PyDecNef: An open-source framework for fMRI-based decoded neurofeedback

Pedro Margolles [1] Ning Mei, Patxi Elosegi, and David Soto [1,2]

[1] Basque Center on Cognition, Brain and Language, San Sebastian, Spain

[2] Ikerbasque, Basque Foundation for Science, Bilbao, Spain

Correspondence should be addressed to: pedromargolles@gmail.com, d.soto@bcbl.eu, Basque Center on Cognition, Brain and Language, Paseo Mikeletegi 69, 2nd Floor 20009 San Sebastian.

Abstract

Real time fMRI research has suffered from inaccessible analysis pipelines, hindering collaboration and reproducibility. Here we present PyDecNef, a Python-based platform designed to advance real-time fMRI analysis and fuel exploration of close-loop neuroimaging for cognitive neuroscience studies. Creating a real-time fMRI analysis pipeline from scratch poses formidable technical challenges, involving data transfer, experimental software, and machine learning classifier preparation. Existing tools like FRIEND, Brain-Voyant, and OpenNFT demand expensive licenses or rely on proprietary software, impeding accessibility and customizability. PyDecNef offers a solution: a transparent, versatile, and open workflow for real-time fMRI decoding protocols. This open-source platform simplifies decoder construction, real-time preprocessing, decoding, and feedback signal generation. It empowers researchers to launch DecNef experiments efficiently, saving time and resources. Moreover, its openness promotes collaboration, enhancing research quality, replicability, and impact. With PyDecNef, the path to advancing DecNef studies becomes more accessible and collaborative. PyDecNef resources for real-time fMRI analysis can be found at https://pedromargolles.github.io/pyDecNef/,. Here we also provide experimental data illustrating that PyDecNef provides more fine-grained and less binomial/overconfident neurofeedback signals compared to previous DecNef approaches that have relied on sparse multinomial logistic regression classifiers, and hence, potentially helping participants to learn better how to self-regulate their brain activity.

Introduction

One of the main challenges for the development of decoded neurofeedback (DecNef) studies is setting up a real-time fMRI processing framework. Until now, DecNef studies employed analysis pipelines that were not openly available to the scientific community. We introduce PyDecNef, a Python-based, flexible, fully open platform for real time fMRI analysis that we make here openly available to the scientific community to advance research in this area, foster collaborative efforts, reproducibility, and promote the emergence of new research lines in DecNef studies.

Building from scratch a fMRI real-time analysis pipeline entails a big technical challenge for any researcher that would like to enter the field of DecNef. Multiple components are involved, namely: to manage the data transfer between the MRI scanner, the real time analysis PC, and the experimental software that controls the stimuli, task and neurofeedback signals given to the participant. A key step in this process involves the preparation of the machine learning classifier for real time application. Here it is critical to enhance the precision and generalizability of the classifier in the real-time fMRI context. When we started to develop PyDecNef,, there were a few software packages to perform decoded neurofeedback studies, namely, the FRIEND toolbox (Sato et al., 2018), Brain-Voyant (Cohen, Malach, Koppel, & Friedman, 2018), Turbo-BrainVoyager (Goebel, 2012), BrainIAK (Kumar et al., 2019) and OpenNFT (Koush et al., 2017) However, some of these packages are commercial requiring the acquisition of an expensive use license or written with the proprietary and clouse-source MATLAB software, while others lack customizability or clear documentation.

We believe that a truly open, simple but flexible and customizable workflow for real time fMRI analysis and decoded neurofeedback would be extremely valuable for any research group thinking about starting to run their own DecNef experiments, saving them time, money and also a few headaches along the way. Moreover, an open pipeline for DecNef would facilitate collaboration across many laboratories, enhancing the quality, the replicability and the impact of DecNef research. With these motivations in mind, we decided to develop and release our open-source fMRI-based DecNef pipeline.

PyDecNef is available at https://github.com/pedromargolles/pyDecNef. It provides an open and simple framework for performing real-time fMRI decoded neurofeedback studies using Python 3.6. The library includes documented scripts for decoder construction and also for real time fMRI preprocessing, online decoding, feedback signaling etc. Overall, the pyDecNef framework provides the researcher with a number of benefits over other real-time fMRI analysis software, namely:

- *Customizable*: pyDecNef code frontend was written in Python 3.6. Python stands out for being an open, powerful, flexible, user-friendly and efficient high-level language. This feature in combination with the modular and object-oriented programming approach used

for the development of the pipeline allows for easy customization of DecNef paradigms without the need of deep coding skills.

