UC Davis
UC Davis Previously Published Works

Title
BRAND: a platform for closed-loop experiments with deep network models

Permalink
https://escholarship.org/uc/item/4vg252gb

Journal
Journal of Neural Engineering, 21(2)

ISSN
1741-2560

Authors
Ali, Yahia H
Bodkin, Kevin
Rigotti-Thompson, Mattia
et al.

Publication Date
2024-04-01

DOI
10.1088/1741-2552/ad3b3a

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

Journal of Neural Engineering

OPEN ACCESS

RECEIVED
11 August 2023

REVISED
27 January 2024

ACCEPTED FOR PUBLICATION
5 April 2024

PUBLISHED
17 April 2024

Original content from this work may be used under the terms of the Creative Commons Attribution 4.0 licence.

Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.

PAPER

BRAND: a platform for closed-loop experiments with deep network models

Yahia H Ali[1], Kevin Bodkin[2], Mattia Rigotti-Thompson[1], Kushant Patel[7], Nicholas S Card[7], Bareesh Bhaduri[1], Samuel R Nason-Tomaszewski[1], Domenick M Mifsud[1], Xianda Hou[7], Claire Nicolas[8], Shane Allcroft[9], Leigh R Hochberg[8,9,10,11], Nicholas Au Yong[6], Sergey D Stavisky[7], Lee E Miller[2,3,4,5], David M Brandman[7,12] and Chethan Pandarinath[1,6,12,*]

[1] Wallace H. Coulter Department of Biomedical Engineering, Emory University and Georgia Institute of Technology, Atlanta, GA, United States of America
[2] Department of Neuroscience, Northwestern University, Chicago, IL, United States of America
[3] Department of Biomedical Engineering, Northwestern University, Evanston, IL, United States of America
[4] Department of Physical Medicine and Rehabilitation, Northwestern University, Chicago, IL, United States of America
[5] Shirley Ryan AbilityLab, Chicago, IL, United States of America
[6] Department of Neurosurgery, Emory University, Atlanta, GA, United States of America
[7] Department of Neurological Surgery, University of California, Davis, CA, United States of America
[8] Center for Neurotechnology and Neurorecovery, Department of Neurology, Massachusetts General Hospital, Boston, MA, United States of America
[9] School of Engineering and Carney Institute for Brain Science, Brown University, Providence, RI, United States of America
[10] Harvard Medical School, Boston, MA, United States of America
[11] Veterans Affairs Rehabilitation Research & Development Center for Neurorestoration and Neurotechnology, Providence VA Medical Center, Providence, RI, United States of America
[12] Contributed equally to this work.
[*] Author to whom any correspondence should be addressed.

E-mail: chethan.pandarinath@emory.edu

Keywords: brain–computer interface, closed-loop, artificial neural network, real-time

Supplementary material for this article is available online

Abstract

Objective. Artificial neural networks (ANNs) are state-of-the-art tools for modeling and decoding neural activity, but deploying them in closed-loop experiments with tight timing constraints is challenging due to their limited support in existing real-time frameworks. Researchers need a platform that fully supports high-level languages for running ANNs (e.g. Python and Julia) while maintaining support for languages that are critical for low-latency data acquisition and processing (e.g. C and C++). *Approach.* To address these needs, we introduce the Backend for Realtime Asynchronous Neural Decoding (BRAND). BRAND comprises Linux processes, termed *nodes*, which communicate with each other in a *graph* via streams of data. Its asynchronous design allows for acquisition, control, and analysis to be executed in parallel on streams of data that may operate at different timescales. BRAND uses Redis, an in-memory database, to send data between nodes, which enables fast inter-process communication and supports 54 different programming languages. Thus, developers can easily deploy existing ANN models in BRAND with minimal implementation changes. *Main results.* In our tests, BRAND achieved <600 microsecond latency between processes when sending large quantities of data (1024 channels of 30 kHz neural data in 1 ms chunks). BRAND runs a brain-computer interface with a recurrent neural network (RNN) decoder with less than 8 ms of latency from neural data input to decoder prediction. In a real-world demonstration of the system, participant T11 in the BrainGate2 clinical trial (ClinicalTrials.gov Identifier: NCT00912041) performed a standard cursor control task, in which 30 kHz signal processing, RNN decoding, task control, and graphics were all executed in BRAND. This system also supports real-time inference with complex latent variable models like Latent Factor Analysis via Dynamical Systems. *Significance.* By providing a framework that is fast, modular, and language-agnostic, BRAND lowers the barriers to integrating the latest tools in neuroscience and machine learning into closed-loop experiments.

© 2024 The Author(s). Published by IOP Publishing Ltd

1. Introduction

In neuroscience and neuroengineering, researchers use closed-loop systems to respond to neural activity in real-time—for example, to stimulate a neural circuit or control an end effector—in order to test properties of the brain or build devices that interface with it for a therapeutic purpose. In research with intracortical brain-computer interfaces (iBCI), closed-loop systems have enabled people with paralysis to control a robotic arm, spell sentences, and move limbs with functional electrical stimulation [1–5]. These systems are referred to as 'closed-loop' because the brain controls some action via the iBCI and the user receives visual feedback from that action in real time.

Closed-loop systems must meet stringent timing requirements that are derived from the timescales of the neural processes being studied. Neuronal action potentials, or 'spikes', have a waveform on the order of a millisecond, so systems that process spiking data need to acquire the waveform at a sub-millisecond resolution and detect spike events at 1 kHz resolution. Another important factor is the control latency, i.e. the time it takes to produce a control signal from a window of neural activity. For applications such as using an iBCI to control a computer cursor, the latency from neural recording to behavioral prediction has typically been 15–20 ms [3, 6]; increasing latency is known to decrease control performance [7, 8]. We thus want a system that can receive and process spiking data at 1 kHz and predict movement intention with less than 15 ms of latency.

Several groups have released software packages for building real-time systems, but they lack the features needed to run both the highly-optimized C/C++ code that handles data acquisition and the modular Python code that modern machine learning libraries are built in. They are typically designed to run code in a limited set of programming languages, making it difficult to find a system that provides broad support for the many high-level languages used in the neuroscience field, including Python, MATLAB, and Julia. Simulink Real-Time (Mathworks) provides a visual programming interface that supports real-time MATLAB and C/C++ code [9]. RTXI and Falcon provide sub-millisecond timing guarantees but are restricted to running C/C++ code [10, 11]. These existing systems lack the language support and communication mechanisms needed to run artificial neural networks (ANNs) natively in their original programming environment, presenting a barrier to deploying ANN models in closed-loop experiments. LiCoRICE supports interprocess communication (IPC) between Python and C [12], but to our knowledge it has not yet been demonstrated in multi-computer applications, with real-time ANN deployment, or used by a wide set of labs.

Partially due to the challenge of deploying them in real-time, computational models that are promising for closed-loop neuroscience applications often go untested in a closed-loop setting. These models were developed with offline analyses on existing data, which are important for rapidly iterating on model architectures and hyperparameters but fall short of capturing the real-time response of the brain in situations involving closed-loop feedback. For example, closed-loop feedback is known to be a critical component of iBCI control, so, in the iBCI context, offline analysis can provide only a partial validation of a decoding model [7, 13]. Evaluating such models in closed-loop iBCI experiments would be a worthwhile research direction, but the space of models that can be tested is limited by the labor-intensive (and potentially error-prone) process of reimplementing models that contain millions of parameters to achieve compatibility with existing closed-loop software architectures. Thus, there is a 'translation gap' between the development of new computational models and the evaluation of those models in closed-loop experiments.

