented on AWS Lambda.

Deployment of each blueprint illustrates a critical feature of NeuroCAAS's design. Deploying a single blueprint creates a dedicated job manager for the corresponding analysis, but *does not* directly create a corresponding immutable analysis environment. Instead, through protocols the job manager is given the ability to *autonomously* deploy and dissolve any number of immutable analysis environments in response to requests from approved users (Figure 5, red). For each analysis request submitted by a user, an entirely new immutable analysis environment is instantiated on-demand, and subsequently destroyed at the end of processing.

NeuroCAAS job managers also have the ability to deploy multiple immutable analysis environments in response to batched inputs from a single user. When provided with multiple different datasets all to be analyzed with the same parameters, job managers for the analyses that we host deploy one immutable analysis environment for each, providing easy parallelization.

We collected all of the NeuroCAAS blueprints into a version controlled repository, and stipulated that all NeuroCAAS analyses can be updated only by directly editing blueprints (see **supplementary methods**). By enforcing a tight correspondence between blueprints and analyses, we ensured the reproducibility of all analyses conducted via NeuroCAAS, regardless of ongoing updates to the underlying infrastructure or algorithm (Figure 5). For details on reproducing previously run analyses, see **supplements** and `neurocaas.org`.

9.4 Benchmarking algorithms on NeuroCAAS

For each analysis currently on NeuroCAAS, the specific infrastructure choices in the corresponding blueprint (Figure 5, right) are given in Table 1. To meaningfully benchmark NeuroCAAS against current standards, we built corresponding *local* infrastructure to simulate resources that might be available to a typical user. Local infrastructure was also built on AWS, and spans resources comparable to personal hardware and cluster compute, depending on the use case (see Table 2). As a general guideline, we chose local infrastructure representatives that would reasonably be available to a typical researcher, unless the datasets we considered required more powerful resources. To account for the diversity of resources available to neuroscience users, we offer alternative quantifications to those presented in Figure 4 in the supplementary methods (see Figure 9), and make calculations available to users who would like to compare to their own infrastructure through a custom tool on our project repository (see README): `https://github.com/cunningham-lab/neurocaas`.

NeuroCAAS Available Analyses									
Analysis Name	Storage	Memory	GPU	CPU count	OS	Job Monitor	Resource Usage	Version Control	Packages
CaImAn	500 GB SSD	275 GB	N/A	64 vCPU[1]	Ubuntu 16.04 (Linux HVM)	stdout	CPU Utilization	Github	pip requirements file
DeepLabCut	200 GB SSD	65 GB	Nvidia Tesla K80	4 vCPU[2]	Ubuntu 16.04 (Linux HVM)	stdout	CPU Utilization	Github	conda environment file
PMD	75 GB SSD	275 GB	N/A	64 vCPU[1]	Ubuntu 18.04 (Linux HVM)	stdout	CPU Utilization	Github	Conda Package
LocaNMF	75 GB SSD	65 GB	Nvidia Tesla V100	8 vCPU[3]	Ubuntu 16.04 (Linux HVM)	stdout	CPU Utilization	Github	Conda Package

[1] Intel Xeon Platinum 8000 series (Skylake-SP): https://aws.amazon.com/ec2/instance-types/m5/

[2] Intel Broadwell (AWS): https://aws.amazon.com/blogs/aws/new-p2-instance-type-for-amazon-ec2-up-to-16-gpus/

[3] Intel Xeon E5-2686v4: https://aws.amazon.com/blogs/aws/new-amazon-ec2-instances-with-up-to-8-nvidia-tesla-v100-gpus-p3/

Table 1: Infrastructure details for implemented algorithms. Job Monitor refers to the mechanisms used to track the status of ongoing jobs. Resource Usage refers to the hardware diagnostics tracked by NeuroCAAS. Version Control refers to the version control mechanisms used to maintain fidelity of core analysis code. Packages refers to the mechanisms used to handle analysis dependencies.