- *Adaptable*: PyDecNef can be easily adapted for the execution of other type of neurofeedback studies that involve self-regulation of brain activity levels within a target region of interest (e.g. Scharnowski et al., 2012) or to further develop other closed-loop fMRI protocols, for instance applying Bayesian Optimization to delineate the key stimulus parameter space that is most suitable to maximize a relevant neural pattern in the brain (Lorenz et al., 2016).
- *Integral*: Our framework provides a complete approach for the development of DecNef experiments, including tools for processing and decoding fMRI data in real-time and raw example data and resources for different decoder construction, DecNef approaches (i..e. static vs dynamic), off-line analyses, and experimental software for stimulus and feedback presentation.
- *Simple*: pyDecNef framework is easy to use and understand, whether you are a neuroimage novice or an expert. All scripts are well documented.
- *Fast*: While Python is known to be a highly-interpretable and easy to use programming language, it does not stand out for its processing speed. Computational efficiency is an essential requirement in real-time MRI settings, specially in DecNef paradigms where feedback given to participants need to be as close as possible to target the neural pattern of interest. By means of the Nipype library (Gorgolewski et al.. 2011), pyDecNef incorporates functionalities from computationally efficient neuroimage software written in C language (i.e. Afni). This makes pyDecNef both flexible and fast.
- *Fully open-access*: Last but not least pyDecNef is an open-access, collaborative and free to use, which will improve henceforth owing to the efforts of the scientific. Technical support and consulting is also freely given to any researcher intending to run a DecNef experiment using our pipeline.

Installation of the library

To set up the necessary infrastructure, it is needed an Ethernet Local Area Network (LAN) connecting three computers:

Server Computer: The server computer is responsible for running real-time volume preprocessing and decoding scripts.

MRI Scanner Host Computer: The pyDecNef framework is compatible with various types of MRI scanners. However, for the smooth integration with FIRMM software, necessary for transferring volumes from the MRI scanner host computer to the real time analysis computer (i.e. the server), it

is recommended to use Siemens or GE scanner models. The host computer is used for managing scanning sessions and transferring the data to the server computer.

The MRI scanner host computer should be configured to copy functional DICOM files to a designated folder on the server computer in real-time. The server computer runs real-time processing scripts, including a watcher class responsible for monitoring the folder and initiating volume preprocessing as soon as a new volume is written to it. To achieve this, Siemens Trio and Prisma scanner host computers can use the "ideacmdtool" program in conjunction with the FIRMM software, while GE scanners can utilize the rsync protocol.

Using ideacmdtool + FIRMM with SIEMENS scanners
Using rsync + FIRMM with GE scanners
FIRMM installation

Client Computer: This computer runs the Python-based software for presenting stimuli and feedback to the participant. Example scripts are included to run in Opensesame (Mathôt, Schreij & Theeuwes, 2012)

Opensesame installation

Figure 1 below illustrates the set-up.

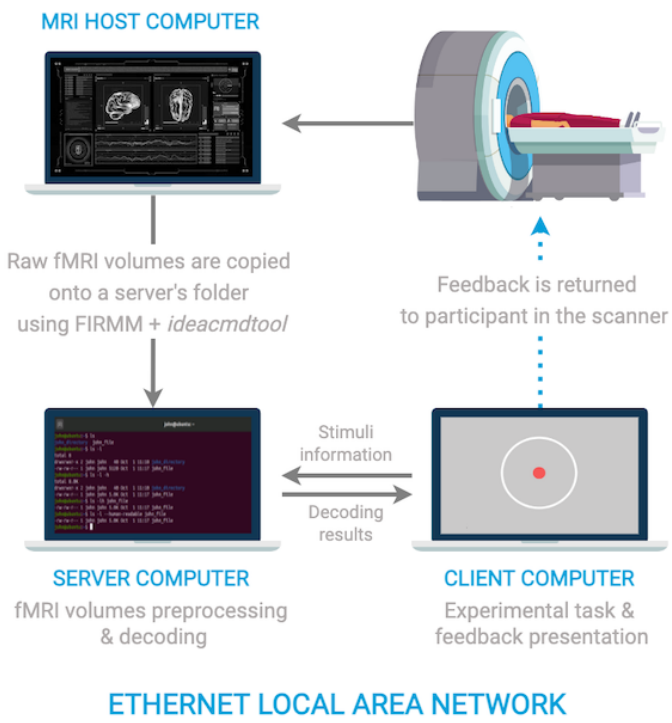

Figure 1. *Illustration of the setup of the three computers needed to perform real-time fMRI.*

