Repairing Brain-Computer Interfaces with Fault-Based Data Acquisition

Cailin Winston
University of Washington
Seattle, Washington, USA
cailinw@cs.washington.edu

Caleb Winston
University of Washington
Seattle, Washington, USA
calebwin@cs.washington.edu

Chloe N Winston
University of Washington
Seattle, Washington, USA
chloewin@cs.washington.edu

Claris Winston
University of Washington
Seattle, Washington, USA
clarisw@cs.washington.edu

Cleah Winston
University of Washington
Seattle, Washington, USA
cleahwin@gmail.com

Rajesh P N Rao
University of Washington
Seattle, Washington, USA
rao@cs.washington.edu

René Just
University of Washington
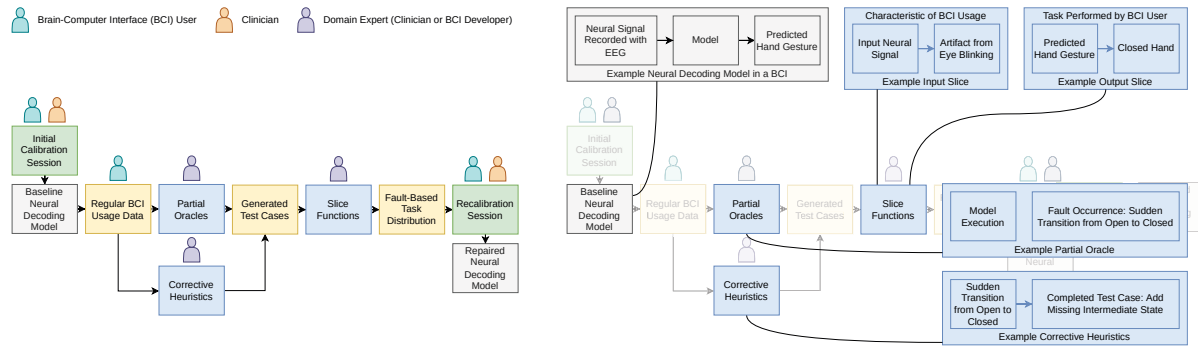Seattle, Washington, USA
rjust@cs.washington.edu

Figure 1: Proposed end-to-end methodology for repairing BCIs (left) and a concrete example for a motor decoding BCI (right).

ABSTRACT

Brain-computer interfaces (BCIs) decode recorded neural signals from the brain and/or stimulate the brain with encoded neural signals. BCIs span both hardware and software and have a wide range of applications in restorative medicine, from restoring movement through prostheses and robotic limbs to restoring sensation and communication through spellers. BCIs also have applications in diagnostic medicine, e.g., providing clinicians with data for detecting seizures, sleep patterns, or emotions.

Despite their promise, BCIs have not yet been adopted for long-term, day-to-day use because of challenges related to reliability and robustness, which are needed for safe operation in all scenarios. Ensuring safe operation currently requires hours of manual data collection and recalibration, involving both patients and clinicians. However, data collection is not targeted at eliminating specific faults in a BCI. This paper presents a new methodology for characterizing, detecting, and localizing faults in BCIs. Specifically, it proposes partial test oracles as a method for detecting faults and slice functions as a method for localizing faults to characteristic patterns in the input data or relevant tasks performed by the user. Through targeted data acquisition and retraining, the proposed methodology improves the correctness of BCIs. We evaluated the proposed methodology on five BCI applications. The results show that the proposed methodology (1) precisely localizes faults and (2) can significantly reduce the frequency of faults through retraining based on targeted, fault-based data acquisition. These results suggest that the proposed methodology is a promising step towards repairing faulty BCIs.

CCS CONCEPTS

• **Software and its engineering → Correctness**; **Software verification**; **Software safety**.

KEYWORDS

Brain-computer interface, neural decoding, partial test oracles, fault localization

Winston et al.

1 INTRODUCTION

Brain-computer interfaces (BCIs) are systems that enable communication between the brain and hardware [1, 27], by recording neural signals from the brain and/or stimulating the brain. By decoding recorded neural signals (neural decoding) or encoding neural signals for stimulation (neural encoding), BCIs can restore lost function or even enhance normal function. For example, BCIs such as neuroprosthetic limbs have restored motor function in individuals with muscle impairments or spinal cord injury. Through neural decoding, BCIs can also serve important diagnostic purposes such as detecting seizures, sleep stages, and emotion.

Because of the close interaction between BCIs and the brain, it is critical to ensure their safety and correctness. Unfortunately, BCIs can exhibit faulty behavior with decoded intentions (subsequently referred to as *tasks*) that can be (a) impossible for a prosthetic to execute, (b) undesirable for the user (e.g., wavering movements), or (c) unsafe (e.g., rapid motion of the prosthetic that could harm the user or a bystander). These are referred to as *faults*. Typically, BCIs are fine-tuned during a calibration session with the prospective user, ideally to eliminate such faults. However, data shifts can occur during use thereafter due to the non-stationarity of brain signals and differing environmental conditions not accounted for during the controlled calibration sessions. As a result, faults which were eliminated during initial calibration may reoccur. To subsequently eliminate these faults, further recalibration of the model or improved data pre-processing is required, which may involve conducting additional experiments to collect neural data while users perform tasks. Identification of tasks a BCI performs poorly on is solely based on user reports, which may not be comprehensive. To the best of our knowledge, no methodology exists for continuously testing BCIs based on a partial specification of correctness and subsequently localizing and repairing discovered faults.

We propose a methodology for testing and repairing BCIs based on expert-defined partial test oracles (partial specification of correctness), corrective heuristics (data transformation functions), and slice functions for fault localization (Figure 1). For a given class of BCIs, a domain expert such as a clinician or BCI developer defines:

- *Partial test oracles*: detect faults during BCI usage.
- *Corrective heuristics*: transform faulty model outputs by mapping implausible output values to plausible ones.
- *Slice functions*: slice and classify BCI inputs and outputs.

Together, these enable repairing faulty BCIs through targeted data acquisition informed by fault localization. Specifically, detected faults are localized to co-occurring input or output slices. Input slices correspond to characteristics of BCI usage, such as the individual using the BCI, environmental settings, and the quality of the pre-processed data that the underlying neural decoding model ingests. Output slices correspond to tasks which are the various actions that a user can perform with the BCI. Clinicians can then acquire data by targeting tasks co-occurring with faults. To repair the BCI, the underlying neural decoding model is retrained using the acquired data. Localizing faults to input slices may also be used, e.g., to improve the pre-processing of the neural signal to eliminate faults stemming from artifacts in that signal. Note that corrective heuristics transform faulty model outputs to aid localizing faults to tasks. This is distinct from repairing a BCI, which involves targeted

data acquisition, based on a task distribution, and model retraining in an attempt to reduce the likelihood of faults occurring.

As an example, consider a BCI that predicts hand gestures from neural signals. A domain expert may define a partial test oracle that identifies sudden transitions between hand positions as faults, a corrective heuristic function that corrects a sudden-transition fault by smoothly interpolating between the hand positions, and slice functions that classify the various hand gestures. If instances of such sudden-transition faults occur more frequently for slices of the heuristically transformed output that correspond to a closed gesture (i.e., the fault is localized to the closed gesture task), clinicians and users would be recommended to collect additional data for this task. Further calibration of the gesture prediction model can reduce the number of sudden-transition faults. The domain expert may also define a slice function that classifies input slices that contain artifacts from eye blinking (a characteristic interference in EEG data). If faults are localized to these input slices, BCI developers could be asked to improve pre-processing to eliminate such artifacts.

We evaluated the proposed methodology on five BCI applications. The results show that (1) partial test oracles can detect faults in BCIs, (2) fault localization can identify input and output slices co-occurring with faults, and (3) targeted, fault-based data acquisition can significantly reduce the frequency of faults in BCIs.