We developed the Backend for Realtime Asynchronous Neural Decoding (BRAND) to address three critical needs: (1) running closed-loop ANN inference in the same runtime environments used for offline analyses, (2) supporting several programming languages, and (3) providing sub-millisecond, high-bandwidth communication between system processes. This system uses the popular Redis in-memory database, which provides it with low-latency IPC and broad compatibility across the 54 different programming languages for which Redis client libraries exist [14]. With BRAND, researchers can easily integrate a variety of computational models into their experimental pipelines by structuring those models to read from and write to the Redis database. Computational models can then integrate with the system components responsible for signal processing, behavioral tasks, and visual feedback with sub-millisecond communication latency. This results in a system that makes it easier to prototype and study new computational models, processing techniques, and behavioral tasks in a research setting.

In this paper, we validate BRAND in three contexts: (1) high-bandwidth IPC, (2) BCI control with ANNs, and (3) neural data simulation. BRAND's IPC latency was consistently less than 600 microseconds with inputs of up to 1024 channels of 30 kHz neural data. In the BCI control benchmark, BRAND runs all of the components needed to acquire and process 30 kHz neural data over a network and produce hand movement predictions with an ANN in less than 8 ms. Using this pipeline, we conducted a closed-loop demonstration of iBCI cursor control with a participant in the BrainGate2 clinical trial (CAUTION: Investigational device. Limited by federal law to investigational use.). BRAND was also used to generate simulated neural data for both speech decoding and cursor control. With these results, BRAND is shown to be a versatile system for building a wide range of closed-loop experiments with low latencies and full support for state-of-the-art machine learning techniques. BRAND's deployment in multiple BCI groups (7, as of this writing) has resulted in many users providing feedback on its documentation, usability, and software engineering practices.

2. Methods

2.1. System architecture

BRAND follows a modular design that divides the control of a neuroscience experiment into small reusable components, each of which is designed to complete a part of the overall computational pipeline. We refer to each component as a *node*; several nodes are combined together to form a *graph*. For example, a minimal graph for an iBCI experiment might consist of a feature extraction node (e.g. computing local field potential power or spike rate), a decoding node, and an effector control node (figure 1(a)). Nodes run in parallel as separate processes, which allows us to decouple critical, highly-optimized code like data acquisition from slower, experimental code like neural network inference. Data are passed between nodes asynchronously via *Streams* in a Redis database. Each node can publish data to several output streams, and receive data by subscribing to several input streams. These streams provide a straightforward way to build a chain of interchangeable nodes that run in parallel. Running nodes in parallel allows BRAND to process incoming data at a higher rate than an equivalent system that runs sequentially (figures 1(b) and (c)). Streams are append-only logs that persist in the database even when data have already been read, so they also maintain a record of all data within an experiment and can be saved for later analysis.

In BRAND, graphs for closed-loop experiments are configured using Yet Another Markup Language (YAML). Each graph is configured by a single YAML file that lists the nodes that will be run in the graph and the parameters for each of those nodes. A script, called *supervisor*, parses this YAML file and then initializes the specified nodes as separate Linux processes. The parameters for the graph are then sent to those nodes through Redis. Once a graph is loaded, the supervisor may then start or stop nodes according to commands received via Redis. In a typical experimental session, a researcher will supply a list of graphs (as YAML files) that each configure the system for a different step of the experiment. The researcher will start and stop these graphs by sending commands to supervisor. BRAND provides flexibility in this experimental workflow, allowing researchers to control the system with the Redis command line or develop their own graphical user interfaces for selecting graphs and monitoring or adjusting the parameters of their nodes. Since Redis enables communication across multiple computers on the same network, BRAND includes another launcher script (called 'booter') that extends *supervisor's* capabilities to additional machines. To run nodes across several machines, the researcher would start a *supervisor* instance on the machine that hosts the Redis database and start a *booter* instance on each client machine. Nodes can then be configured to run on any of these machines and send and receive data from the common Redis database. Users can also configure the process priority and processor affinity for each node to make full use of the resource allocation tools available in Linux.

Redis is the open-source database system used with BRAND. The Redis database provides the interface for communicating across nodes and logging data. BRAND defines how those nodes should be configured and executed to achieve low-latency performance while allowing for collaborative development of new nodes. To maintain access to the full suite of Redis features, BRAND does not impose any requirements on the way in which programmers interface with the Redis database. However, programmers are encouraged to use the Stream data type for communication between nodes, as it facilitates data logging, and provides a standard asynchronous communication mechanism between nodes. Redis can either be configured for communication via Transmission Control Protocol (TCP) sockets, which allows multi-computer communication, or Unix sockets, which allows for faster IPC within a single machine. Libraries for Redis exist in many different languages, including C, Python, Go, Julia, and MATLAB [14]. With BRAND, the optimizations to achieve low-jitter real-time performance (PREEMPT_RT kernel, setting process priorities) are done at the system level, not at the level of the programming language, so we do not expect to see a loss of real-time performance if languages other than Python or C are used.

In summary, BRAND was designed with a modular node-graph structure that is configurable via YAML files and uses Redis for communication across multiple programming languages and multiple computers. With these design choices, we aim to make it easy to both integrate and share individual components of an experimental pipeline across experiments and labs.

2.2. Validation

We tested BRAND on the Linux operating system. Most demonstrations in this manuscript were conducted using Ubuntu 20.04 LTS and the Linux kernel with a PREEMPT_RT patch (version 5.15.43-rt45) [15] and run on a Dell Optiplex 7000 small form factor PC with an Intel i9-12900 processor and