System requirements for the server computer

For the server's role in collecting and processing fMRI volumes in real-time, it is essential to have a computer with exceptional performance and ample storage capacity. The server computer must be capable of preprocessing and decoding volumes in less time than the selected fMRI repetition time (TR). Ideally, the server computer should run a Linux system such as Debian/Ubuntu 16+ or Redhat/CentOS 7+ to enable the use of either Docker or Singularity for the FIRMM software and either the Samba or rsync protocols for DICOM transfer.

Successful implementation of pyDecNef real-time scripts has been demonstrated on a computer running the CentOS Linux 7 operating system with the following specifications:

Memory: 31 GB
Processor: Intel© Core™ i9-9900K CPU @ 3.60GHz x 16
Graphics: Intel© HD Graphics (Coffeelake 3x8 GT2)
GNOME: Version 3.28.2
OS type: 64-bit
Disk: 2.0TB

FIRMM: To receive fMRI volumes from the MRI scanner host computer, it is necessary to install FIRMM version 2.1 or a newer version on the server computer. This installation facilitates data transfer using either the Samba or rsync methods, depending on whether the MRI scanner is Siemens or GE, respectively.

Neuroimage Analysis Software: To enhance the preprocessing speed of fMRI volumes and leverage the versatility offered by the Python language, pyDecNef incorporates various specialized neuroimage analysis software written in C. These software tools are seamlessly integrated into a unified Python workflow through the use of the Nipype library.

AFNI installation
FSL installation
dcm2niix installation

Python tools: The pyDecNef scripts are designed to run on Python 3.6 or any newer version, minimizing the reliance on external libraries and maximizing compatibility across different Python versions. It is essential to have Python installed on both the server and client computers.

Python Dependencies: The pipeline was tested in both Linux and Mac using Anaconda 2020.11, and through the creation of an environment as follows

```
conda create -n rtfmri
conda activate rtfmri
conda install python=3.7
conda install -c conda-forge dcm2niix # did not install with pip
pip install nilearn sockets colorama dash GPyOpt nipype tk scikit-learn==0.22.2.post1
```

We also installed the tensorflow library for the decoder. This requires the latest pip (pip install --upgrade pip) and then:

```
pip install tensorflow==2.8.
```

Libraries for data preprocessing and decoding

To minimize processing times, our pyDecNef Python framework leverages the Nipype interface module to connect Python with neuroimage software written in computationally efficient programming languages such as C, including dcm2nii, and AFNI. Further, to prevent potential overlap during processing of fMRI volumes, pyDecNef includes control measures that allow for independent processing of volumes in separate threads running in parallel. The preprocessing framework was designed to maximize the similarity of the data preprocessing for the decoder construction and the real-time processing during DecNef.

- Nipype
- Nilearn
- scikit-learn
- Tensorflow

Libraries for visualization of results

- Colorama
- Matplotlib
- seaborn
- plotly
- Dash

PyDecNef

The repository of PyDecNef scripts can be accessed through this link. Below we describe their functionality separately for each repository folder.

Real-time fMRI Scripts

These can be found here:
https://github.com/pedromargolles/pyDecNef/tree/main/2.decnef_training/server/scripts/rt_decnef

This section includes all the Python scripts necessary for the server computer to preprocess and decode volumes during neurofeedback training sessions in real-time..

The main.py script integrates the "config," "classes," and "pipelines" modules together and it is the main script that has to be run from the server, followed by the stimulus presentation software, in order to run a real time fMRI experiment. .

Config Module:

exp_config.py: The primary experimental configuration file where you specify details such as the number of MRI scanner heat-up volumes, baseline volumes, hemodynamic response function peak limits, fMRI TRs, types of z-scoring procedures, decoding procedures, and essential files for neurofeedback decoding (e.g., regions of interest, machine learning models, reference volumes, z-scoring values).

connection_config.py: Facilitates the connection between scripts running on the server and client computers and enables data sharing between them.

listener.py: Matches specific client requests (e.g., from the experimental software) with custom server actions.

shared_instances.py: Enables the sharing of global variables and class objects instantiated in "main.py" across all modules.