The key contribution of this paper is an end-to-end, human-in-the-loop methodology for repairing deployed BCIs. Specific contributions include:

- A classification of BCI applications and faults (Section 3).
- A novel application of partial test oracles and fault localization to BCIs, enabling targeted fault-based data acquisition to repair faulty BCIs (Section 4).
- A validation of the proposed methodology using five BCI applications, demonstrating that faulty BCIs can be successfully repaired post-deployment (Sections 5 and 6).

2 BACKGROUND
2.1 Brain-Computer Interfaces

Many BCIs involve translating neural signals, which are recorded from the brain and reflect a person's intended action (task), into control signals (e.g., for a prosthetic limb in a paralyzed person, for an avatar in an augmented or virtual reality environment for rehabilitation therapy, or for non-medical applications such as brain-controlled gaming.) A BCI has components for signal acquisition, signal processing and feature extraction, and translation of these signals into a meaningful output through neural decoding. Neural signaling in the brain utilizes electrical impulses communicated between neurons. This electrical activity can be measured through a variety of techniques, ranging from highly invasive procedures involving brain implants to noninvasive methods such as electroencephalography (EEG). The signal is often recorded as a 2D array, capturing time and input channels—streams of input from electrodes placed on specific places on the scalp (e.g., EEG) or within the brain (e.g., intracortical recordings). The recorded brain activity can be affected by artifacts, such as line noise from external electrical impulses and movement-related artifacts. The brain signal can also be weak and mixed due to signal propagation through biological tissue before measurement. Signal processing and feature extraction are typically used to reduce noise and extract task-relevant features
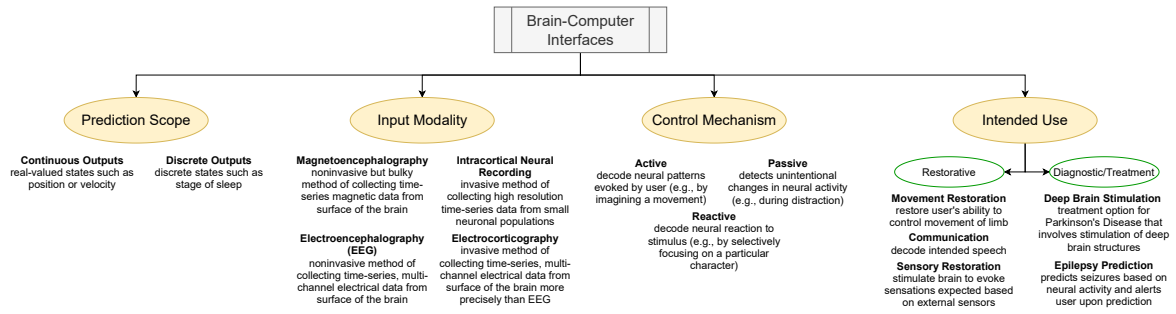
Figure 2: Classification of Brain-Computer Interfaces

from the data. Even with such processing, noise often persists in the signal and can result in faults.

The processed neural data is then translated into a desired output using a neural decoder. While some neural decoders are hand-crafted models with highly interpretable parameters, other decoders utilize classical machine learning methods and, more recently, deep neural networks that can model complex patterns in data with less manual parameter tuning. The type of decoder is determined based on the application and the complexity of the neural data.

Neural decoders are often trained from hand-labeled datasets collected from studies with several participants. However, before deploying such models to a device, such as a prosthetic, a calibration session is required with the individual user, due to the variability of neural signals across individuals [21]. In this session, neural data is collected while the individual performs or imagines performing certain tasks. For example, for a hand prosthetic, an individual may be asked to move their hand in specified directions and grasp an item. The model is calibrated on the neural data that is collected while the user performs these tasks, so that it can accurately decode the neural data and predict the true intention of the individual.

2.2 Challenges in Deploying BCIs

There are multiple challenges associated with deploying accurate BCIs. One challenge is the need for extensive data collection and training of the neural decoder. In the successful operation of any BCI, two systems are at play—the neural decoder and the user's brain. While the neural decoder learns neural patterns that correspond to specific user states, the user's brain adapts to control the BCI. Consider a motor BCI whose goal is to perform movements desired by the user. First, the neural decoder is calibrated [22]. Neural activity is recorded while the user imagines various movements (imagination is required when the individual is paralyzed or otherwise unable to physically perform the desired task) [22]. This requires hours of experiments in the clinic, with the user performing many tasks that the BCI would be expected to perform. The data must also be manually labeled and annotated in some applications, and this may be error-prone. A neural decoder is fit to this data. Next, the user begins using the BCI and learning how to control it. If faults persist, further calibration may be required.

Another challenge is that in current BCI applications, it is hard to monitor and characterize faults after deployment. If a BCI performs poorly, a user may report failure scenarios, which a clinician may use to prescribe additional data collection sessions. Faults that are

not reported are not necessarily accounted for, and thus it is hard to assess the performance and safety of the device.

2.3 Software Testing and Debugging

Recognizing the need for a methodology for identifying and repairing faults in BCIs post deployment, we explore the applicability of existing software testing constructs to this emerging domain. Partial test oracles, which encode a partial specification of correct behavior, have been extensively used to automate software testing for complex systems (e.g., [3, 16, 30, 34]. While some partial test oracles specify which properties of a system's output must hold for any input, others specify such properties with respect to a given input or set of inputs. In our use case, partial test oracles test a property of neural decoding model executions.

Software debugging for traditional software systems (as opposed to learned ML models) involves localizing a fault to its root cause in the program's source code as well as repairing that fault. The software engineering research community has developed a variety of approaches to aid and automate software debugging. For example, statistical fault localization approaches aim to rank program elements by their suspiciousness [25, 35]. Similarly, program slicing approaches [33] aim to identify slices of a program of interest, based on slicing criteria (e.g., derived from data- or control-flow properties.) Information about a fault's root cause can then be used to repair the faulty system. The key difference between our application of such debugging techniques and the application to traditional software systems is the focus on data as opposed to programs.

3 CLASSIFICATION OF BCI APPLICATIONS AND FAULTS

Prior work has classified BCIs based on system characteristics and target users and tasks (e.g., [20, 31]). Since differences in BCIs have implications on the ability to define partial test oracles, corrective heuristics, and slice functions, we expand upon existing classifications of BCI applications and additionally classify BCI faults.

3.1 Classification of BCI Applications

Our classification focuses on four key dimensions along which BCI applications can be classified (Figure 2).

Prediction Scope   BCIs vary in their prediction scope. BCIs may predict continuous outputs (e.g., position and velocity for motor decoding BCIs) or discrete outputs (e.g., stages of sleep).

Table 1: Classification of BCI Faults.

Category	Fault	Description	Implication	BCI Application(s)*
Input Validation	Artifact (A)	Signal noise from irrelevant motor or eye movement	Interference with neural signal input to decoder	—
Temporal Validation	Temporal Inconsistency (TI)	Flickering in decoder predictions	Undesirable behavior	MD, OE, CC, FN, SS
	Illegal Transition (IT)	Impossible/improbable transition between predicted states	Impossible/undesirable behavior	MD, SS
	Rapid Motion (RM)	Improbably large change in prediction	Undesirable/unsafe behavior	CC, FD
Consistency	Multimodal Inconsistency (MM)	Disagreement between predictions by decoder and by auxiliary model	Potentially incorrect prediction	SS
Domain Knowledge	Out of Bounds (OB)	Impossible prediction	Impossible behavior	CC

*Evaluated BCI Applications (Table 2). MD: Motor Decoding, OE: Observe-or-Execute Classification, SS: Sleep Stage Classification, CC: Cursor Control, FN: Field Navigation.