For each analysis that we benchmarked on NeuroCAAS, we chose three datasets of increasing size as representative use cases of the algorithms in question. The size differences of these datasets reflect the diversity of potential use cases among different users of the same algorithm. The CaImAn benchmarking data consists of datasets N.02.00, J_123, J_115 from the data shared with the CaImAn paper (Giovannucci et al., 2019). Benchmark analysis is based on a script provided to regenerate Figure 4 of the CaImAn paper. Note that although this data could be batched, we choose to maintain all three datasets as contiguous wholes. DeepLabCut benchmarking data consists of behavioral video capturing social interactions between two mice in their home cage. Data is provided courtesy of Robert C. Froemke and Ioana Carcea, as analyzed and presented in Carcea et al. (2019). Data processing consisted of analyzing these videos with a model that had previously been trained on other images from the same dataset. The same dataset was used to benchmark PMD and LocaNMF as a single analysis pipeline with two discrete parts. Input data consist of the dataset ("mSM30"), comprising widefield calcium imaging data videos, provided courtesy of Simon Musall and Anne Churchland, as used in Musall et al. (2019) and Saxena et al. (2020). The full dataset is available in a denoised format at `http://repository.cshl.edu/id/eprint/38599/`. Data processing on NeuroCAAS consisted of first processing the raw videos with PMD, then passing the resulting output to LocaNMF. Further details on the datasets used can be found in Table 4.

We split the time taken to run analyses on NeuroCAAS into two separate quantities. First, we quantified the time taken to upload data from local machines to NeuroCAAS, denoted as NeuroCAAS (Upload) in Figure 4. This time depends upon the specifics of the internet connection that is being used. It is also a one time cost: once data is uploaded to NeuroCAAS, it can be reanalyzed many times without incurring this cost again. Upload times were measured from the same NeuroCAAS interface made available to the user. (This upload time was skipped in the quantification of local processing time.) Second, we automatically quantified the total time elapsed between job submission and job termination, when results have been delivered back to the end user in the NeuroCAAS interface (denoted as NeuroCAAS (Compute) in Figure 4) via AWS native tools (see **supplement**

NeuroCAAS Local Simulation									
Analysis Name	Storage	Memory	GPU	CPU count	OS	Job Monitor	Resource Usage	Version Control	Packages
CaImAn	500 GB SSD	17 GB	N/A	4 vCPU[1]	Ubuntu 16.04 (Linux HVM)	stdout	None	github	Pip requirements file
DeepLabCut	200 GB SSD	65 GB	Nvidia Tesla K80	4 vCPU[2]	Ubuntu 16.04 (Linux HVM)	stdout	None	github	Conda Package
PMD	75 GB SSD	131 GB	N/A	16 vCPU[3]	Ubuntu 18.04 (Linux HVM)	stdout	None	Github	Conda Package
LocaNMF	75 GB SSD	65 GB	Nvidia Tesla K80	4 vCPU[2]	Ubuntu 16.04 (Linux HVM)	stdout	None	github	Conda Package

[1] AMD EPYC 7000 Series: https://aws.amazon.com/ec2/instance-types/m5/

[2] Intel Broadwell (AWS): https://aws.amazon.com/blogs/aws/new-p2-instance-type-for-amazon-ec2-up-to-16-gpus/

[3] Intel Xeon E5 Broadwell Processors: https://aws.amazon.com/blogs/aws/new-next-generation-r4-memory-optimized-ec2-instances/

Table 2: Details of infrastructure used to simulate local processing. The column labels mirror those in Table 1 .

Pricing List		
Resource	Metrics	Rate
EC2 (Compute)	Time	Hardware Dependent, Fluctuates
Lambda (Workflow)[1]	Data Size × Time	\$1.66667 × 1e−5 per GB-second
S3 (Data Transfer Out)[2]	Data Size	\$0.09 per GB

[1] AWS Lambda is also priced for number of requests, but this is a negligible cost for a single analysis run.

[2] Data Transfer is only priced out of Amazon Web Services, i.e. in returning results to the end user.

Table 3: Pricing details for implemented algorithms

for details). Local timings were measured on automated portions of workflow in the same manner as NeuroCAAS (Compute).