Classes Module:

classes.py: Defines attributes and methods for Python classes that interact to store information and process volumes in real-time. These include independent volumes, volume time series, independent trials, file watchers, client listeners, and data loggers.

Pipelines Module:

corregistration_pipeline.py: This basic pipeline utilizes dcm2niix, AFNI, nipype, and nilearn to perform real-time corregistration of volumes to a reference volume from the model construction session. It also masks the volume using a region of interest mask file.

preproc_vol_to_timeseries_pipeline.py: Contains three different pipelines to detrend and z-score each neurofeedback training session volume using time series information from that fMRI session run.

trial_decoding_pipeline.py: Contains three different pipelines to decode neurofeedback training trials and volumes using a previously trained machine learning model.

These scripts are intended to run in parallel with the **Python-based experimental software** (i.e. Opensesame) that controls stimuli, task and feedback presentation. These scripts can be found here:

https://github.com/pedromargolles/pyDecNef/tree/main/2.decnef_training/client

Note the connection_config.py file is needed to set the connection between server and client PCs. The scripts contain examples of static and dynamic neurofeedback paradigms built within the OpenSesame software (Mathôt, Schreij & Theeuwes, 2012). The static paradigm is the standard approach used in decoded neurofeedback involving the so-called "induction" runs in which the participant is required to self-regulate her brain activity in order to maximize the size of a feedback disk on each trial. Unbeknown to participants the size of the feedback is proportional to the probability of decoding the target state in the brain activity patterns. The dynamic paradigm involves the continuous decoding of every scan and hence could be used to "trigger" the presentation of a stimulus based on the decoded brain activity pattern (Debettencourt, Cohen, Lee, Norman. & Turk-Browne, 2015).. These two types of paradigms can be adapted for your specific experimental purposes.

fMRI Simulator (3.fMRI_simulator_realdata)

https://github.com/pedromargolles/pyDecNef/tree/main/2.decnef_training/server/scripts/mri_simulator

generate.py: Simulates an fMRI scanner working in real-time. This tool is used for offline testing of your decoded neurofeedback experimental paradigm, for instance, to ensure accurate synchronization between scripts in the server and client PCs, namely to simulate real time DecNef preprocessing and decoding (using raw volumes from a participant from the decoder construction session) alongside the stimulus and feedback presentation.

We also include a script for visualization of the probabilities of decoding the target class across trials of the DecNef training, which can be found here.

https://github.com/pedromargolles/pyDecNef/tree/main/2.decnef_training/server/scripts/rt_visualization

Model Construction

Decoding neurofeedback experiments typically commence with this session. During an experimental session, data is gathered to train a multivariate machine learning classifier, which is

subsequently applied in real-time neurofeedback training sessions with participants. It is crucial to note that decoding neurofeedback experiments require high-performance classifiers (i..e. with a decoding performance above 70% ROC-AUC) that are capable of discriminating the relevant brain representation. Model construction paradigms should be designed with this criterion in mind.

PyDecNef includes an example model construction paradigm in OpenSesame, including raw data and data processing scripts. Scripts can be found here:

https://github.com/pedromargolles/pyDecNef/tree/main/1.model_construction

Simply replace the provided data with your own, adjust the scripts to match your experimental settings, and you are ready to proceed.

fMRI Scan Preprocessing

Preprocessing Pipeline: The scans used for decoder construction and decoded neurofeedback are preprocessed using the same pipeline to minimize potential sources of variability and enhance the real time classifier performance. As a starting point, a functional scan is designated as the reference volume for both model construction and neurofeedback training sessions. All functional volumes from the model construction session are individually co-registered to the reference volume. Each volume is labeled with information pertaining to the run, onset time, stimuli category, etc. After labeling, volumes undergo linear detrending within each fMRI run. Subsequently, volumes of interest falling within the HRF peak of each trial are selected for model construction. These are then subjected to voxel-level Z-score normalization.