Input Modalities   BCIs use various types of input modalities (i.e., types of neural data analyzed). These range from noninvasive techniques such as magnetoencephalography and electroencephalography (EEG) to invasive but more precise techniques such as intracortical neural recordings and electrocorticography. Despite differences in techniques, most modalities yield time-series data that are occasionally multi-channel. Specific features, such as frequency components of signals or average firing rates, may be computed from raw data and used to predict states such as intended movement.

Control Mechanism   BCIs vary in their control mechanisms (i.e., how a user interacts with the BCI.) Some motor BCIs require users to actively control their neural patterns through imagination to signal their intended movement. Other BCIs detect specific types of neural activity that are evoked upon sensory stimulation. In this way, such BCIs may be able to detect whether the user is selectively focusing on an object, such as a character in a speech-decoding BCI. Still other BCIs may detect changes in the neural activity that are not intentionally modulated by the user.

Intended Use   BCIs vary in their intended use. Some are restorative, aiming to provide sensory or motor function. Others are diagnostic or treatment-centered. Examples of the latter category include deep brain stimulation, which stimulates deep regions of the brain and has been found to mitigate symptoms of Parkinson's Disease and epilepsy, and epilepsy predictors that may alert the user upon prediction of a seizure [8].

3.2 Classification of BCI Faults

We classify BCI faults into four non-disjoint categories, summarized in Table 1: input validation, temporal validation, consistency across modalities and models, and domain knowledge violation. Categorizing faults in such a way motivates our methodology in which faults are detected through partial test oracles (Section 4).

Input validation   Neural data can be noisy due to muscle movement and electrical interference. Careful pre-processing and feature extraction to remove the resulting artifacts in the input data is critical. Since the inputs that a model sees during training likely have few artifacts due to controlled experimental conditions, artifacts remaining in the input data due to faulty pre-processing are a particular concern for regular usage of the BCI after deployment.

Temporal validation   Model output may be temporally incorrect. In some applications, for example, the output class should not

change too rapidly between certain states or within a small range of the output space. Indeed, one of the challenges in developing prosthetics is producing smooth limb movement when operating in free space or when precisely grasping small objects [24]. Furthermore, certain state transitions are invalid or improbable, due to physiological constraints. Thus, even though the neural decoder itself may be time-independent and may have no constraints on the validity between disjoint predictions of the model, temporal faults can occur, since the model is utilized in a time-dependent context.

Consistency   If present, auxiliary data such as respiration, heart rate, or movement may be utilized to enforce consistency between the auxiliary information and the neural decoder's predictions. Furthermore, some BCIs utilize multiple models or data from different modalities of recording neural signal to make predictions. For such BCIs, the outputs across these models or modalities must be consistent. For example, the presence of an artifact in the pre-processed data must be consistent across different recording modalities.

Domain knowledge   Domain knowledge about the expected behavior of a system can aid testing. For example, there may be physical constraints on limb positions, such as invalid finger positions for a hand grasping BCI, and there could be patterns in model output that are known to be improbable based on domain knowledge. Knowledge of an individual user's health conditions and daily activities may also be applied to enforce constraints.

4 REPAIRING BCIS WITH FAULT-BASED DATA ACQUISITION

This section details the proposed methodology depicted in Figure 1. Sections 4.1–4.3 describe partial test oracles, corrective heuristics, and slice functions, and section 4.4 details how these components are used for targeted, fault-based data acquisition and repair.

4.1 Detecting Faults with Partial Test Oracles

Our methodology uses partial test oracles as a means to test BCIs and detect faulty outputs of the underlying neural decoding model. The behavior and expected output of a neural decoder usually cannot be precisely specified, which makes partial oracles especially suitable for testing. These oracles capture domain knowledge of a partial specification of a BCI—specifically, what properties of the neural decoder output violate necessary conditions.

Partial Test Oracles for Input Validation   Software and hardware systems usually have defined preconditions that must be met for valid execution. In software systems for example, these preconditions can be expressed through assertions. In the context of BCIs, input validation is similar to such preconditions: if pre-processed neural data is dissimilar to the type of data a decoder was trained on, the decoder cannot be expected to produce valid output. Artifacts that occur in neural data due to unrelated movement and interference with the neural data collection are often removed by signal processing before the data is passed through the neural decoder. Partial test oracles can be defined to capture cases in which these artifacts do not get removed from the data input to the decoder. These cases can then be later analyzed and used to either improve the signal processing steps or improve the generalizability and robustness of the decoder.

Partial Test Oracles for Temporal Validation   Partial test oracles can also capture neural decoder faults that are temporal. Oracles for temporal faults utilize ideas from metamorphic testing in that previous model output is considered when testing subsequent model output. Metamorphic relations are useful, for example, for detecting illegal transitions in model output.

Partial Test Oracles for Consistency   Partial test oracles can detect inconsistencies between different models or across different modalities of neural recording. For example, a simpler, precise auxiliary model that accepts the same input as a neural decoder, or related data, can be used to identify faults in the neural decoder's output.

Partial Test Oracles for Domain Knowledge   Test oracles can also capture faults based on domain knowledge of the likelihood of various events. For example, clinicians may have noticed in experiments that a certain state transition in the model output space is extremely unlikely in natural execution or that a particular position is physically impossible or unsafe. Such experience-based constraints can be defined in form of a partial test oracle.

4.2 Transforming Implausible to Plausible Model Outputs with Corrective Heuristics

While partial test oracles can detect faulty predictions, correcting these predictions is challenging because the correct output is unknown—as is the user's activity during regular usage of the BCI that resulted in the fault occurring.

Our methodology uses corrective heuristics that transform faulty predictions by mapping implausible output values to plausible ones. A different corrective heuristic is defined for each type of fault. For example, a corrective heuristic may transform an illegal transition by changing the faulty prediction to the previously predicted state or to a state that is reachable from the previous state. Similarly, a corrective heuristic may transform an out-of-bounds coordinate to the nearest in-bound coordinate. While there is no guarantee that the resulting value is itself correct, it is at least plausible.

Corrective heuristics may utilize auxiliary information, such as a state transition probability matrix learned on a ground truth dataset, while others may generate corrections based on the neural decoder's predictions on surrounding data points. Corrective heuristics for neural decoders are specific to a fault in a specific BCI application, but they do share some common characteristics. First, unlike labeling functions used for weak supervision [28], corrective heuristics only apply to faulty predictions. Second, since BCIs operate on time-series data (i.e., neural signals over time), corrective heuristics rely on previous predictions. For example, a corrective heuristic for illegal state transitions may assume that the previous decoder predictions were correct and may select the most common recently predicted state as the plausible output.

While corrective heuristics can transform individual faulty predictions, they do not truly repair the neural decoding model itself, and hence cannot be expected to generalize to long-term BCI usage. Consequently, our methodology uses corrective heuristics to improve fault localization but not as a standalone solution to repair.

4.3 Localizing Faults with Slice Functions

Our methodology uses slice functions to localize a detected fault to co-occurring input slices and output slices. Guided by domain knowledge, these slice functions capture characteristics of the input and map outputs to tasks, given a stream of neural decoder model executions. For applications with discrete output, output classes can be directly mapped to tasks; for applications with continuous output, the output is discretized into bins, which correspond to tasks.

While output slices are directly related to tasks that the user might perform during daily usage of the BCI, input slices are related to subtler characteristics such as data artifacts, environment settings, or characteristics of the user. Localizing faults to input slices enables possible refinement of tasks (e.g., performing a task in a particular environment setting) and improvement of BCI components such as pre-processing (e.g., artifact removal).