We quantified the cost of running analysis on NeuroCAAS by enumerating costs of each of the AWS resources used in the course of a single analysis. Costs can be found in Table 3. We provide the raw quantification data and corresponding prices in Table 3. To further reduce costs, we also offer the option to utilize AWS Spot Instances (dedicated duration); these are functionally identical to standard compute instances, but are provisioned for a pre-determined amount of time with the benefit of significantly reduced prices. We provide the estimated cost of running analyses with both of these options in Figure 4, with quantifications labeled "NeuroCAAS Save" corresponding to analyses run with dedicated duration spot instances, and those labeled "NeuroCAAS Std" corresponding to those run with standard instances. For more on Spot Instance price quantification, see **supplementary methods**.

Dataset Details							
Analysis	Format	Size (Small)	Dim (Small)	Size (Medium)	Dim (Medium)	Size (Large)	Dim (Large)
CaImAn	zipped tiff	8.39GB	$8000 \times 512 \times 512$[1]	35.84GB	$41000 \times 458 \times 477$[1]	78.70GB	$90000 \times 463 \times 472$[2]
DeepLabCut	mpeg	5×214.8MB	$5 \times 36000 \times 340 \times 420$[3]	10×214.8MB	$10 \times 36000 \times 340 \times 420$[3]	15×214.8MB	$15 \times 36000 \times 340 \times 420$[3]
PMD + LocaNMF	numpy array	1×20.1GB	$[500 \times 600 \times 1697, 1697 \times 8979]$[4]	3×20.1GB	$[500 \times 600 \times 1697, 1697 \times 8979]$; $[500 \times 600 \times 1652, 1652 \times 9000]$; $[500 \times 600 \times 2298, 2298 \times 8988]$[4]	5×20.1GB	$[500 \times 600 \times 1697, 1697 \times 8979]$; $[500 \times 600 \times 1652, 1652 \times 9000]$; $[500 \times 600 \times 2298, 2298 \times 8988]$; $[500 \times 600 \times 2304, 2304 \times 8992]$; $[500 \times 600 \times 1910, 1910 \times 8952]$[4]

1 [Time × X × Y] at 7 hz Giovannucci et al. (2019)
2 [Time × X × Y] at 30 hz Giovannucci et al. (2019)
3 [Batch × Time × X × Y] at 30 hz
4 [X × Y × Rank, Rank × Time] at 30 Hz

Table 4: Details of the datasets used to benchmark performance. Sizes given for the three datasets tested for each pipeline shown. Dataset dimensionality labels are included in footnotes provided.

We hosted local infrastructures on AWS, but calculated local costs for them by pricing analogous computing resources as available through a personal workstation, or a local cluster (Table 7). In Figure 4, we assume that the local infrastructures considered are hosted on typical local laptop or desktop computing resources, supplemented with the resources necessary to run analyses as they were done on NeuroCAAS (additional storage, memory, GPU, etc), while maintaining approximate parity in processor power. We referred to (Morey and Nambiar, 2009) to convert pricetag costs of local machines to Equivalent Annual Costs, i.e. the effective cost per year if we assume our local machines will remain in service for a given number of years, as our implementation of a TCO calculation (as is often done in industry). Given a price tag cost x_{local}, an assumed lifetime n, an annuity rate r, and $c_s(n)$ defined as the estimated annual cost of local machine support given a lifetime n, we follow Mahvi and Zarfaty (2009), Morey and Nambiar (2009) in calculating the Equivalent Annual Cost as:

$$EAC(x_{local}, n, r) = \frac{x_{local}}{\frac{1-(1+r)^{-n}}{r}} + c_s(n).$$

Here $c_s(n)$ is provided in the cited paper (Morey and Nambiar, 2009), estimated from representative data across many different industries. The denominator of the first term is an annuity factor. We consider two different values for n, which we label as "realistic" (2 years) and "optimistic" (4 years) in the text. In industry, 3-4 years is the generally accepted optimal lifespan for computers, after which support costs outweigh the value of maintaining an old machine ("Pilot Study", 2004, Mahvi and Zarfaty, 2009, Morey and Nambiar, 2009). Some have argued that with more modern hardware, the optimal refresh cycle has shortened to 2 years (J.Gold Associates LLC, 2014). By providing quantifications assuming two and four year refresh cycle, we consider the short and long end of this generally discussed optimal range.