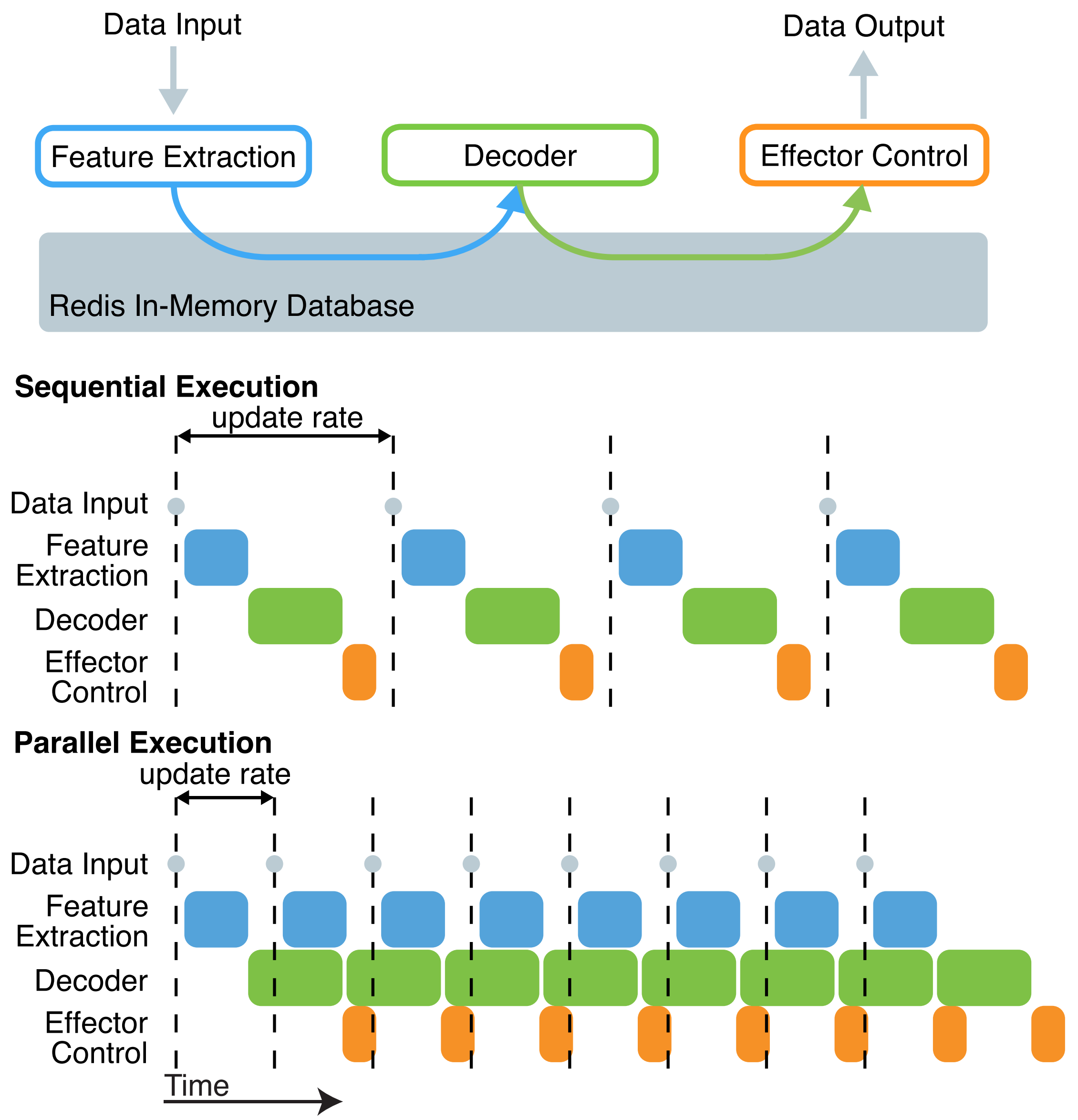

Figure 1. Software architecture schematic. (a) BRAND consists of a set of processes, or 'nodes', that each receive inputs and/or produce outputs during an experiment. (b) If nodes were run sequentially (as if they are in a script), all nodes would need to finish processing a given sample before the next one could be processed. Delays in any part of the processing chain would cause the whole system to fall behind and delay critical events like acquiring an incoming sample. (c) In BRAND, nodes run in parallel and communicate asynchronously, allowing them to maximize the rate at which data are processed and minimize the chance that delays in downstream nodes would cause the system to fall behind.

128 GB of memory. The speech simulator benchmark (figure 5(d) and (e)) was conducted on an AMD Ryzen 9 7950X processor running Ubuntu 22.04.2 LTS and Linux kernel 5.19.0–41-generic (without the PREEMPT_RT patch). The communication latency benchmarks in figures S3 and S4 were conducted on a Dell Optiplex 7000 small form factor PC with an Intel i7-12700 processor and 64 GB of memory running a Linux kernel with a PREEMPT_RT patch (version 5.15.43-rt45).

2.2.1. Communication latency

To test the communication latency of the Redis database, we ran a benchmark in which packets approximately matching the size of 30 kHz neural data from the Blackrock Neural Signal Processor (Blackrock Neurotech, Salt Lake City, Utah, USA) (NSP) were sent from a *publisher* node to a *subscriber* node. We recorded timestamps from the system clock (with Python 3.8.2's *time.monotonic_ns()* function) before the data were written to Redis in the publisher and after the data were read from Redis in the subscriber. The difference between these two timestamps was considered the 'communication latency' of the Redis database when using Unix sockets on a single machine. For each test condition we measured 300 000 pairs of timestamps. Both the publisher and subscriber nodes were implemented in Python and then compiled using the Cython package (version 0.29.18).

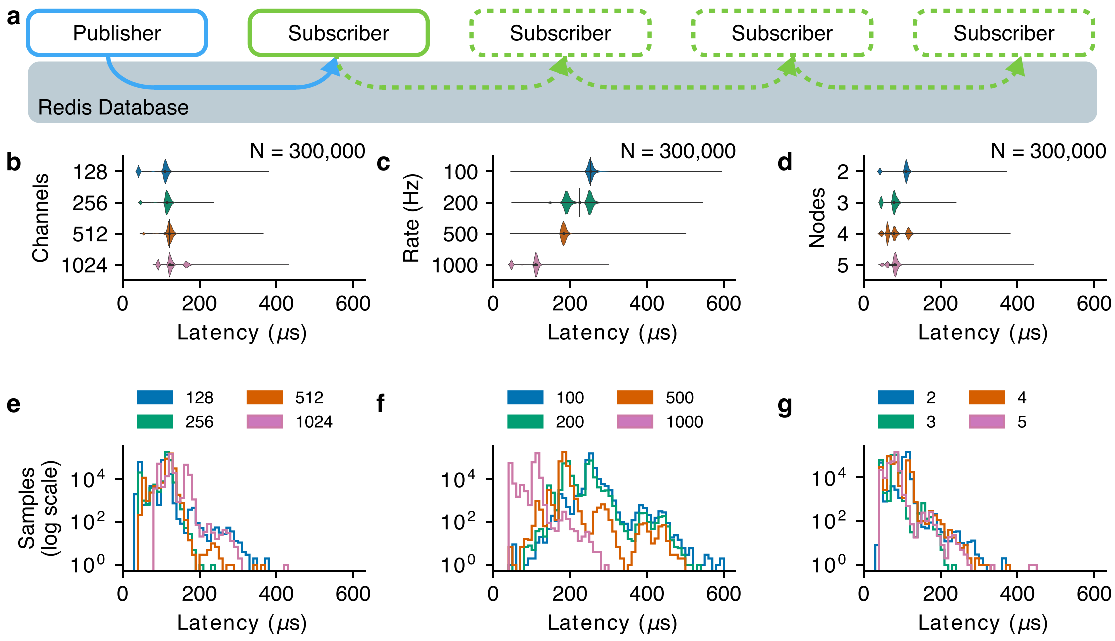

Figure 2. BRAND achieves low-latency inter-node communication. (a) To test inter-node communication latency, a publisher node sends 30 kHz neural data (grouped in 1 ms packets) to a subscriber node via the Redis database. Violin plot of the resulting latency measurements showing that the inter-node communication latency is consistently below 600 microseconds even as (b) the channel count is scaled up to 1024 channels, (c) the sampling rate is changed, and (d) additional subscriber nodes are added. Vertical lines indicate the location of the median in each violin. Histograms of these data show the distribution of latency measurements for each (e) channel count, (f) output rate, and (g) number of nodes.

We ran three variations of this benchmark when varying (1) the number of input data channels, (2) the publishing rate, and (3) the number of nodes. In all tests, publisher packets were encoded as arrays of 16-bit integers, and the number of values in each array was 30 times the number of channels (since we sent one packet of 30 kHz data every millisecond). In the first test, a publisher node sent packets at 1000 Hz and a subscriber node read each of them and logged the timestamp at which it received them. This test was repeated for four different packet sizes: 128, 256, 512, and 1024 channels. In the second test, the number of channels was held constant at 128, and the output rate was set at 100, 200, 500, or 1000 Hz. In the third test, 2–4 nodes were chained together and all but the last node forwarded the data they had received as input. This means that each subscriber node (except for the last one) writes the data it receives back into the Redis database. This test was run with 128 channels and a 1000 Hz sampling rate (figure 2(a)). The supplementary material includes additional variations of this test in which the number of channels and number of nodes are varied simultaneously (figure S3) and a lower-bandwidth signal is used to test communication latency at up to 16 nodes (figure S4).