This paper studies the effectiveness of localizing faults to both input and output slices but only uses the output slices (tasks) to guide targeted data acquisition by generating a fault-based task distribution. We conjecture that faults localized to input slices can inform BCI developers about necessary improvements of data-processing components. We leave a deeper exploration as future work.

4.4 Retraining with Fault-Based Data Acquisition

Our methodology uses fault-based data acquisition that is focused on eliminating specific faults. Simply retraining a neural decoder on heuristically corrected data or randomly sampling data is insufficient. First, heuristically corrected data, when directly used for retraining, would be inherently noisy. Second, non-targeted data acquisition is inefficient because faults are not uniformly distributed across tasks. Targeted, fault-based data acquisition, by contrast, yields a more focused training dataset.

Data acquisition is the standard method for training and recalibrating BCIs for users' needs. However, the traditional approach to data acquisition is not targeted at eliminating particular faults. For example, a user's report that the BCI sometimes rapidly switches between acquiring and releasing more than once for a single grasping task has an unknown root cause and cannot be easily addressed.

Using partial test oracles, corrective heuristics, and slice functions, our methodology computes a coincidence distribution of tasks based on fault occurrences from past BCI usage. This computed distribution of tasks may indicate that a particular task (e.g., a stage of acquiring or grasping) tends to coincide with faults more often and thus requires more training data to inform the model's predictions for that faulty task. While the example of simple grasp control

Table 2: Summary of BCI Applications Used for Evaluation.

BCI Application	Input Modality	Output Prediction	Control Mechanism	Intended Use	Decoder Model*	Performance Metric
Motor Decoding (MD)	EEG	One of 3 hand postures	Active	Restorative	LSTM	Classification Accuracy
Observe-or-Execute Classification (OE)	EEG	Observation or execution	Active	Restorative	SVM	Classification Accuracy
Sleep Stage Classification (SS)	EEG	One of four sleep stages	Passive	Diagnostic	RF	Classification Accuracy
Cursor Control (CC)	SNR	2D cursor position	Active	Restorative	WCD	Mean Squared Error
Field Navigation (FN)	SNR	2D position in field	Active	Diagnostic	Neural Net	Mean Squared Error

*SNR: Single neuron recordings; SVM: Support vector machine; RF: Random forest classifier; WCD: Weiner Cascade Decoder.

has a small set of tasks, BCIs that control a wider range of hand gestures perform a larger set of tasks. The more tasks a BCI performs, the more important it is to acquire training data according to a task distribution to effectively target and eliminate faults.

The output slice functions are applied to the heuristically corrected model outputs to determine a more accurate task distribution (e.g., an illegal state or out-of-bounds prediction cannot be directly mapped to a task). The task distribution is then used to retrain the underlying model.

5 EVALUATION APPROACH

Section 4 proposes a methodology for detecting and repairing BCI faults. To evaluate the efficacy of the individual components of this methodology, we studied the following three research questions:

- RQ1: Can partial test oracles precisely detect BCI faults?
- RQ2: Can corrective heuristics accurately transform faulty model outputs?
- RQ3: Can BCI faults be localized to input and output slices?

To evaluate the efficacy of BCI repair via targeted, fault-based data acquisition, we answer the following two research questions:

- RQ4: Can fault-based data acquisition and model retraining repair BCI faults?
- RQ5: Are corrective heuristics or fault localization sufficient on their own?

To answer our research questions, we selected 5 different neural decoding BCI applications (summarized in Table 2). Each decoder uses neural data (EEG or single neuron recordings) in some model species (human, monkey, or rat) to decode an aspect of behavior (motion or sleep). We used publicly available labeled datasets and existing implementations of neural decoding models.

We split each ground truth labeled dataset into a training set for initial training and calibration, an observation set to simulate regular usage, an acquisition set that can be sampled from to simulate clinical data acquisition, and a testing set for evaluating performance. We generally split each dataset based on a 6:2:1:1 ratio. For the Field Navigation application, we observed during initial exploration that 20% of the data had substantially lower quality than the rest; we discarded the data. The baseline model of the Cursor Control application had a fault frequency very close to zero, which would have rendered it as not suitable for our evaluation. We discarded 20% of the training data, still leaving the Cursor Control application with a substantially lower fault frequency than all other applications (Table 4).

We first trained each neural decoder on the training set to achieve a baseline model. We then made predictions on the observation set using the baseline neural decoder, simulating regular usage of the BCI and capturing fault occurrences. We used the collected faults to perform some method of repair and re-assess the performance. The model performance was always computed on the testing set. For the results on performance and fault frequency, we ran ten trials using different splits of the dataset and averaged the results.

5.1 Selected BCI Applications

We selected five neural decoder applications (Table 2), aiming at a representative selection based on the classification in Section 3.1. Specifically, we selected both discrete-output and continuous-output decoding BCI applications, those that use either EEG or single-neuron recording (SNR) as the input modality, both active and passive BCIs, and both restorative and diagnostic BCIs. Selecting BCIs that vary along the four dimensions enables us to more comprehensively evaluate our proposed methodology. For example, BCIs with discrete output classes allow for testing output validity based on state transition violation, while those with continuous output space require assertions on the magnitude and rate of change of the output and also make defining accurate corrective heuristics more challenging. Furthermore, classical ML models usually operate on manually engineered features, while deep neural networks do not require such feature engineering, and this may have implications on the ability to repair faults.

5.1.1 EEG Motor Decoding (MD). We used the EEG Motor Movement/Imagery Dataset from Physionet to train an LSTM-based neural decoder [12, 29, 37] to classify three fist positions. EEG data was recorded from 64 electrodes placed in precise locations on the scalp while participants closed their left fist, right fist, or neither based on the location of a cursor on a screen. This decoder has immediate applications in a hand prosthetic that must decode the user's intention to open or close a particular fist to grasp an object.

5.1.2 Observation-or-Execution Classification (OE). In this application, a support vector machine (SVM) model was used to classify between action observation and motor execution of the sit to stand action [5, 6]. EEG and electromyography (EMG) data were collected while participants were shown a video stimulus of someone performing the sitting to standing task (action observation), given a rest period, and then asked to execute that task (motor execution). The data collected while executing the action was processed to extract movement related cortical potentials before being classified. This decoder has applications in motor prosthesis, because distinguishing the intent of a user to perform an action vs. a passive thought or observation of an action is critical for correctness.

Table 3: Corrective Heuristics for BCI Faults.

Fault	BCI Application(s)*	Description of Corrective Heuristic
Temporal Inconsistency	OE, SS	Select the most common state predicted by the decoder on surrounding data points
	CC, FN	Compute the average of decoder predictions on surrounding data points
Illegal Transition	MD	Sample from state transition probabilities learned on training dataset
	SS	Select a state predicted by decoder on previous data point
Rapid Motion	FN	Compute the mean of the most recent outputs or closest possible output
Multimodal Inconsistency	SS	Compute the most common state predicted by the decoder on previous data points
Out of Bounds	FN	Compute the mean of the most recent outputs or the closest in-bounds value

*MD: Motor Decoding, OE: Observe-or-Execute Classification, SS: Sleep Stage Classification, CC: Cursor Control, FN: Field Navigation.

5.1.3 Sleep Stage Classification (SS). For application diversity, we selected a passive (diagnostic) BCI which classifies sleep/wake stages into five possible outcomes. We used the Sleep-EDF dataset containing whole-night PolySomnoGraphic EEG recordings and a Random Forest Classifier to classify between stages [7, 13]. Such a BCI can help monitor and detect abnormalities in individuals sleep patterns that could signify underlying medical conditions.

5.1.4 Cursor Control (CC). In this continuous decoding application, a Wiener Cascade decoder predicts the x- and y- coordinates of a cursor on a 2D screen controlled by a monkey physically manipulating a manipulandum. We used a dataset that contained single-cell recordings from 52 neurons in the brain and was acquired from the Kording Lab [11]. The ability to control a digital cursor with one's mind has significant implications for individuals with impaired mobility and communication skills.