Given a per-dataset NeuroCAAS cost $x_{\text{NeuroCAAS}}$, we further derive the Local Cost Crossover (LCC), the threshold weekly data analysis rate at which it becomes cost-effective to buy a local machine. The LCC is given

Calcium Imaging										
Algorithm Name	Publication	Version Control	Packages	Operating System	Job Monitor	Resource Usage	Storage	Memory	GPU	CPU
CaImAn	Giovannucci et al. 2017	✓	✓	X	✓	X	X	X	X	X
CNMF-E	Zhou et al. 2018	✓	✓	X	✓	X	X	X	X	X
Suite2p	Pachitariu et al. 2017	✓	✓	X	✓	X	X	X	X	X
ABLE	Reynolds et al. 2017	✓	✓	X	✓	X	X	X	X	X
SCALPEL	Petersen et al. 2018	X	X	X	✓	X	X	X	X	X
Min1PIPE	Lu et al. 2018	✓	✓	X	✓	X	X	X	X	X
SamuROI	Rueckl et al. 2017	✓	X	X	✓	X	X	X	X	X
Romano	Romano et al. 2017	✓	✓	X	✓	X	X	X	X	X
FISSA	Keemink et al. 2018	✓	✓	X	✓	X	X	X	X	X
OASIS	Friedrich et al. 2017	✓	✓	X	✓	X	X	X	X	X
Percentage Supporting		90%	80%	0%	100%	0%	0%	0%	0%	0%

Table 5: Infrastructure support for Calcium Imaging Algorithms. Labels mirror those in Table 1.

by:

$$LCC(x_{local}, n, r, x_{\text{NeuroCAAS}}) = \frac{EAC(x_{local}, n, r)}{52 \times x_{\text{NeuroCAAS}}}.$$

Furthermore, given the per-dataset local analysis time, we can estimate the corresponding Local Utilization Crossover (LUC). The LUC considers the LCC in the context of the maximal achievable data analysis rate on local infrastructure as calculated in the previous section. If the time taken to analyze a dataset on a local machine is given by t_{local} (in seconds), The LUC is given by:

$$LUC(t_{local}, x_{local}, n, r, x_{\text{NeuroCAAS}}) = \frac{LCC(x_{local}, n, r, x_{\text{NeuroCAAS}}) \times t_{local} \times 100}{604800}.$$

9.5 Survey

We characterized data analysis infrastructure as consisting of three hierarchical parts (Dependencies, System, Hardware), segmented consistently with infrastructure descriptions referenced elsewhere (Demchenko et al., 2013, Zhou et al., 2016). In several different subfields of neuroscience, we then selected 10 recent or prominent analysis techniques, and asked how they fulfilled each component of data analysis infrastructure. We denoted a particular infrastructure component as supported if it is referenced in the relevant installation and usage guides as being provided in a reliable, automated manner (e.g., automatic package installation via pip). Survey details are provided in tables 5, 6. We addressed the question of how data analyses are installed and used with these surveys in the tradition of the open source usability literature. Surveys such as these are standard methodology in this field, which relies upon empirical data from studies of user's usage habits (Nichols et al., 2001, Zhao and Deek, 2005), developer sentiment (Terry et al., 2010), and observation of user-developer interactions via platforms like github (Cheng and Guo, 2018).

Behavioral Quantification.										
Algorithm Name	Publication	Version Control	Packages	Operating System	Job Monitor	Resource Usage	Storage	Memory	GPU	CPU
DeepLabCut	Mathis et al. 2018	✓	✓	✓	✓	X	X	X	X	X
DeepFly3D	Günel et al. 2019	✓	✓	X	✓	X	X	X	X	X
JAABA	Kabra et al. 2012	✓	✓	X	✓	X	X	X	X	X
Ctrax	Branson et al. 2009	✓	✓	X	✓	X	X	X	X	X
DeepPoseKit	Graving et al. 2019	✓	X	X	✓	X	X	X	X	X
Ethovision	—-	✓	✓	X	✓	X	X	X	X	X
APT	—-	✓	✓	✓	✓	X	X	X	X	X
bonsai	Lopes et al. 2015	X	✓	X	✓	X	X	X	X	X
Miceprofiler	de Chaumont et al. 2012	✓	✓	X	✓	X	X	X	X	X
LEAP	Pereira et al. 2018	✓	✓	X	✓	X	X	X	X	X
Percentage Supporting		90%	90%	20%	100%	0%	0%	0%	0%	0%