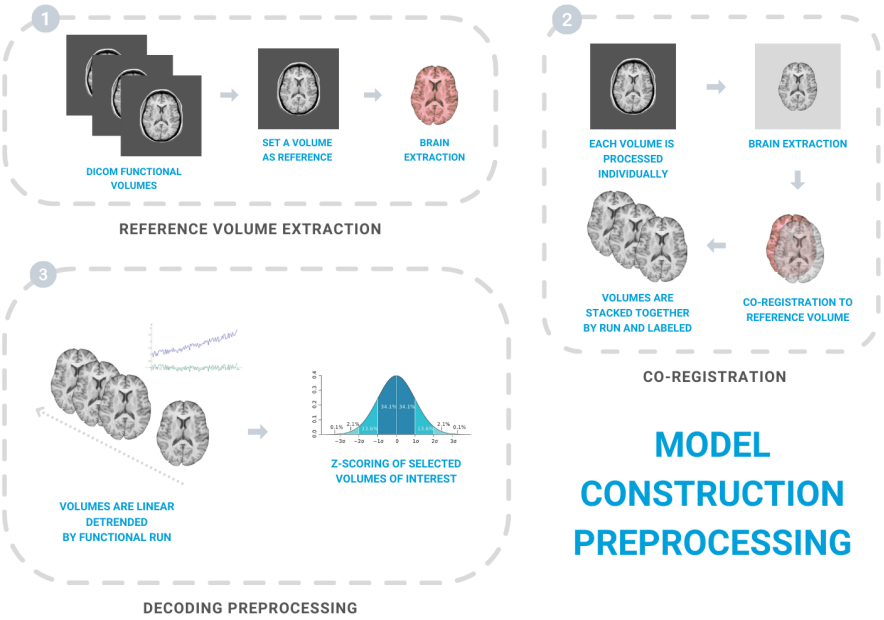

Figure 2. *Illustration of the online preprocessing approach.*

Preprocessing scripts for decoder construction can be found at: https://github.com/pedromargolles/pyDecNef/tree/main/1.model_construction/sub-1/scripts organized by their execution order. PyDecNef also includes example raw data in DICOM format following the data structure specified in the preprocessing scripts. Replacing this data with your own and modifying the scripts to match your experimental settings will prepare your pipeline for use. Below we describe each of the steps.

1. Reference Volume Extraction: The first preprocessing step involves selecting a raw functional volume to serve as the reference image for all other volumes in the model construction session, as well as for neurofeedback training sessions. Typically, fMRI runs begin with a few volumes (5-10 volumes) for image stabilization, which are then discarded. In this pipeline, the first raw volume following the MRI scanner's heat-up phase is selected as the reference image. This DICOM volume is converted to NIfTI format (without compression) using dcm2niix. It is also skull-stripped using AFNI to facilitate co-registration with subsequent volumes.

2. Volumes Co-Registration: Raw volumes are converted to NIfTI format, skull-stripped, and co-registered to the reference volume using a rigid body (6-parameter) transformations via AFNI's 3dvolreg program. A 7th-order polynomial interpolation method is applied as the default spatial interpolation method during co-registration. To improve co-registration accuracy between scans, one can manipulate the "erode" or "clfrac" parameters in the brain extraction preprocessing step. Adjusting these parameters erodes the brain mask inwards to exclude skull and tissue fragments,

improving co-registration accuracy. Note that if you change these parameters, they should also be updated within the co-registration pipeline function used during real-time neurofeedback training for that subject (i.e., "corregistration_pipeline.py").

3. Decoding Preprocessing: After labeling and co-registration the volumes, we proceed with training the machine learning classifier.

Comparison of different classification pipelines for DecNef training

Previous studies on DecNef (Watanabe et al., 2017) employed classification models that often generated excessively binary predictions when applied to out-of-sample data. Consequently, these models hindered the utilization of nuanced neurofeedback signals to effectively facilitate participants' learning in self-regulating their brain activity patterns. To resolve this issue, PyDecNef implemented the following strategies: (i) we developed a custom regression model in Tensorflow using Binary Cross-Entropy Loss to provide a more fine-grained decoding of the target class while accounting for noisy representations using soft-labeling (ii) we standardized the fMRI scans during DecNef based on the data used for classifier training (iii) we aimed to reduced the binomiality of the predictions by averaging decoding probabilities from volumes within the HRF peak of each trial.

This comparison of classification pipelines was performed based on the data from the DecNef study by Margolles, Elosegi, Mei and Soto (2023). Data for decoder construction involved four perceptual runs and two mental imagery runs. The perceptual runs showed pictures of Dogs and Scissors. Following four perceptual runs, participants completed two mental imagery runs in which they were asked to vividly imagine one example picture from each of the visual categories.