5.1.5 Field Navigation (FN). The final application involved using a deep neural network to predict the x- and y-coordinates of a rat navigating towards rewards on a platform [11]. We utilized another dataset from the Kording Lab, which contains single-cell recordings from the rat's hippocampus, a part of the brain involved in memory. Such motor movement-related BCI applications have practical applications for developing prostheses.

5.2 Partial Test Oracles

We defined partial test oracles for each of the five BCI applications. Most of these oracles are stateful (i.e., state is maintained across multiple executions of the decoder model) and are similar to metamorphic relations [30]. Given each new model execution, a partial test oracle specifies what executions among the current and past consecutive model executions are faulty, if any.

5.2.1 EEG Motor Decoding (MD). In this application, we identified *illegal state transition* and *temporal inconsistency* faults, both of which fall under the category of temporal validation. First, based on the experimental setup for this application, it is impossible for the participant to directly switch between having their left fist closed and right fist closed, and thus transitions between these states can be considered "illegal." Furthermore, due to the presence of noise in the input EEG signals and limitations of the baseline decoder, the model's prediction may flicker rapidly between states

over a period of time instead of consistently remaining in a certain state for a reasonable amount of time (*temporal inconsistency*).

5.2.2 Observation-or-Execution Classification (OE). For the observe-or-execute classification application, we focused on a temporal inconsistency fault: an individual cannot rapidly switch between action observation and motor execution since they remain in each state for a non-trivial period of time, and thus the output of the decoder should not flicker between those states.

5.2.3 Sleep Stage Classification (SS). For the sleep stage classification application, we cover three faults, of which two fall into the category of temporal validation. We identified an *illegal transition* fault as it is physiologically impossible to transition between certain stages of sleep, and a *temporal inconsistency* fault because individuals remain in one stage of sleep for an extended period of time. The third partial test oracle (for the *multimodal inconsistency* fault) identified inconsistencies between the neural decoder's output and the output of a binary classifier that classifies whether the BCI user is awake or sleeping based on auxiliary data, including respiration, body temperature, and EMG data.

5.2.4 Cursor Control (CC). For the cursor control application, we identified two different temporal faults, temporal inconsistency and rapid motion. First, the BCI can make multiple faulty predictions of position values that change significantly in a short period of time (*temporal inconsistency*) when the cursor movement should be smooth. Second, the BCI can consecutively predict two position values whose distance exceeds a given threshold (*rapid motion*), when the cursor should move at a slower pace.

5.2.5 Field Navigation (FN). For the field navigation application, we defined three faults. First, the BCI can make an *out of bounds* prediction, incorrectly predicting a position coordinate that lies outside the field. This is an example of a violation of domain knowledge. Second, we detected *temporal inconsistency* faults in this application: the position in the field cannot change too rapidly due to physical limitations to rat movement. Third, we defined the *rapid motion* fault as predictions with unreasonable sudden jumps.

5.3 Corrective Heuristics and Slice Functions

We defined corrective heuristics for each of the five BCI applications. Table 3 summarizes these heuristics and the faults they apply to.

Table 4: Frequency of BCI Faults Detected by Partial Test Oracles. Precision is the percentage of correctly detected faulty outputs, given ground truth. *For continuous-output applications the precision is generally unknowable without defining an epsilon up to which model outputs are considered equal; for some faults such as out of bounds (OB), however, precision is guaranteed to be 100%.

BCI Application	Fault	Precision	Frequency
Motor Decoding (MD)	Temporal Inconsistency (TI)	76%	12.7%
	Illegal Transition (IT)	91%	3.13%
Observe-or-Execute Classification (OE)	Temporal Inconsistency (TI)	62%	37.63%
Sleep Stage Classification (SS)	Temporal Inconsistency (TI)	59%	26.6%
	Illegal Transition (IT)	72%	7.66%
	Multimodal Inconsistency (MM)	94%	7.35%
Cursor Control (CC)	Temporal Inconsistency (TI)	—*	0.69%
	Rapid Motion (RM)	—*	0.26%
Field Navigation (FN)	Temporal Inconsistency (TI)	—*	7.89%
	Rapid Motion (RM)	—*	7.89%
	Out of Bounds (OB)	100%*	1.28%

For each of the five BCI applications, we defined slice functions for capturing discrete input and output slices, given a stream of neural decoder model executions. For the three applications with discrete output (motor decoding, observe-or-execute classification, and sleep stage classification), we defined output slice functions that map output classes to tasks. For both the Cursor Control and the Field Navigation application, we defined output slice functions to classify the direction of movement of the cursor or the rat, respectively. For the Cursor Control application, we additionally defined output slice functions to classify the position (mapping each coordinate to a quadrant subspace of the screen) of the cursor.

Both the Cursor Control application and the Field Navigation application additionally utilized functions that capture input slices. We defined two slices based on properties of the firing activity of single neurons. The *hypoactivity* and *hyperactivity* slice functions detect when the neural signal over an entire time bin is below or above certain thresholds, respectively.

5.4 Retraining and Evaluating Repair

While the Motor Decoding and Field Navigation neural decoders used neural networks and could be efficiently retrained by incrementally training on new data, the other three decoders used traditional ML models and required retraining with a concatenation of the initial and new training data. To acquire the new training data, we sampled 500 data points from the acquisition dataset.

To assess the effectiveness of our methodology for repairing BCIs, we compared both fault frequency and overall prediction performance of a repaired model to the baseline model. We specifically assess whether different data acquisition strategies lead to significant differences.

6 EVALUATION RESULTS

6.1 RQ1: Can partial test oracles precisely detect BCI faults?

Since faults are detected through partial test oracles, we computed precision as the percentage of correctly detected faults, based on ground truth, out of all faults detected by an oracle.

Precision of BCI Fault Detection via Partial Test Oracles According to Table 4, the oracle capturing inconsistency with the auxiliary model (Multimodal Inconsistency) has the greatest precision, followed by the oracles capturing illegal transitions, as they identify states of the decoder that are certainly incorrect. While the other oracles have lower precision, they can be fine-tuned to improve precision. A test oracle that detects temporal inconsistency, for example, can be hard to define, as one must fine-tune the window size of model executions to consider at a time to determine inconsistency. However, we observe in Section 6.5 that oracles with lower precision are still valuable in improving decoder performance.

Prevalence of BCI Faults According to Table 4, the most prevalent faults detected by partial test oracles across all the applications are illegal transition and temporal inconsistency. This is not surprising because neural decoders process temporal data that varies in very specific ways across time. Furthermore, the prevalence of all faults indicates that there is potential for improving the performance of neural decoders by eliminating faults.

6.2 RQ2: Can corrective heuristics accurately transform faulty model outputs?

We evaluated the efficacy of each corrective heuristics, based on ground truth labels. Specifically, we compared model accuracy before and after applying the corrective heuristics to the baseline neural decoder predictions. Note that the performance for the cursor control and field navigation applications is measured using mean squared error, for which lower is better. The baseline is the faulty neural decoding model that was initially trained for regular BCI usage. Figure 3 gives the results and shows that 4 out of 5 BCI applications demonstrated a significant improvement in accuracy after applying the corrective heuristics. The accuracy of the motor decoding model decreased, but the difference was not significant.