Table 6: Infrastructure support for Behavioral Quantification Algorithms. Labels mirror those in Table 1.

Cost (Local)						
Algorithm Name	vCPU count	GPU	Memory	Storage Capacity	Workstation Price, US Dollars (Estimated Price Tag Cost)	Cluster Price, US Dollars (Estimated Price Tag Cost from Amazon TCO Calculator)
CaImAn	4	N/A	16 GiB	500 GB	1618[2]	1499+1000
DeepLabCut	4	Tesla K80	61GiB	200 GB	3120[3]	1701+400+1555[5]
PMD + LocaNMF [1]	16	Tesla K80	122 GiB	150 GB	5436[4]	10836+300+1555[5]

[1] Cost for PMD and LocaNMF refers to hardware cost for a local instance that can account for processing done on both.

[2] `https://www.newegg.com/p/1TS-000D-052P6`

[3] `https://www.newegg.com/p/1VK-001E-1SVY3?Item=9SIADB38AG7178&Description=1080%20ti%20workstation%2064%20gB%204%20core&cm_re=1080_ti_workstation_64_gB_4_core-_-1VK-001E-1SVY3-_-Product`

[4] `https://www.newegg.com/p/1VK-001E-1A6V1`

[5] `https://www.vgastore.com/2023019/hp-j0g95a-tesla-k80-24gb-384-bit-gddr5-pci-e-3-0-x16-graphics-card`

Table 7: Instance and hardware cost details for local cost comparisons. Estimated Price tag prices as of May 3rd, 2020. Price tag estimation of workstation style hardware was based on market prices chosen to reflect the infrastructure implementation as given in Table 2, in particular, CPU make. Estimation of cluster style hardware cost was based on the AWS TCO calculator (`https://awstcocalculator.com`), as of January 25th, 2020, incorporating the total server hardware cost (undiscounted) and acquisition cost of SAN storage.

10 Supplementary Materials

10.1 Managing users from user profiles

On NeuroCAAS, users resources were defined in code via JSON documents we call user profiles. New users were registered by filling in a corresponding user profile, which was then deployed to automatically generate storage space, dedicated login credentials, and permissions to use analyses for the user. The user profile is similar in format to the UXData segment of the blueprint as given in Figure 8, and can be found in the NeuroCAAS codebase online.

We created a secure, virtualized storage location where users could store their data on NeuroCAAS before and in between analyses. Data storage on NeuroCAAS is shared within a user group (i.e. a lab), but private from all other parties. In figure 4, NeuroCAAS Upload time refers to the time required to upload data from local machines to this storage- once uploaded, data can be deleted post analysis or maintained over the course of several analyses. Maintaining data post analysis cuts out NeuroCAAS Upload time on subsequent upload events.

User credentials are automatically generated upon new user sign up. Permissions to use analyses are generated by updating NeuroCAAS blueprints with the information of newly added users, and redeploying the analysis in question. Upon redeployment, the corresponding job manager begins monitoring this new user for analysis requests. Importantly, this addition is minimally disruptive to ongoing analyses.

10.2 Managing analyses from blueprints

On NeuroCAAS, all changes to analyses in the course of development were recorded in the corresponding analysis blueprint. We updated analyses during development by instantiating the analysis's immutable analysis environment in "developer mode", which allowed ssh. After making the appropriate edits, details of the resulting environment were saved to the corresponding blueprint. The analysis blueprint could then be redeployed, reconfiguring the job manager to create and destroy the new, updated analysis environment. Likewise, the addition of new users and adjustments to the job manager's programming was managed via blueprints. Any divergence of deployed analyses from blueprints due to manual adjustment was detected via AWS Cloudformation, and corrected by redeploying blueprints. Importantly, blueprint redeployment first determines the minimal set of changes necessary to update existing analyses, and implements them in a way that minimizes interruptions to ongoing analyses.