2.2.2. iBCI control

To evaluate the latency of BRAND when performing the processing tasks needed for an iBCI, we benchmarked an iBCI control graph with two different decoders. In this benchmark, simulated 30 kHz data from two NSPs (firmware version 6.05.02) with a Neural Signal Simulator (Blackrock Neurotech, Salt Lake City, Utah, USA) were acquired over the network and then filtered with a 250 Hz high-pass filter applied forward and backwards to a 4 ms buffer of the incoming data. The filtered data were then thresholded at −3.5 times the root-mean-square voltage of each channel to detect threshold crossings within each 1 ms window. These threshold crossings were then binned into 10 ms bins and normalized before being passed to a decoder. The decoder's cursor velocity predictions were smoothed with an exponential moving average and scaled before the cursor position and task state were updated (figures 3(a) and S1(a)) [16, 17].

For neural decoding, we implemented two different decoding algorithms: an optimal linear estimator (OLE) and a recurrent neural network (RNN). The OLE decoder estimated the cursor velocity as a linear combination of the input features at a single time step. The OLE decoder had a 384-dimensional input consisting of threshold crossings and spike-band power [18] from 192 channels of neural data. Spike-band power was computed by applying a 250 Hz high-pass filter to the 30 kHz neural data, squaring the result, and averaging across time within each 1 ms window. Each feature was normalized using its mean and standard deviation from the previous period of neural recording. This decoder was fit using ridge regression, with the weight of the L2 regularization term being chosen via three-fold cross-validation.

The RNN decoder was implemented in PyTorch 1.12.1 and PyTorch Lightning 1.7.1 and consisted of a 76-unit long short-term memory (LSTM) layer followed by a two-unit linear fully-connected layer. This decoder received 192 channels of binned threshold crossings as input and applied normalization using the mean and standard deviation of the training data. Its L2 regularization, dropout, learning rate, and LSTM dimensionality were chosen using a random search on previously-collected data. During the session, this decoder was trained on four minutes of closed-loop cursor control data with the OLE decoder using an Nvidia GeForce RTX 3090 graphics processing unit (GPU).

To highlight the capabilities of the BRAND system, we also ran real-time inference with two large neural networks designed for feature extraction: the Neural Data Transformer (NDT) and Latent Factor Analysis via Dynamical Systems (LFADS). These autoencoder networks denoise neural data by modeling the dynamics of population-level activity and could be useful additions to many closed-loop neuroscience experiments [19, 20]. Both networks were run as Cython-compiled nodes in BRAND using their published PyTorch (version 1.12.1) and Tensorflow (version 2.2) implementations. Inference latencies were measured by comparing the monotonic clock timestamps between each node's input and output.

2.2.3. Simulation

We also benchmarked the timing performance of BRAND when operating as a neural data simulator. For this, we implemented two different simulators: one for cursor control and one for speech decoding. In the cursor control simulator, the human-controlled movements of a computer mouse were translated into firing rates via a cosine tuning model. In the speech simulator, audio spoken into a microphone was encoded as mel-frequency cepstral coefficients (MFCC) [21], which were then used to generate neuronal firing rates. Both simulators then generated 96 channels of 30 kHz simulated voltages from the firing rates and broadcast them using the same packet structure as Blackrock NSPs. Latency measurements were taken using timestamps logged immediately prior to writing the output of each simulator node to Redis.

2.3. Closed loop validation

2.3.1. Clinical trial participant

T11 is an ambidextrous man, 38 years old at the time of the study, who had suffered a C4 AIS-B spinal cord injury approximately 11 years prior to enrolling in the BrainGate2 clinical trial (ClinicalTrials.gov Identifier: NCT00912041). Neural data were recorded from two 96-channel microelectrode arrays placed in the 'hand knob' area of his left (dominant) dorsal precentral gyrus in the BrainGate2 trial. Data were collected on Trial Day 1321 (1,321 d after surgical placement of the arrays) and Trial Day 1491. The Institutional Review Boards of Mass General Brigham (#2009P000505) and Providence VA Medical Center granted permission for this study. Participant T11 provided informed consent to participate in this study. This study was performed in accordance with the Declaration of Helsinki. All system security, data security, and data privacy practices were consistent with institutional guidelines and the approved IRB protocols.

2.3.2. Cursor control task

We asked participant T11 to perform the classic radial-8 center-out-and-back cursor control task [22]. This research session was organized into 'blocks', which are periods of data collection lasting 3–4 min. During these sessions, T11 was seated comfortably in his wheelchair and looked at a computer monitor. We began with an open-loop cursor calibration block, in which the cursor automatically moved to targets which T11 was asked to attempt to follow with his right thumb as if using a joystick. We then trained an OLE decoder to predict the cursor's velocity from neural data [23]. In the next block, we set up the OLE decoder to control the cursor velocity and asked him to attempt to move it to the active target. Data from this block were used to train a RNN decoder to predict his intended cursor velocity. The intended cursor velocity was approximated as the vector resulting from subtracting the cursor's position from the target's position scaled to the original OLE prediction's magnitude or zeroed if the cursor reached the target [24]. In a later block, T11 was asked to perform the radial-8 task again with this RNN decoder. The initial open-loop block lasted three minutes while each of the subsequent closed-loop blocks lasted four minutes.

2.3.3. Signal processing

Neural data were filtered by the Blackrock NSP at 0.3 Hz–7.5 kHz and then broadcast in UDP packets over Ethernet. These data were acquired and published to Redis with a BRAND node and then, in other nodes, the data were common-average referenced, filtered, and thresholded to yield threshold crossing and spike-band power features. Those features were then grouped into 10 ms bins, normalized, and sent into the decoder node [6]. The decoder's predictions were smoothed exponentially and scaled according to a gain parameter and then passed to a finite-state machine (FSM) node that updated the cursor's position and task state. Finally, a graphics node rendered and displayed the task on a separate PC according to the cursor and target information sent by the FSM.

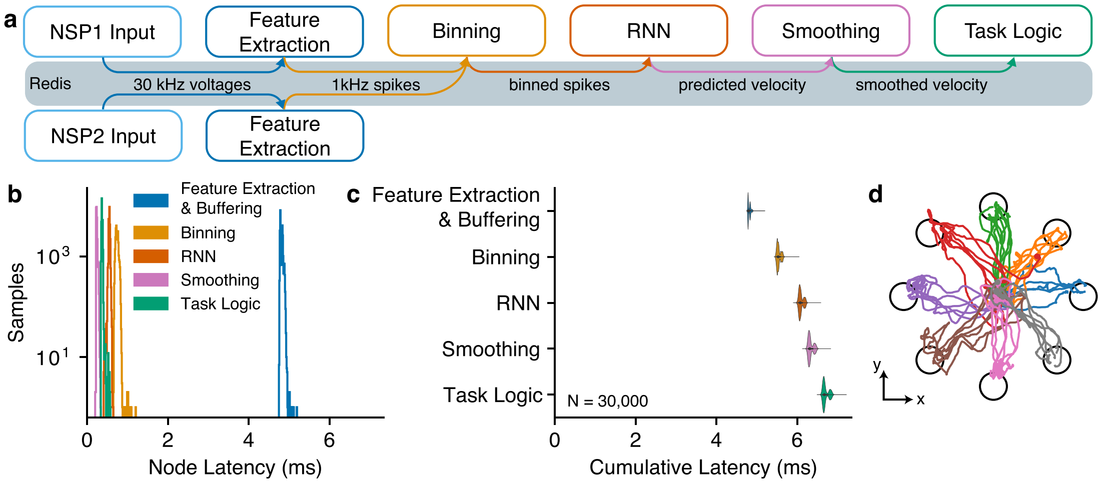

Figure 3. BRAND can be used for low-latency iBCI control. (a) To test end-to-end iBCI control latency, we ran a graph that received 30 kHz 96-channel neural spiking data via UDP (Ethernet) from two Blackrock NSPs (total of 192 channels), extracted spiking features at 1 kHz, binned spikes into 10 millisecond bins, ran decoding, and updated the location of the cursor in the task. This test used a recurrent neural network (RNN) decoder. This graph was benchmarked using simulated data. (b) Latency measurements for each node were plotted as histograms ($N = 30\,000$ packets). (c) The cumulative latency is plotted relative to the time at which each node (vertical axis) wrote its output to the Redis database. On the horizontal axis, zero is the time at which the last sample in each bin was received over the network from the NSPs. (d) Cursor positions during iBCI-enabled cursor control.