6.3 RQ3: Can BCI faults be localized to input and output slices?

In order to test whether faults can be localized to input and output slices, we compared the distributions of input and output slices,
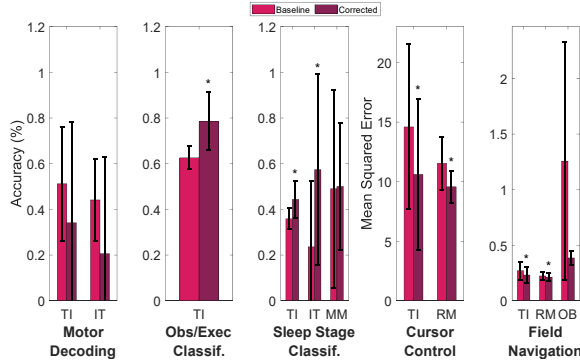
Figure 3: Efficacy of Corrective Heuristics. An asterisk (*) indicates a significant difference (paired two-sample t-test, p-value<0.05). Note that for mean squared error, lower is better.

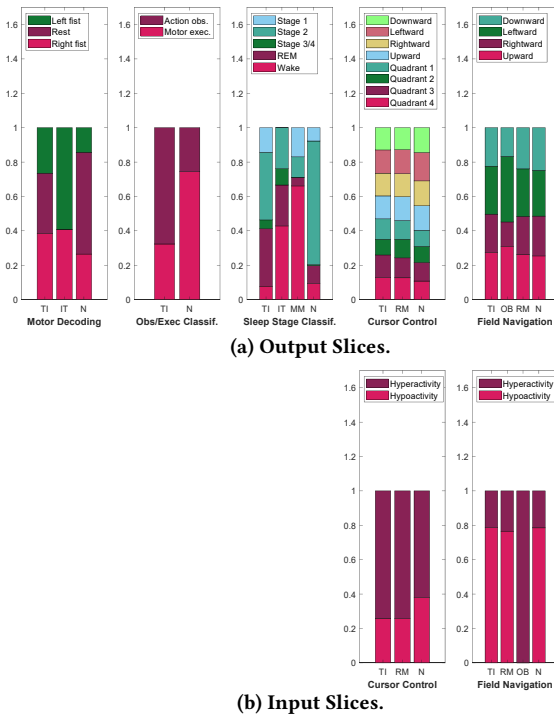**(a) Output Slices.**

(b) Input Slices.

Figure 4: Localizing faults to specific output slices (top) and input slices (bottom). Different faults (on the x-axis, with N referring to no fault) exhibit different distributions of co-occurring slices. All distributions are normalized to 1, but slices may co-occur.

conditioned on whether a particular type of fault occurred. We used a Chi-squared test of independence on the relationship between the presence of a given type of fault and the distribution of input and output slices, respectively.

Figure 4 shows that most types of faults exhibit distinct distributions of input and/or output slices for—both compared to the distributions of other faults and no fault in a given application. For example, temporal inconsistency faults in the motor decoding application coincide with instances of the resting task, whereas illegal transition faults almost never occur during resting (Chi-squared test of independence, p<0.05). In this case, the illegal transition fault

can be localized to a task distribution primarily comprised of the non-resting tasks. The same observation holds for the observe-or-execute classification, with only one type of fault. The distribution of tasks being simultaneously performed during fault occurrence is significantly different than the no-fault distribution (Chi-squared test of independence, p<0.05): when temporal inconsistency faults occur, a BCI user is more likely to be observing an action. In this case, temporal inconsistency faults can be localized to a distribution skewed towards the observation task.

Overall, our results show that for most types of faults in the five BCI applications, there is a statistically significant association between fault occurrence and slice distribution.

6.4 RQ4: Can fault-based data acquisition and model retraining repair BCI faults?

To evaluate our proposed methodology for fault-based data acquisition and repair, we trained the underlying neural decoder on data acquired based on the computed fault-based task distributions, for all five BCI applications. We measured repair success for each type of fault in isolation as well as for all types of faults combined. Specifically, we computed the change in the number of fault occurrences after training. Although individual types of faults can co-occur, each fault represents a unique issue. Therefore, when measuring repair success for all types of faults combined, we sum the fault occurrences of each individual type of fault (as opposed to counting unique faulty model outputs).

Table 5 gives the overall results and shows that retraining the underlying neural decoding model after fault-based data acquisition can reduce the number of faults. For example, for the observe-or-execute classification application, training on data acquired through a task distribution yields a statistically significant improvement over the baseline and over training on data acquired through the natural task distribution.

For other applications, we did not observe such a statistical significant reduction, and we conjecture three reasons. First, the acquisition dataset is limited, being only one-tenth of the original dataset. Since we acquired data by selecting a fixed number of datapoints from this already limited acquisition dataset, there is less room for the task distribution to show a significant effect, compared to the natural distribution. Second, since we use previously collected data sets, we cannot acquire new data based on the task distribution, but instead simulate data acquisition by sampling from a portion of the existing dataset that the model has never seen. Fault-based data acquisition in a real calibration session may be more targeted. Third, the corrective heuristics that we defined for this evaluation may not be accurate enough to produce accurate task distributions. We wish to explore these further in future work, and carrying out a user study may provide additional insights.

Fault-based data acquisition can achieve a reduction in faults. However, a secondary concern is that retraining must simultaneously ensure that accuracy (or related measure of performance such as mean-squared error) does not significantly degrade. We evaluate this by computing the change in accuracy after retraining on data acquired through fault localization. We found that there was a significant *increase* in accuracy for the motor decoding application, and no significant change for the other applications, suggesting that fault-based data acquisition can repair BCIs without degrading accuracy.

Table 5: Repairing BCIs with Fault-Based Data Acquisition. For each application, we seek to eliminate either a single fault or all faults combined. We compare the number of faults (averaged over 10 trials) before retraining (Baseline), after retraining with data acquired based on the natural distribution of tasks (After Bernoulli), and after retraining with data acquired based on a fault-based distribution of tasks (After Fault Localization w/ Corrections). We also compare to simply applying corrective heuristics (After Corrections) and fault-based data acquisition without applying corrective heuristics first (After Fault Localization w/o Corrections). Comparison to baseline: $^{***}p < 0.01$, $^{**}p < 0.05$, $^{*}p < 0.1$. Comparison to Bernoulli-sampled task distribution: $^{+++}p < 0.01$, $^{++}p < 0.05$, $^{+}p < 0.1$

BCI Application	Fault	Baseline	After Bernoulli	After Fault Localization w/ Corrections	After Corrections	After Fault Localization w/o Corrections
EEG Motor Decoding	Temporal Inconsistency	230.75	241.96	244.79 **	-	240.17 **
	Illegal Transition	78.29	76.38	73.96	-	207.79 ***+++
	All faults combined	309.04	318.33	308.13	266.25 ***+++	305.88
Observation-or-Execution Classification	Temporal Inconsistency	62.56	61.44	46.22 *****++	-	53.78
	All faults combined	62.56	61.44	46.22 *****	71.67 *	53.78
Sleep Stage Classification	Temporal Inconsistency	153.00	147.04	151.67 +++	-	153.83 +
	Illegal Transition	31.92	27.50	44.50 ***+++	-	24.38 ***++
	Multimodal Inconsistency	23.79	19.79	25.83 ++	-	51.75 ***+++
	All faults combined	208.71	194.33	199.88 *	221.17 ***++	199.25 **
BCI Cursor Control	Temporal Inconsistency	100.00	63.00	62.00 *	-	62.00
	Rapid Motion	349.83	296.50	292.50	-	293.17
	All faults combined	449.83	359.50	355.67 *	525.50 +++	358.50 *
Field Navigation	Temporal Inconsistency	1220.20	1200.40	1178.40 **	-	1161.40 ***+++
	Rapid Motion	2002.00	2048.20	2040.60	-	2001.40
	Out of Bounds	93.60	77.40	69.80	-	84.80 +
	All faults combined	3315.80	3326.00	3274.60	3305.20	3282.60

6.5 RQ5: Are corrective heuristics or fault localization sufficient on their own?

Section 6.2 established the effectiveness of fault localization based on heuristically corrected model outputs, used to compute a task distribution for data acquisition and model retraining. This method requires both (1) corrective heuristics to correct faulty predictions and (2) fault localization to inform data acquisition. Here, we explore whether either of these components suffices. In other words, we test whether either training on heuristically corrected predictions without a fault-based distribution or localizing faults to tasks without correcting faulty predictions would confer performance improvements. The two right-most columns in Table 5 show the results for these two alternative approaches.