10.3 Automatic compute benchmarking

The duration of NeuroCAAS Compute and Local analysis time was recorded automatically with cloud native resource monitoring tools. These tools were automatically notified of the creation and destruction of immutable analysis environments, and recorded the relevant timestamps at millisecond resolution. These monitoring tools were also managed via NeuroCAAS blueprints, and their design can be found in the same blueprint codebase. Automatic monitoring was implemented via AWS Lambda, AWS Cloudwatch Events, and AWS S3.

10.4 Spot instance pricing

The virtualized hardware underlying immutable analysis environments can be provisioned at several different prices. We used Spot Instance pricing to reduce costs, having known beforehands how long the analyses would take. At the moment, we depict prices based on spot instance availability in September 2019. Empirically, we observe that spot instance price fluctuations give standard deviations on the order of cents over a period of months (see code for experiments).

10.5 Analysis reproducibility

Because we designed analysis blueprint to be git versioned, we can reproduce the infrastructure and software configuration used to generate any analysis, up to the reliability of Amazon AWS. Since we returned identifying information about the blueprint to the user in a certificate along with configuration parameters, data is the only portion of an analysis that must be maintained to ensure perfect analysis reproducibility. Although not implemented here, AWS offers cheap, glacial storage that can be used to preserve data for long amounts of time

NeuroCAAS AWS Specifics			
Analysis Name	Instance (NeuroCAAS)	Instance (local)	AMI ID
CaImAn	m5.16xlarge	m5a.xlarge	ami-01dc867df8c05aa5a)
DeepLabCut	p2.xlarge	p2.xlarge	ami-00b1babeb8637f5c3)
PMD	m5.16xlarge	r4.4xlarge	ami-0007adf33fbcf0c1c)
LocaNMF	p3.2xlarge	p2.xlarge	ami-04ebe747c2e33038c)

Table 8: Instance and Amazon Machine Image (AMI) details for implemented algorithms.

under conditions of infrequent access, offering a feasible solution for guaranteed total analysis reproducibility on NeuroCAAS.

10.6 Alternative local crossovers

Because the instances offered on AWS are not wholly analogous to either personal hardware or cluster resources, we offer additional comparisons that span the range of prices.

Cluster pricing was calculated with the AWS TCO calculator `https://awstcocalculator.com/#`. We calculated the cost of infrastructure as a subset of the TCO provided by AWS. In particular, we calculated x_{local} as the total server hardware cost (undiscounted) and acquisition cost of NAS storage, and the cost of a GPU, with additional yearly recurring costs $c_s(n)$ given by storage administration cost, server hardware maintenance cost, and IT Labor costs. We then calculated the LCC and LUC from these quantities as described in the Methods.

The results of these quantifications are given in Figure 9.