We assessed how well our custom logistic regression classifier with BCE Loss performed compared to the standard and sparse logistic regression classifiers commonly used in DecNef research (Miyawaki et al., 2008; Yamashita et al., 2008). We evaluated the AUC and prediction probability distributions of the different decoders under three scenarios across participants: perception data, cross-domain generalization from perception to mental imagery data, and cross-domain generalization from perception to DecNef data.

We used the perception data to implement a leave-one-run-out cross-validation approach to estimate performance of the classifier. Here the decoder was iteratively trained using data from 3 perceptual runs, and then tested on the remaining run. For the standard logistic regression classifier, the maximum number of iterations was set to 5000. The L2 regularization penalty was applied to the logistic regression coefficients to prevent overfitting. The inverse of the regularization strength, known as C, was set to 1 to balance the model complexity and fit to the data. Lastly, the lbfgs solver was used to optimize the logistic regression objective function, since it is well-suited for small to medium-sized datasets. For the sparse logistic regression classifier (Yamashita et al., 2008), we used the Sparse Multinomial Logistic Regression (SMLR) Python implementation (Majima,

2015) available through this link: https://github.com/KamitaniLab/smlr. For comparability purposes we used default parameters. To be specific, we set a maximum number of iterations of 1000 for decoder training and used 1e-5 as the tolerance value of the stopping criteria. Visual noise trials were not used neither for training-testing standard nor sparse logistic regression classifiers. Finally, to maintain consistency with our experimental approach, we trained our custom logistic regression classifier with BCE Loss using the same parameters as mentioned in the Decoder training section. During the training process, the data was labeled with visual noise samples assigned to 0.5 during the training process. Training the decoder with Tensorflow required a validation set and accordingly we used a cross-validation approach in which the left-out run was divided into two halves to create a validation set and a test set. To obtain a more comprehensive assessment of the decoder performance, we repeated this cross-validation procedure by swapping the validation and test sets, ensuring that all data was used as a test set.

In the cross-domain generalization from perception to mental imagery data, the standard and sparse logistic regression classifiers were trained using all data from perception sessions (again, excluding visual noise trials) and then tested over the HRF peak volumes from the two mental imagery runs. The training data for our decoder with BCE loss consisted of volumes from 'Dogs', 'Scissors', and Visual noise trials from the first three perceptual runs. The validation set included 'Dogs' and 'Scissors' volumes from the 4th perceptual run, while the mental imagery data served as the test set.

Lastly, in cross-domain generalization from perception to DecNef data, the standard and sparse logistic regression classifiers were trained using 'Dogs' and 'Scissors' data from all perception runs and evaluated on all the HRF peak volumes within each DecNef training trial. To evaluate our custom logistic regression classifier, we utilized the pre-trained classifiers used in our experiment for each participant. We compared the different decoders by assessing the distribution of the predicted probabilities. Aggregated predicted probabilities for the three decoders across subjects can be visualized in Figure 3.

We used the Kolmogorov-Smirnov test to compare the distributions of predictions of the different decoders, which include standard vs. sparse logistic regression, standard vs. custom logistic regression, and sparse vs. custom logistic regression. In all scenarios, the decoder distributions were found to be significantly different from each other.

In the perception condition [Standard vs. Sparse - KS statistic: .056, $p < .001$; Standard vs. Custom - KS statistic: .149, $p < .001$; Sparse vs. Custom - KS statistic: .19, $p < .001$]; from perception to mental imagery [Standard vs. Sparse - KS statistic: .053, $p < .001$; Standard vs. Custom - KS statistic: .175, $p < .001$; Sparse vs. Custom - KS statistic: .161, $p < .001$]; and from perception to neurofeedback [Standard vs. Sparse - KS statistic: .059, $p < .001$; Standard vs. Custom - KS statistic: .156, $p < .001$; Sparse vs. Custom - KS statistic: .212, $p < .001$].

The probability distributions generated by the standard and sparse logistic regression classifiers exhibited overconfidence and greater similarity. These characteristics are particularly

noticeable when examining cross-domain generalization scenarios. While maintaining performance, our customized classifier helped to mitigate the overly confident predictions in all scenarios.

To further tackle this issue of the binomial predictions, we calculated the average predicted probability for each volume of interest within a DecNef trial (i.e., volumes captured between 5 and 11 seconds after the hiragana presentation), instead of using the more conventional approach of averaging all volumes prior to decoding. Figure 4 below compares the predicted probabilities for each DecNef trial obtained through both approaches for the standard, sparse and our custom logistic regression decoder with BCE Loss. The approach used here also seemed to help to moderate the overconfidence of the classifier, thereby providing participants with finer-grained and more informative feedback.