3. Results

3.1. Inter-process communication latency

We evaluated BRAND using a publisher-subscriber benchmark, where 30 kHz neural data emitted at 1 kHz (chosen to reflect the sampling rates of the Blackrock NSP system for intracranial recordings) were streamed from the publisher to the subscriber in one-millisecond packets. In this test, we found that BRAND had a communication latency of less than 600 microseconds when sending up to 1024 channels of neural data. This held true for several sampling rates and also when we increased the number of subscribers in the graph from one to four (figure 2). These results indicate that BRAND's chosen communication mechanism, Redis, is fast enough to consistently transmit the high-bandwidth data encountered in neural recordings with sub-millisecond latency.

3.2. iBCI control

To evaluate the practical application of BRAND in an iBCI control setting, we developed a benchmark in which 30 kHz neural data were acquired, filtered, and thresholded to obtain spiking features (see Methods for details). The features were then binned and passed into two different types of decoders (OLE and RNN), and the decoder predictions were sent to a node that controls the task state.

For these decoders to be considered real-time, they needed to process incoming data within a set latency deadline and with minimal jitter. In this case, for a 10 ms bin size, a new sample arrives at the decoder every 10 ms. We defined the 'per-node latency' as the difference between the time the previous node wrote its output to Redis and the time the current node wrote its output to Redis. In both cases, these times were measured immediately before calling the database write operation that produced each node's output. Real-time processing would be achieved by having a per-node latency of less than 10 ms for each of these decoders. When using the OLE decoder, we found decoding node latencies to be consistently less than 0.6 ms (figure S1). With the RNN decoder node latencies were consistently below 1.2 ms (figure 3). Both of these decoders were well below the 10 ms deadline for real-time processing.

In closed-loop tests with these decoders, participant T11 performed a task in which he moved a cursor to one of eight radially-arranged targets on a screen and then returned the cursor to the center (Trial Day 1321). T11 achieved a median target acquisition time of 1.76 ± 0.63 s with the OLE decoder and 1.79 ± 0.74 s with the RNN decoder. This performance was consistent with previous demonstrations of this now-standard cursor control BCI task [6, 25].

To further evaluate the advanced ANN inference capabilities that BRAND provides, we sought to benchmark the latencies of two state-of-the-art latent variable models: LFADS and NDT [19, 20]. LFADS and NDT were designed to improve the extraction of neural features for use in decoding, and have previously achieved a velocity prediction R^2 of 0.9097 and 0.8862, respectively in the 'MC_Maze 5 ms' dataset in the Neural Latents Benchmark [26], which is a marked improvement over the 0.6238 achieved using smoothed binned spikes. Each model was made up of several neural network layers that rely on specialized

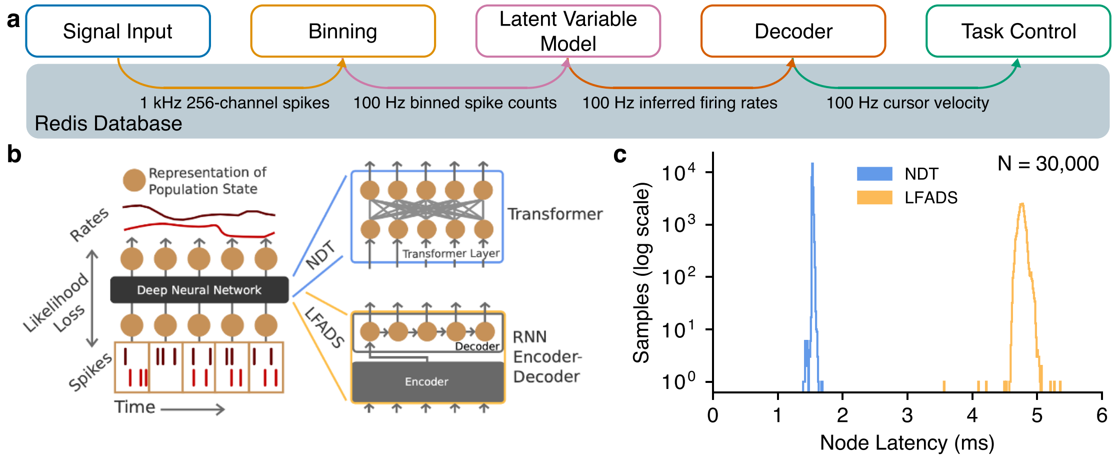

Figure 4. BRAND runs ANN latent variable models with low latency. (a) To test the inference latency of LFADS and NDT, we inserted them into an iBCI control graph that receives 256 channels of simulated threshold crossings at 1 kHz, bins them, runs inference with LFADS or NDT, runs decoding, and updates the task state. Both LFADS and NDT used a sequence length of 30 bins. (b) LFADS and NDT use different types of sequence models, an RNN and a Transformer, respectively. Reprinted from Ye and Pandarinath 2021 [19]. (c) NDT inference times were consistently below 2 ms, while LFADS inference times were consistently below 6 ms. Reproduced from [19]. CC BY 4.0.

Python libraries (Tensorflow and PyTorch) for training and inference. Previous real-time systems that lacked Python support would have required a full reimplementation of these models in another language like C or C++ to run this test [11, 27]. With BRAND, we can simply connect the existing TensorFlow and PyTorch implementations of these models to the fast Redis IPC mechanism that is used throughout the system. In our testing, LFADS inference times were consistently below 6 ms and NDT inference times were below 2 ms, with both models staying under our 10 ms latency criterion for real-time inference (figure 4).

In a later research session with Participant T11 (Trial Day 1491), we made latency measurements while he performed a cursor control task using an OLE decoder with and without the LFADS model. We found that BRAND maintained real-time performance under both decoding conditions. Without the LFADS model, end-to-end decoding latency was consistently under 8 ms. With the LFADS model, the maximum end-to-end decoding latency increased to 16 ms (figure S5).

3.3. Neural simulation

Neural data simulators have been a critical component of neuroscience research, particularly in the clinical iBCI research. Simulators enable the testing of iBCI decoding algorithms and tasks before they are used in a clinical research setting, where time with participants is extremely limited and valuable. This allows researchers to rigorously validate their signal processing, decoders, and tasks prior to an experiment. Full system simulation also helps lower the chance that software issues will impede data collection.

We evaluated whether BRAND was able to act as a real-time simulator for testing iBCI applications. With its modular design, we can use interchangeable inputs and firing rate encoding models to support simulating neural activity during different types of behavior. In this case, we tested two simulators: one for cursor control and one for a speech decoding. We demonstrated that BRAND can run both simulations in real-time. The cursor control simulator ran with less than 4 ms of end-to-end latency and the speech simulator ran in less than 3 ms (figure 5).