10.7 AWS details

We provide details on the AWS implementation of analyses used to generate time and cost data in Table 8.

```
#!/bin/bash
## Import functions for workflow management.
## Get the path to this function:
execpath="$0"
echo execpath
scriptpath="$(dirname "$execpath")/ncap_utils"

source "$scriptpath/workflow.sh"
## Import functions for data transfer
source "$scriptpath/transfer.sh"

## Set up error logging.
errorlog
## Custom setup for this workflow.
source .dlamirc
export PATH="/home/ubuntu/anaconda3/bin:$PATH"
source activate dlcami

## Declare local storage locations:
userhome="/home/ubuntu"
datastore="ncapdata/localdata/"
outstore="ncapdata/localout/"
## Make local storage locations
accessdir "$userhome/$datastore" "$userhome/$outstore"

## Stereotyped download script for data. The only reason this comes after something custom is because
    we depend upon the AWS CLI and installed credentials.
download "$inputpath" "$bucketname" "$datastore"

## Stereotyped download script for config:
download "$configpath" "$bucketname" "$datastore"
#############################################################################################
## parse the config to place model folder, myconfig_analysis, in the right places.
modelpath=$(cat "$userhome/$datastore/$configname" | jq '.modeldata.modelpath' | sed 's/"//g')
configpath=$(cat "$userhome/$datastore/$configname" | jq '.modeldata.configpath' | sed 's/"//g')

## place python myconfig_analysis file and modelfolder in the right location.
aws s3 sync "s3://""$bucketname"/"$modelpath" "$userhome/DeepLabCut/pose-tensorflow/models/"$(basename
    "$modelpath")""
aws s3 cp "s3://""$bucketname"/"$configpath" "$userhome/DeepLabCut/myconfig_analysis.py"
## Replace the video location in the config folder.
python "ncap_remote/substitute_config.py"

## Run deeplabcut analysis:
cd DeepLabCut/Analysis-tools

python AnalyzeVideos.py
cd "$userhome"/"$datastore"

find -iname "*.h5" -exec cp {} "$userhome"/"$outstore" \;
find -iname "*.pickle" -exec cp {} "$userhome"/"$outstore" \;

## Copy:
cd "$userhome"
#############################################################################################
## Stereotyped upload script for the data
upload "$outstore" "$bucketname" "$groupdir" "$resultdir" "mp4"
```

Figure 6: DeepLabCut script, written in bash. Referenced variables are provided by template scripts we wrote (see supplement). Called functions are native to DeepLabCut interface; Most of script (outside of demarked lines) are agnostic to this algorithm, and can be copied directly to a newly developed algorithm.

```
#!/bin/bash
## Bash script that establishes ncap monitoring routines with minimal dependencies on other packages.
## Load in helper functions.
execpath="$0"
echo execpath
scriptpath="$(dirname "$execpath")/ncap_utils"

source "$scriptpath/workflow.sh"
## Import functions for data transfer
source "$scriptpath/transfer.sh"

## Now parse arguments in to relevant variables:
# Bucket Name $bucketname
# Group Directory $groupdir
# Results Directory $resultdir
# Dataset Name $dataname
# Dataset Full Path $datapath
# Configuration Name # configname
# Configuration Path # configpath
set -a
parseargsstd "$1" "$2" "$3" "$4"
set +a

## Example usage:
echo "$bucketname"/"$groupdir"/"$resultdir"/logs/DATASET_NAME:"$dataname"_STATUS.txt""
## Set up Error Status Reporting:
errorlog_init

## Set up STDOUT and STDERR Monitoring:
errorlog_background &
background_pid=$!
echo $background_pid, "is the pid of the background process"

## MAIN SCRIPT GOES HERE ######################
bash /home/ubuntu/ncap_remote/run_caiman.sh
##############################################
errorlog_final
kill "$background_pid"
```

Figure 7: Analysis agnostic use case implementing dataflow toolbox, written in bash. Declares variables referenced in Figure 6.

```
{
    "PipelineName":"ncapexamplepipeline",
    "REGION":"region of service for users",
    "Lambda":{
        "CodeUri":"Codebase for \ncap Compute",
        "Handler":"Module for \ncap Compute",
        "Launch":"Whether or not to launch new pipelines. ",
        "LambdaConfig":{
            "AMI":"AMI id of the developer-configured instance",
            "INSTANCE_TYPE": "virtualized hardware instance id. ",
            "REGION": "us-east-1",
            "SECURITY_GROUPS":"network configuration",
            "IAM_ROLE":"permissions to launch new immutable analysis environments",
            "KEY_NAME":"permissions to access immutable analysis environments",
            "WORKING_DIRECTORY":"immutable analysis environment code",
            "COMMAND":"code to run to initiate processing",
            "SHUTDOWN_BEHAVIOR":"destroy immutable analysis environment after processing terminates",
            "CONFIG":"location of additional configuration parameters",
            "MISSING_CONFIG_ERROR":"We need a config file to analyze data.",
            "EXECUTION_TIMEOUT":"Additional parameters for \ncap Compute",
            "SSM_TIMEOUT":"Additional parameters for \ncap Compute",
            "LOGDIR":"Parameters for \ncap interface",
            "OUTDIR":"Parameters for \ncap interface",
            "INDIR":"Parameters for \ncap interface",
            "LAUNCH":"Launching new pipelines",
            "LOGFILE":"Logging location for diagnostic information",
            "DEPLOY_LIMIT":"Maximum number of concurrent instances to deploy",
            "MONITOR":"Enable or disable detailed monitoring"
        }
    },
    "UXData":{
    "Affiliates":[
        {
            "AffiliateName":"examplegroup1",
            "UserNames":["ian","shreya","taiga"],
            "UserInput":true,
            "ContactEmail":"The email we should notify regarding processing status."
        },
        {
            "AffiliateName":"examplegroup2",
            "UserNames":["liam","john"],
            "UserInput":true,
            "ContactEmail":"The email we should notify regarding processing status."
        }
    ]
    }
}
```