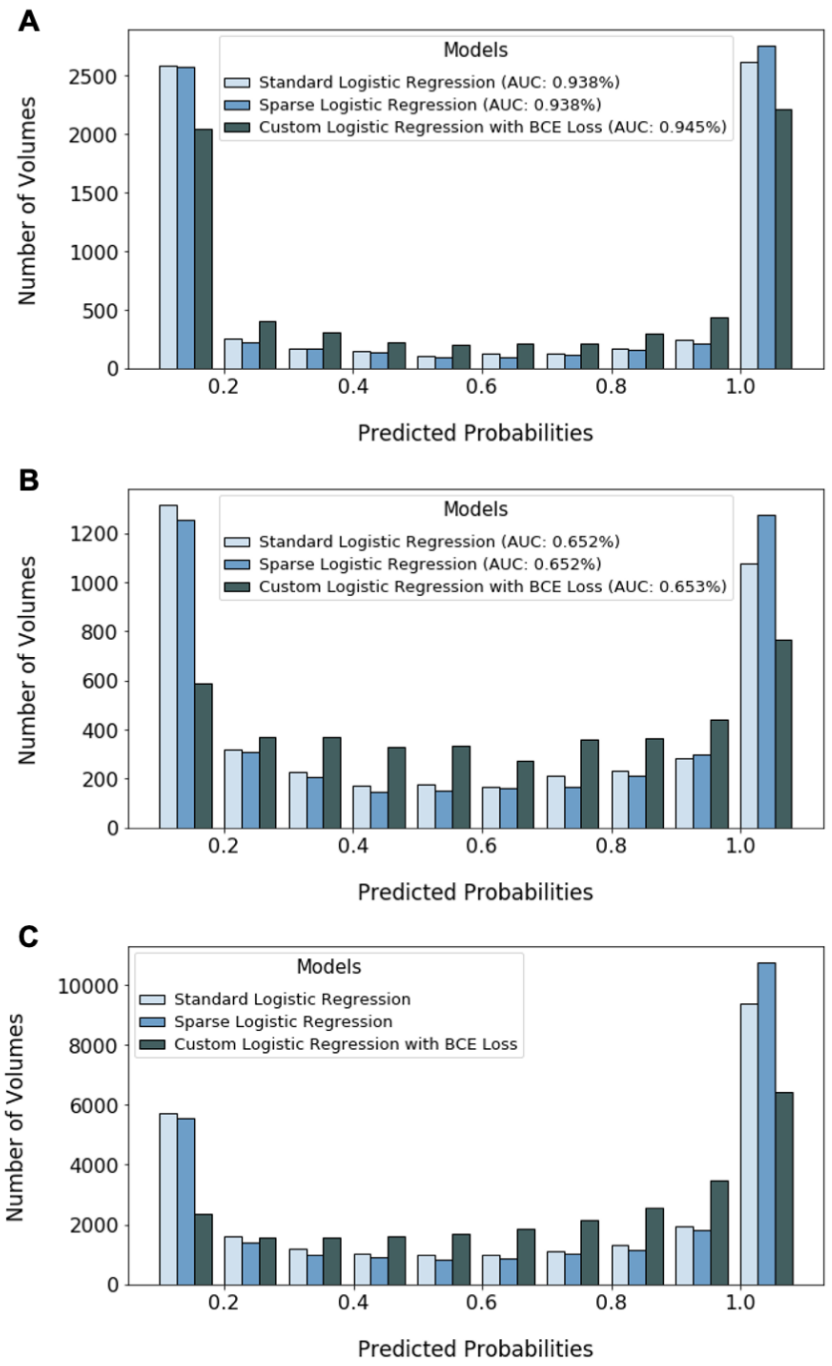

Figure 3. *Histogram illustrating the predicted probability distribution (i.e., the number of volumes across participants) for various decoders, including standard logistic regression, sparse logistic regression, and our custom logistic regression classifier with Binary Cross-Entropy (BCE) Loss. This distribution is shown across different classification problems. (A) Perception: aggregated probabilities across the four cross-validation folds. (B) Visual imagery: aggregated cross-domain generalization probabilities based on a classifier trained on perception. (C) Aggregated cross-domain generalization probabilities during DecNef based on the decoder trained on perception and validated on imagery.*