First, we discuss the results of directly retraining the neural decoder on heuristically corrected faults, without assuming a fault-based task distribution. As shown in Table 5, we observe that training on heuristically corrected faults decreased the number of faults for the motor decoding application. However, we also observed a significant increase in the number of faults for 3 out of 5 applications. We attribute this to two reasons. First, the amount of the data the decoder model is trained on is dependent on the number of detected faults. In the applications that we tested, fault frequency is overall low, and training on this small number of specific data points likely causes the model to overfit. Second, since corrections are based on heuristics, training on them could bias the model, especially if there are few corrected datapoints to retrain on. We conclude that while corrective heuristics are a useful tool for correcting individual predictions, training on the corrected data points is insufficient for repairing the underlying decoder.

Next, we discuss the effectiveness of fault-based data acquisition without the use of corrective heuristics. In Section 6.4, corrective heuristics were used as a means for weakly correcting model predictions in order to accurately determine the task being performed. However, correcting model predictions with imprecise heuristics may not result in a more accurate task distribution. To test whether effective fault-localization could be achieved without corrective heuristics, we computed the number of faults when retraining on fault-based acquired data but without applying corrective heuristics to model predictions first. We found that removing corrective heuristics significantly decreased the number of faults only for illegal-transition faults in the sleep stage classification application. For all other faults and applications, the results of this approach are comparable to or significantly worse than the results of fault-based data acquisition with corrective heuristics applied. We conclude that corrective heuristics are an important component in the methodology to compute an accurate fault-based task distribution.

7 THREATS TO VALIDITY

External Validity We selected five BCI applications for evaluating the applicability of our proposed methodology for repairing BCIs post-deployment. The comprehensiveness and representativeness of this selection affects external validity, but we aimed for a diverse set of applications by considering the different dimensions for classifying BCIs (Section 3). Additionally, our methodology assumes that neural decoder model predictions can be mapped to a sufficiently large number of tasks for fine-grained fault localization. We argue that this assumption is satisfied in practice: BCIs such as neuroprostheses indeed perform a variety of tasks. For example, hand posture can be classified into more than 40 types, and since

BCIs are an emerging technology, advances will likely results in an even greater number of supported postures.

Construct Validity   Our evaluation used existing, labeled datasets to simulate real-world BCI usage. These datasets were collected in a rigid and non-naturalistic experiment settings. For example, in the EEG motor decoding application, participants opened and closed their fists on cue, but during real-world usage these actions would be naturalistic. As a result, our efficacy measures of partial oracles, corrective heuristics, and slice functions could be specific to the non-naturalistic setting. A related concern is that our measures for fault occurrence and frequency may be specific to the experimental setting and are not accurate measures for faults that will occur during real-world usage. Likewise, faults that *do* appear in an experimental setting may not occur during real-world usage. However, many of these faults such as temporal inconsistency do occur in other ML-based control systems [17]. Thus, while most BCI research has reported on overall model performance (as opposed to reporting on fault occurrences), it is reasonable to expect that these faults do occur in BCIs as well.

Internal Validity   A threat to internal validity is model overfitting, which could prevent our evaluation from properly isolating the effect of retraining based on a given task distribution. Recall that our evaluation used existing data sets and retrained models on small, fixed sets of newly acquired data. There is a risk of overfitting to the initial training data and/or to the newly acquired data, which could worsen model performance. This, however, would likely underestimate the efficacy of fault-based data acquisition because real-world data acquisition would not be subject to the same restrictions.

8 RELATED WORK

There are three avenues of closely related work: (1) methods for improving the performance of BCIs, (2) applications of software testing research to machine learning systems, and (3) miscellaneous work combining BCIs and software engineering.

Optimizing BCI Performance   Prior work has sought to improve the data collection methodology for BCIs through emerging technologies or novel sampling techniques. For example, virtual reality has been applied to more realistically simulate the surrounding environment of a BCI user [4]. Active class selection (ACS) has also been previously explored as a means for more efficient data acquisition for BCIs. A combination of entropy and the inverse of class accuracy has been applied to motor imagery for choosing classes that are informative and in need of improved accuracy [14]. Data acquisition through ACS has been combined with transfer learning in an application to a BCI [36]. By contrast, instead of the general goal of increasing accuracy of neural decoding predictions, our stated goal is to repair specific faults as reported by users. Additionally instead of applying a sophisticated sampling algorithm, we opt for a more software-testing-oriented and human-in-the-loop approach with partial test oracles defined by domain experts.

Software Testing and ML Systems   The problems encountered when testing BCIs are similar to those encountered when testing other data-driven ML systems. For example, prior work has shown that neural network accuracy can be improved through fault localization by localizing errors to specific neurons in the network [10] and that fairness and accuracy can be improved by localizing errors

to influential data points [18, 32]. Machine learning has also been applied to improve debugging techniques for traditional software systems (e.g., [2, 19, 23]).

Software Engineering and BCIs   Other work has sought to apply software engineering to BCIs. Application areas have included writing, comprehension, and debugging of code [4, 9, 26]. The application of software engineering research to BCIs has been proposed with a particular emphasis on self-adaptive systems reacting to inputs where those inputs are now neural signals [15].

9 CONCLUSION

This paper explores the application of software testing and debugging methods to the emerging field of brain-computer interfaces (BCIs). Specifically, it proposes a human-in-the-loop methodology for testing and repairing BCIs post-deployment. The evaluation based on five BCI applications shows that (1) partial test oracles can indeed detect faults in BCIs, (2) corrective heuristics and slice functions can localize faults to specific data slices in both the input and output spaces, and (3) fault-based data acquisition can reduce the fault frequency while maintaining model accuracy. Overall, the results suggest that the proposed methodology is a promising step towards more accurate and safer BCIs.

REFERENCES

[1] Brendan Z Allison, Elizabeth Winter Wolpaw, and Jonathan R Wolpaw. 2007. Brain–computer interface systems: progress and prospects. *Expert review of medical devices* 4, 4 (2007), 463–474.

[2] Luciano C Ascari, Lucilia Y Araki, Aurora RT Pozo, and Silvia R Vergilio. 2009. Exploring machine learning techniques for fault localization. In *2009 10th Latin American Test Workshop*. IEEE, 1–6.

[3] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. 2015. The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering* 41, 5 (2015), 507–525. https://doi.org/10.1109/tse.2014.2372785

[4] Tarik Boukhalfi, Christian Joyal, Stéphane Bouchard, Sarah Michelle Neveu, and Patrice Renaud. 2015. Tools and techniques for real-time data acquisition and analysis in brain computer interface studies using qEEG and eye tracking in virtual reality environment. *IFAC-PapersOnLine* 48, 3 (2015), 46–51.