4. Discussion

We introduce and validate the BRAND real-time asynchronous neural decoding system, a real-time software platform that aims to meet the need in contemporary experimental neuroscience and neuroengineering for software frameworks that support both ANN inference in Python and low-latency control for intracortical BCIs. BRAND fills this need by providing three critical features: (1) running closed-loop ANN inference in the same runtime environments used for offline analyses, (2) supporting a wide range of programming languages, and (3) providing sub-millisecond high-bandwidth communication between system processes. The latency and jitter of the system were benchmarked in three settings: streaming of high-bandwidth neural data, simulation of neural data from user input, and iBCI control of a computer cursor. We demonstrated that BRAND can perform low-latency data acquisition in C and Python while also running real-time inference with multiple neural network libraries. By providing a language-agnostic framework for real-time software, BRAND reduces the need to rewrite complex computational models

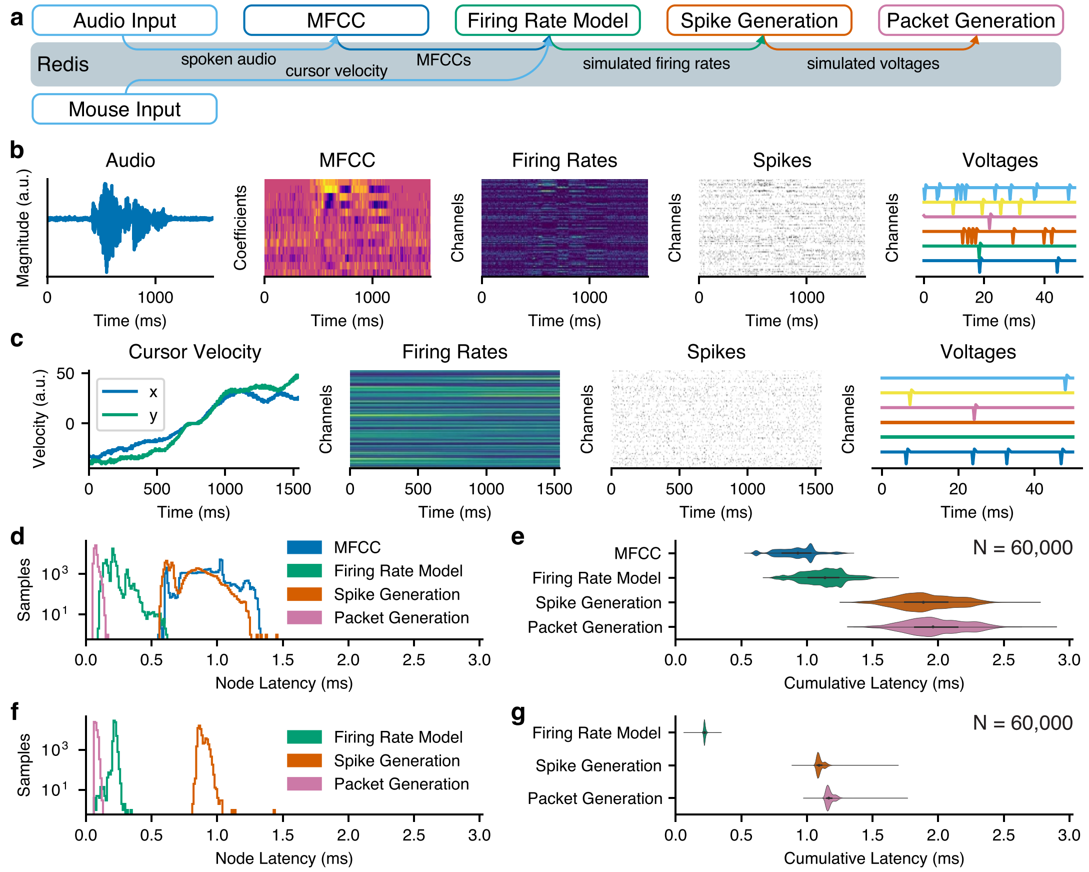

Figure 5. BRAND enables low-latency, real-time simulation of neural data. (a) In the speech simulator, spoken audio is translated into spectral features and, from there, into neural firing rates with a cosine tuning model. These firing rates are used to generate and broadcast 30 kHz voltage recordings as ethernet packets. In the cursor control simulator, computer mouse movements are translated into neural firing rates with a cosine tuning model. (b) Examples of data recorded from the speech simulator. (c) Examples of data recorded from the cursor control simulator. (d) Latency of each node in the speech simulator and (e) cumulative latency. (f) Latency of each node in the cursor control simulator and (g) cumulative latency.

when integrating them into a real-time experiment, allowing researchers to more rapidly explore and validate novel computational models.

4.1. Open-source software

The BRAND code is publicly available, and users are encouraged to develop plugins (nodes) that make BRAND work with additional recording systems, models, and tasks. The open-source approach provides two main benefits: code duplication is minimized and replicating studies across research groups is easier. BRAND's dependencies, including Python, Redis, and Linux, are also free and open-source, so users can build experimental systems without having to pay software licensing fees. The current source code is at: github.com/brandbci/brand. The code has been released under the MIT license.

BRAND's modular design supports seamless code sharing across labs. By standardizing around a common IPC mechanism with Redis, individual nodes (like models or signal processing steps) can be integrated into experimental pipelines in different labs without the need to restructure other components of each lab's systems. For example, two labs with different data acquisition systems can use the same decoding models and behavioral tasks while maintaining separate implementations of their data acquisition node. Similarly, a computational lab can develop and share a real-time implementation of their new decoding model, and labs that conduct experiments can integrate that model into their experimental pipeline without needing to modify their existing data acquisition code. Developers in the wider neurotech industry have also begun to develop tools that expand the capabilities of BRAND by adding features like real-time data visualization (figure S2).

4.2. Comparison to existing systems

Given the fundamental importance of software in performing high-quality neuroscience experiments, several groups have released software packages for closed-loop experiments. Systems like Simulink Real-Time [6], RTXI [10], and Falcon [11] provide tight timing guarantees but lack the Python support that is essential for machine learning research. Timeflux [28] and LabGraph [29] solely support Python, and thus suffer from the lack of support for lower-level languages like C that are more suitable for latency-critical applications. LiCoRICE provides Python and C support, but does so via a custom shared memory interface instead of Redis [12].

These competing systems all require developers to choose from a limited set of programming languages, which restricts the pool of existing libraries that can be used when developing experiments. For example, the machine learning community has standardized around the use of Python, and several libraries (e.g. PyTorch, TensorFlow, Keras, JAX) have been developed to perform the difficult task of training ANNs on GPUs and other hardware accelerators. Meanwhile, the C programming language has long been the foundation of Linux development and offers the low-level control of memory management that is often critical for optimizing compute latency, as well as interfacing with input/output devices and other peripherals. Other languages like Julia, R, and MATLAB are widely used for computational research. Javascript and its derivatives play an important role in web development and GUIs. C# and Java are widely used in game (and thus task) development. Dareplane, another recently-published real-time framework, has been designed to provide broad language compatibility, but unlike BRAND, it has not been validated to support the high-bandwidth communication needed for iBCIs that contain hundreds of channels [30].

We note that another system that is capable of publisher-subscriber communication among several distributed nodes is the Robot Operating System (ROS) [31]. We have not seen it used in BCI applications with the same latency requirements as our work, but, given its extensive use in robotics, it seems capable of being adapted to this purpose. BRAND's configuration, data logging, and session orchestration tools are more tailored to BCI than ROS's, so we expect BRAND to be the preferred system for BCI. If needed, BRAND can be made to interface with ROS by creating a BRAND node that communicates with ROS nodes in their expected format.