Figure 8: NeuroCAAS blueprint template declaring all relevant resources. Immutable Analysis Environments can be defined from Variables in the Lambda.LambdaConfig field, the job manager is defined in Lambda.CodeUri and Lambda.Handler. Users and permissions are defined in UXData.

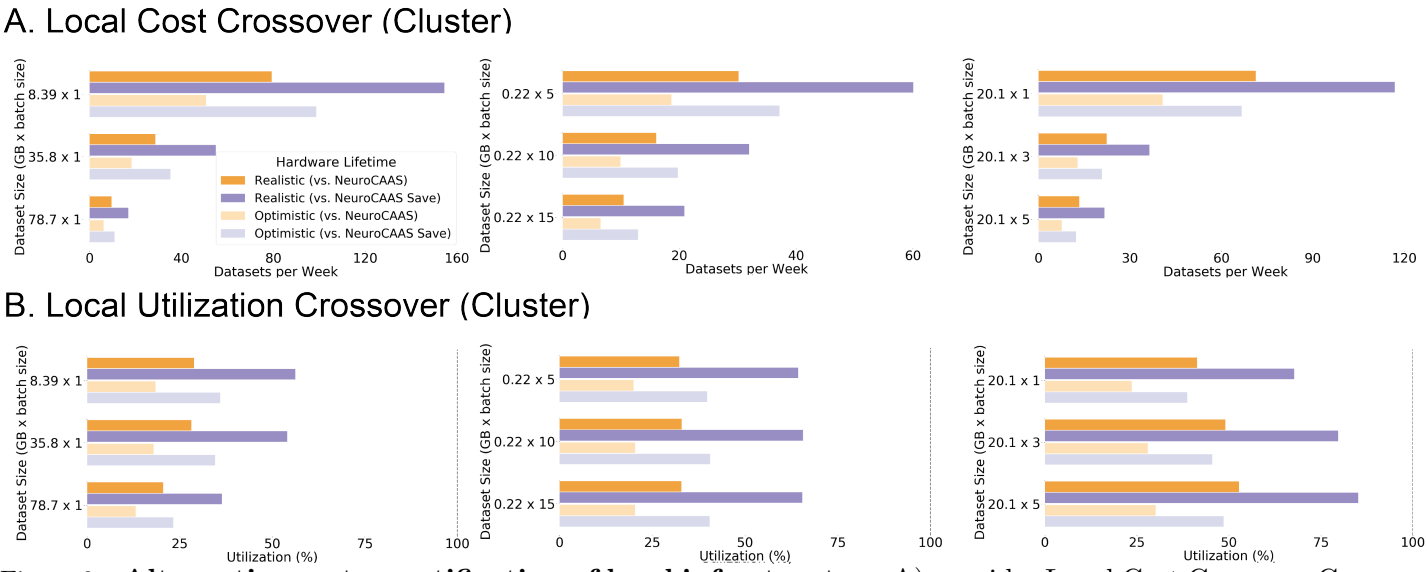


Figure 9: **Alternative cost quantification of local infrastructure** A) provides Local Cost Crossover Crossover for these resources priced as cluster compute resources, priced according to Amazon AWS's TCO calculator. B) provides the same for Local Utilization.