[5] R. Chaisaen, P. Autthasan, N. Mingchinda, P. Leelaarporn, N. Kunaseth, S. Tammajarung, P. Manoonpong, S. C. Mukhopadhyay, and T. Wilaiprasitporn. 2020. Decoding EEG Rhythms During Action Observation, Motor Imagery, and Execution for Standing and Sitting. *IEEE Sensors Journal* 20, 22 (2020), 13776–13786. https://doi.org/10.1109/JSEN.2020.3005968

[6] Rattanaphon Chaisaen, Phairot Autthasan, Nopparada Mingchinda, Pitshaporn Leelaarporn, Narin Kunaseth, Suppakorn Tammajarung, Poramate Manoonpong, Subhas Chandra Mukhopadhyay, and Theerawit Wilaiprasitporn. 2020. Decoding eeg rhythms during action observation, motor imagery, and execution for standing and sitting. *IEEE Sensors Journal* 20, 22 (2020), 13776–13786. https://doi.org/10.1109/jsen.2020.3005968

[7] Stanislas Chambon, Mathieu N. Galtier, Pierrick J. Arnal, Gilles Wainrib, and Alexandre Gramfort. 2018. A Deep Learning Architecture for Temporal Sleep Stage Classification Using Multivariate and Multimodal Time Series. *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 26, 4 (2018), 758–769. https://doi.org/10.1109/TNSRE.2018.2813138

[8] Inchul Choi, Ilsun Rhiu, Yushin Lee, Myung Hwan Yun, and Chang S. Nam. 2017. A systematic review of hybrid brain-computer interfaces: Taxonomy and usability perspectives. *PLOS ONE* 12, 4 (2017). https://doi.org/10.1371/journal.pone.0176674

[9] João Duraes, Henrique Madeira, João Castelhano, Catarina Duarte, and M Castelo Branco. 2016. WAP: understanding the brain at software debugging. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 87–92.

[10] Hasan Ferit Eniser, Simos Gerasimou, and Alper Sen. 2019. Deepfault: Fault localization for deep neural networks. *arXiv preprint arXiv:1902.05974* (2019).

[11] Joshua I Glaser, Ari S Benjamin, Raeed H Chowdhury, Matthew G Perich, Lee E Miller, and Konrad P Kording. 2020. Machine learning for neural decoding. *Eneuro* 7, 4 (2020).

[12] Ary L. Goldberger, Luis A. Amaral, Leon Glass, Jeffrey M. Hausdorff, Plamen Ch. Ivanov, Roger G. Mark, Joseph E. Mietus, George B. Moody, Chung-Kang Peng,

H. Eugene Stanley, and et al. 2000. PhysioBank, PHYSIOTOOLKIT, and PhysioNet. *Circulation* 101, 23 (2000). https://doi.org/10.1161/01.cir.101.23.e215

[13] Alexandre Gramfort, Martin Luessi, Eric Larson, Denis A. Engemann, Daniel Strohmeier, Christian Brodbeck, Roman Goj, Mainak Jas, Teon Brooks, Lauri Parkkonen, and Matti S. Hämäläinen. 2013. MEG and EEG Data Analysis with MNE-Python. *Frontiers in Neuroscience* 7, 267 (2013), 1–13. https://doi.org/10.3389/fnins.2013.00267

[14] Ibrahim Hossain, Abbas Khosravi, and Saeid Nahavandi. 2017. Weighted informative inverse active class selection for motor imagery brain computer interface. In *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*. IEEE, 1–5.

[15] Shihong Huang and Emmanuelle Tognoli. 2014. Brainware: Synergizing software systems and neural inputs. In *Companion Proceedings of the 36th International Conference on Software Engineering*. 444–447.

[16] René Just and Franz Schweiggert. 2011. Automating unit and integration testing with partial oracles. *Software Quality Journal* 19, 4 (2011), 753–769.

[17] Daniel Kang, Deepti Raghavan, Peter Bailis, and Matei Zaharia. 2018. Model assertions for debugging machine learning. In *NeurIPS MLSys Workshop*.

[18] Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning (ICML)*. 1885–1894.

[19] Xia Li, Wei Li, Yuqun Zhang, and Lingming Zhang. 2019. Deepfl: Integrating multiple fault diagnosis dimensions for deep fault localization. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 169–180.

[20] S.G. Mason and G.E. Birch. 2003. A general framework for brain-computer interface design. *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 11, 1, 70–85. https://doi.org/10.1109/TNSRE.2003.810426

[21] Felip Miralles, Eloisa Vargiu, Stefan Dauwalder, Marc Solà, Gernot Müller-Putz, Selina C Wriessnegger, Andreas Pinegger, Andrea Kübler, Sebastian Halder, Ivo Käthner, et al. 2015. Brain computer interface on track to home. *The Scientific World Journal* 2015 (2015).

[22] Jelena Mladenovic, Jérémie Mattout, and Fabien Lotte. 2017. A generic framework for adaptive EEG-based BCI training and operation. *CoRR* abs/1707.07935 (2017). arXiv:1707.07935 http://arxiv.org/abs/1707.07935

[23] Aniruddh Nath and Pedro Domingos. 2016. Learning tractable probabilistic models for fault localization. In *Thirtieth AAAI Conference on Artificial Intelligence*.

[24] Patrick Ofner and Gernot R. Müller-Putz. 2012. Decoding of velocities and positions of 3D arm movement from EEG. In *2012 Annual International Conference*

of the IEEE Engineering in Medicine and Biology Society. 6406–6409. https://doi.org/10.1109/EMBC.2012.6347460

[25] Spencer Pearson, José Campos, René Just, Gordon Fraser, Rui Abreu, Michael D Ernst, Deric Pang, and Benjamin Keller. 2017. Evaluating and improving fault localization. In *International Conference on Software Engineering (ICSE)*. 609–620.

[26] Norman Peitek, Janet Siegmund, Sven Apel, Christian Kästner, Chris Parnin, Anja Bethmann, Thomas Leich, Gunter Saake, and André Brechmann. 2018. A look into programmers' heads. *IEEE Transactions on Software Engineering* 46, 4 (2018), 442–462.

[27] Rajesh P.N. Rao. 2013. *Brain-computer interfacing: An introduction.* Cambridge University Press, New York.

[28] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2020. Snorkel: Rapid training data creation with weak supervision. *The VLDB Journal* 29, 2 (2020), 709–730.

[29] G. Schalk, D.J. McFarland, T. Hinterberger, N. Birbaumer, and J.R. Wolpaw. 2004. Bci2000: A general-purpose brain-computer interface (bci) system. *IEEE Transactions on Biomedical Engineering* 51, 6 (2004), 1034–1043. https://doi.org/10.1109/tbme.2004.827072

[30] Sergio Segura, Gordon Fraser, Ana B Sanchez, and Antonio Ruiz-Cortés. 2016. A survey on metamorphic testing. *IEEE Transactions on software engineering* 42, 9 (2016), 805–824.

[31] Steffen Steinert, Christoph Bublitz, Ralf Jox, and Orsolya Friedrich. 2019. Doing things with thoughts: Brain-computer interfaces and disembodied agency. *Philosophy & Technology* 32, 3 (2019), 457–482.

[32] Sahil Verma, Michael Ernst, and Rene Just. 2021. Removing biased data to improve fairness and accuracy. *arXiv preprint arXiv:2102.03054* (2021).

[33] Mark Weiser. 1984. Program slicing. *IEEE Transactions on software engineering* 4 (1984), 352–357.

[34] Elaine J Weyuker. 1982. On testing non-testable programs. *Comput. J.* 25, 4 (1982), 465–470.

[35] W Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. 2016. A survey on software fault localization. *IEEE Transactions on Software Engineering* 42, 8 (2016), 707–740.

[36] Dongrui Wu, Brent J Lance, and Thomas D Parsons. 2013. Collaborative filtering for brain-computer interaction using transfer learning and active class selection. *PLoS one* 8, 2 (2013), e56624.

[37] Xiang Zhang, Lina Yao, Xianzhi Wang, Jessica JM Monaghan, David Mcalpine, and Yu Zhang. 2020. A survey on deep learning-based non-invasive brain signals: recent advances and new frontiers. *Journal of Neural Engineering* (2020).