From a latency perspective, the Simulink real-time systems most commonly used for iBCI studies have been reported to have an end-to-end latency of 13–20 ms for cursor control [3, 7]. In this paper, BRAND cursor control graphs with similar decoders had end-to-end latencies of 7–9 ms (figures S1 and S5). Note, the latency measurements in [3] and [7] include the internal latencies of the Blackrock hardware (typically 6 ms) and display, while the BRAND latencies reported here do not. We thus approximate that BRAND at least matches, if not exceeds, the performance of the Simulink Real-Time systems used in previous iBCI studies.

4.3. Distributed computing

It is often useful to distribute computational tasks like signal processing, model training, and graphics rendering across multiple machines to make use of additional compute resources and avoid overloading a single computer. For this to work in a real-time system, processes running on each machine need to be able to pass data between one another with low latency. BRAND readily supports distributed computing, since Redis is designed to work across large clusters of computers that rapidly share data and coordinate to perform large-scale computational tasks. Communication with the Redis database is done with the same commands and syntax whether the database is hosted locally or remotely, so nodes can seamlessly be moved from machine to machine without changing their implementation. We used this feature extensively in our development to run simulators and GPU-related code (such as neural network training and graphics rendering) on their own dedicated machines to avoid interfering with the latency-critical signal processing that occurs on our real-time decoding machine.

4.4. Data privacy and security

BRAND is based on a commonly-used operating system (Linux) with an in-memory data store. Researchers working with closed-loop systems in human subjects research settings are responsible for ensuring their use of BRAND follows all considerations for working with research participant data (e.g. data privacy and security standards, HIPAA requirements, adherence to approved IRB protocols, practices consistent with the informed consent provided by participants). For example, the use of BRAND in this study met all IRB-approved study protocols for information security, handling of potentially sensitive information, data transfer and storage, and study participant privacy and consent. Even outside of clinical studies, many institutions have standard policies for securing Linux systems or software running in networked applications that may apply when using a BRAND system. Researchers should ensure that their network security is compliant with institutional guidelines and commensurate with the sensitivity of the data that is being handled.

4.5. Future directions

While the demonstrations in this study involved stereotyped block design-based tasks, BRAND could potentially be extended for personal use of an iBCI [32]. BRAND's modular structure provides a straightforward path to making nodes that can be hot-swapped without interrupting device use. For example, a participant may want to calibrate and deploy a new decoder every few hours to maintain high-performance control or select a different decoder for different tasks. A participant may also want to control a tablet instead of a full-sized computer. BRAND could be extended to this use case by running the Redis database on a separate computer and using an on-tablet Redis client to query the database over a local network. BRAND serves as a useful environment for prototyping this kind of personal use software in research studies.

BRAND also fills a critical need in speech iBCI research, where ANNs have enabled several recent advances in brain-to-text decoding [33, 34] and are anticipated to play an important role in the development of real-time speech synthesis BCIs [35]. Unlike text decoding, speech synthesis is expected to benefit from millisecond-scale closed-loop feedback that mimics the way in which able-bodied people can hear their own voice while speaking. BRAND is uniquely suited to this research area by providing the combination of ANN support and sub-millisecond IPC latency that is needed for a real-time speech synthesis BCI.

BRAND could also become a useful tool for neurological research in general. The development of new computational models is a critical avenue for studying neural activity across several brain functions, including movement, sensation, and cognition [36, 37]. Advances in modeling and decoding neural activity could lead to therapeutic benefits if paired with devices that interface with the brain, like responsive neurostimulators for epilepsy [38]. Existing clinical systems for responsive neuromodulation or deep brain stimulation allow streaming of data over Ethernet and could possibly be integrated into BRAND for use in research studies.

Data availability statement

The data that support the findings of this study are openly available at the following DOI: https://doi.org/10.5281/zenodo.10547705.

Acknowledgments

The authors would like to thank Participant T11, his family and caretakers, Beth Travers, Dave Rosler, and Maryam Masood for their contributions to this research. We also thank Antonio Eudes Lima, Diogo Schwerz de Lucena, and Robert Luke at AE Studio for their development of the Neural Data Visualizer. This work was supported by the Emory Neuromodulation and Technology Innovation Center (ENTICe), NIH Eunice Kennedy Shriver NICHD K12HD073945, NIH-NINDS/OD DP2NS127291 (CP), NIH-NIDCD/OD DP2DC021055, Simons Collaborations for the Global Brain Pilot Award 872146SPI (SDS), NIH-NIBIB T32EB025816 (YHA), NIH-NICHD F32HD112173 (SRN), NIH-NIDCD U01DC017844, and Department of Veterans Affairs Rehabilitation Research and Development Service A2295R and N2864C (LRH). The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health, or the Department of Veterans Affairs, or the United States Government.

Author contributions

Y H A, D M B, and C P conceived the project. Y H A, K B, M R, K P, N S C, B B, S R N, D M M, X H, and D M B contributed to software design and development for BRAND. Y H A developed and ran benchmarks to validate the system and wrote the manuscript. M R and N S C developed and benchmarked the neural data simulators. C N, Y H A, S R N, M R, and D M M conducted the experiments with participant T11. S A and S R N deployed the software and hardware needed for the T11 experiments. L R H is the sponsor-investigator of the multi-site clinical trial. D M B, C P, L E M, S D S, and N A Y supervised and guided the project. Funding was acquired by C P, S R N, Y H A, D M B, S D S, and L R H. All authors reviewed and contributed to the manuscript.

Conflict of interest

The M G H Translational Research Center has clinical research support agreements with Neuralink, Synchron, Reach Neuro, Axoft, and Precision Neuro, for which L R H provides consultative input. M G H is a subcontractor on an NIH SBIR with Paradromics. C P is a consultant for Synchron and Meta (Reality Labs). D M B is a consultant for Paradromics. S D S is an inventor on intellectual property licensed by Stanford University to Blackrock Neurotech and Neuralink Corp. These entities did not support this work, have a role in the study or have any competing interests related to this work. The remaining authors declare no competing interests.

ORCID iDs

Yahia H Ali https://orcid.org/0000-0001-8618-3837
Kevin Bodkin https://orcid.org/0000-0002-6329-7353
Mattia Rigotti-Thompson https://orcid.org/0009-0001-7298-274X
Nicholas S Card https://orcid.org/0000-0002-6858-268X
Bareesh Bhaduri https://orcid.org/0000-0002-4564-2555
Samuel R Nason-Tomaszewski https://orcid.org/0000-0002-7127-0986

Domenick M Mifsud https://orcid.org/0000-0001-8200-8193
Xianda Hou https://orcid.org/0009-0002-2066-8561
Claire Nicolas https://orcid.org/0000-0002-7761-3943
Shane Allcroft https://orcid.org/0000-0002-7903-5091
Leigh R Hochberg https://orcid.org/0000-0003-0261-2273
Nicholas Au Yong https://orcid.org/0000-0002-7898-7832
Sergey D Stavisky https://orcid.org/0000-0002-5238-0573
Lee E Miller https://orcid.org/0000-0001-8675-7140
David M Brandman https://orcid.org/0000-0003-3224-7019
Chethan Pandarinath https://orcid.org/0000-0003-1241-1432

References

[1] Hochberg L R *et al* 2012 Reach and grasp by people with tetraplegia using a neurally controlled robotic arm *Nature* **485** 7398

[2] Collinger J L, Wodlinger B, Downey J E, Wang W, Tyler-Kabara E C, Weber D J, McMorland A J, Velliste M, Boninger M L and Schwartz A B 2013 High-performance neuroprosthetic control by an individual with tetraplegia *Lancet* **381** 557–64

[3] Pandarinath C *et al* 2017 High performance communication by people with paralysis using an intracortical brain-computer interface *eLife* **6** e18554

[4] Willett F R, Avansino D T, Hochberg L R, Henderson J M and Shenoy K V 2021 High-performance brain-to-text communication via handwriting *Nature* **593** 249–54

[5] Ajiboye A B *et al* 2017 Restoration of reaching and grasping movements through brain-controlled muscle stimulation in a person with tetraplegia: a proof-of-concept demonstration *Lancet* **389** 1821–30

[6] Gilja V *et al* 2015 Clinical translation of a high-performance neural prosthesis *Nat. Med.* **21** 1142–5

[7] Cunningham J P, Nuyujukian P, Gilja V, Chestek C A, Ryu S I and Shenoy K V 2011 A closed-loop human simulator for investigating the role of feedback control in brain-machine interfaces *J. Neurophysiol.* **105** 1932–49

[8] Shanechi M M, Orsborn A L, Moorman H G, Gowda S, Dangi S and Carmena J M 2017 Rapid control and feedback rates enhance neuroprosthetic control *Nat. Commun.* **8** 13825

[9] Simulink Real-Time—MATLAB 2024 *MathWorks* (available at: www.mathworks.com/products/simulink-real-time.html) (Accessed 22 January 2024)

[10] Patel Y A, George A, Dorval A D, White J A, Christini D J and Butera R J 2017 Hard real-time closed-loop electrophysiology with the real-time eXperiment interface (RTXI) *PLoS Comput. Biol.* **13** e1005430

[11] Ciliberti D and Kloosterman F 2017 Falcon: a highly flexible open-source software for closed-loop neuroscience *J. Neural Eng.* **14** 045004

[12] Mehrotra P, Dasgupta S, Robertson S and Nuyujukian P 2018 An open-source realtime computational platform (short WIP paper) *Proc. 19th ACM SIGPLAN/SIGBED Int. Conf. on Languages, Compilers, and Tools for Embedded Systems, in LCTES 2018* (*New York, NY, USA*) (Association for Computing Machinery) pp 109–12

[13] Stavisky S D, Kao J C, Nuyujukian P, Ryu S I and Shenoy K V 2015 A high performing brain–machine interface driven by low-frequency local field potentials alone and together with spikes *J. Neural Eng.* **12** 036009

[14] Clients *Redis* (available at: https://redis.io/docs/clients/) (Accessed 15 September 2022)

[15] realtime:start [Wiki] (available at: https://wiki.linuxfoundation.org/realtime/start) (Accessed 22 May 2023)

[16] Brandman D M, Cash S S and Hochberg L R 2017 Review: human intracortical recording and neural decoding for brain–computer interfaces *IEEE Trans. Neural Syst. Rehabil. Eng.* **25** 1687–96

[17] Pandarinath C and Bensmaia S J 2022 The science and engineering behind sensitized brain-controlled bionic hands *Physiol. Rev.* **102** 551–604

[18] Nason S R *et al* 2020 A low-power band of neuronal spiking activity dominated by local single units improves the performance of brain–machine interfaces *Nat. Biomed. Eng.* **4** 973–83

[19] Ye J and Pandarinath C 2021 Representation learning for neural population activity with neural data transformers *Neurons Behav. Data Anal. Theory* **5** 1–18

[20] Pandarinath C *et al* 2018 Inferring single-trial neural population dynamics using sequential auto-encoders *Nat. Methods* **15** 10

[21] Davis S and Mermelstein P 1980 Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences *IEEE/ACM Trans. Audio Speech Lang. Process.* **28** 357–66

[22] Simeral J D, Kim S-P, Black M J, Donoghue J P and Hochberg L R 2011 Neural control of cursor trajectory and click by a human with tetraplegia 1000 days after implant of an intracortical microelectrode array *J. Neural Eng.* **8** 025027

[23] Gilja V *et al* 2012 A high-performance neural prosthesis enabled by control algorithm design *Nat. Neurosci.* **15** 12

[24] Taylor D M, Tillery S I H and Schwartz A B 2002 Direct cortical control of 3D neuroprosthetic devices *Science* **296** 1829–32

[25] Brandman D M *et al* 2018 Rapid calibration of an intracortical brain–computer interface for people with tetraplegia *J. Neural Eng.* **15** 026007

[26] Pei F *et al* 2021 Neural latents benchmark'21: evaluating latent variable models of neural population activity *Proc. Neural Inf. Process. Syst. Track Datasets Benchmarks* vol 1 (available at: https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/979d472a84804b9f647bc185a877a8b5-Abstract-round2.html) (Accessed 9 December 2021)

[27] Lopes G *et al* 2015 Bonsai: an event-based framework for processing and controlling data streams *Front. Neuroinform.* **9** 7

[28] Clisson P, Bertrand-Lalo R, Congedo M, Victor-Thomas G and Chatel-Goldman J 2019 Timeflux: an open-source framework for the acquisition and near real-time processing of signal streams *Proc. 8th Graz Brain-Computer Interface Conf. 2019* (Verlag der Technischen Universität Graz) (https://doi.org/10.3217/978-3-85125-682-6-17)

[29] LabGraph 2023 *Meta Research* (Accessed 27 June 2023) (available at: https://github.com/facebookresearch/labgraph)

[30] Dold M, Pereira J, Janssen M and Tangermann M 2023 Project dareplane for closed-loop deep brain stimulation *Brain Stimul. Basic Transl. Clin. Res. Neuromodulation* **16** 319–20

[31] Quigley M *et al* 2009 ROS: an open-source robot operating system *IEEE Int. Conf. Robot. Autom. Workshop Open Source Softw.*

[32] Simeral J D *et al* 2021 Home use of a percutaneous wireless intracortical brain-computer interface by individuals with tetraplegia *IEEE Trans. Biomed. Eng.* pp 1–1

[33] Willett F R *et al* 2023 A high-performance speech neuroprosthesis *bioRxiv Preprint* (https://doi.org/10.1101/2023.01.21.524489) (Accessed 25 April 2023)

[34] Moses D A *et al* 2021 Neuroprosthesis for decoding speech in a paralyzed person with anarthria *New Engl. J. Med.* **385** 217–27

[35] Wairagkar M, Hochberg L R, Brandman D M and Stavisky S D 2023 Synthesizing speech by decoding intracortical neural activity from dorsal motor cortex *2023 11th Int. IEEE/EMBS Conf. on Neural Engineering (NER)* pp 1–4

[36] Keshtkaran M R *et al* 2021 A large-scale neural network training framework for generalized estimation of single-trial population dynamics p 2021.01.13.426570

[37] Sani O G, Abbaspourazad H, Wong Y T, Pesaran B and Shanechi M M 2021 Modeling behaviorally relevant neural dynamics enabled by preferential subspace identification *Nat. Neurosci.* **24** 1

[38] Santaniello S, Burns S P, Golby A J, Singer J M, Anderson W S and Sarma S V 2011 Quickest detection of drug-resistant seizures: an optimal control approach *Epilepsy Behav.* **22** S49–S60

[39] ZeroMQ 2023 *ZeroMQ* (available at: https://zeromq.org/) (Accessed 28 July 2023)

[40] Schalk G, McFarland D J, Hinterberger T, Birbaumer N and Wolpaw J R 2004 BCI2000: a general-purpose brain-computer interface (BCI) system *IEEE Trans. Biomed Eng.* **51** 1034–43