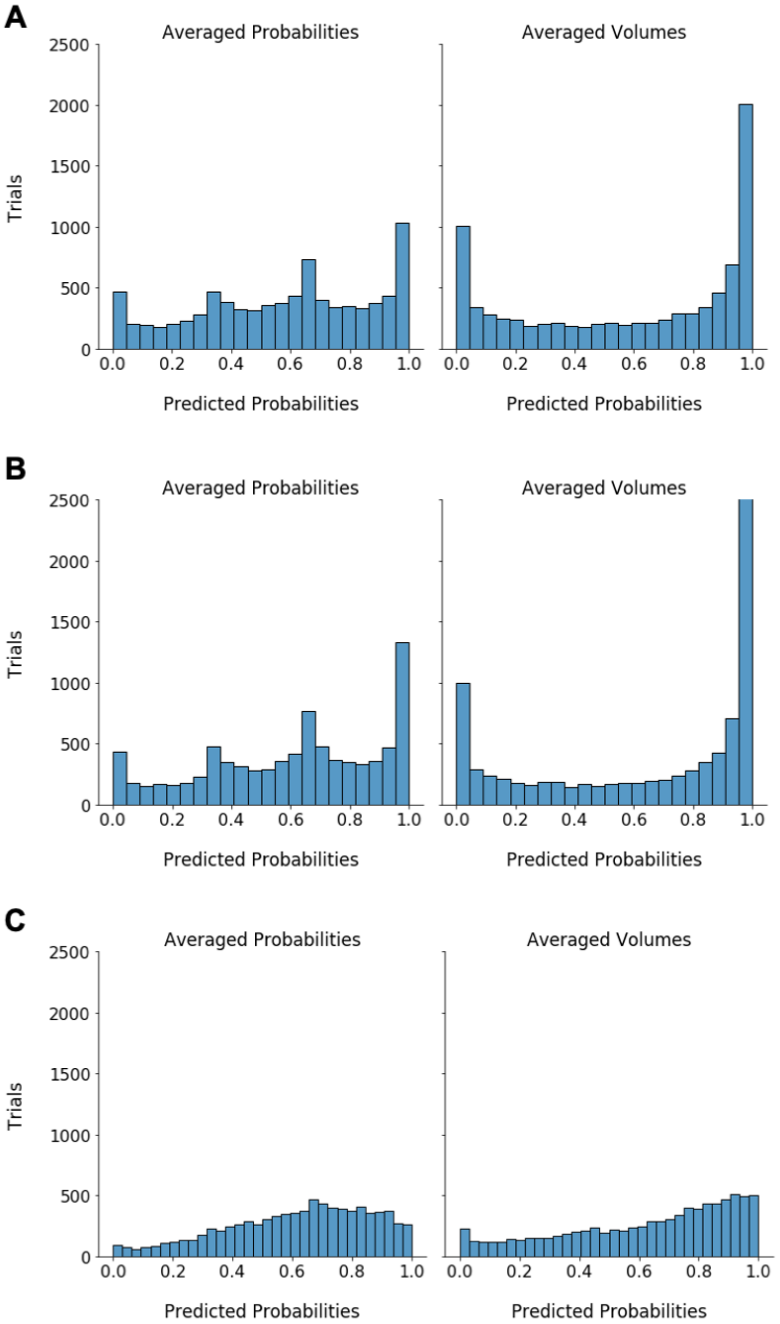

Figure 4. *Histogram illustrating the predicted probability distribution (i.e., the number of DecNef trials across participants) for different types of decoders and averaging methods. (A) Standard logistic regression. (B) Sparse logistic regression. (C) Our custom logistic regression classifier with BCE Loss. 'Averaged Probabilities' (on the left) displays the outcomes when the volumes within each DecNef trial are individually decoded and the resulting probabilities are then averaged. 'Averaged Volumes' (on the right) presents the results when the various volumes within a DecNef trial are first averaged, and then the probability of decoding the target class is computed.*

To assess the influence of several standardization methods on decoding performance during DecNef training, we conducted an analysis across participants using a similar approach to Oblak et al. (2019). This analysis leveraged the mental imagery data for decoder construction at our bilateral fusiform ROI (Margolles, Elosegi, Mei and Soto, 2023). Note that assessing the impact of normalization methods on the actual DecNef training data is difficult due to the absence of ground truth labels.

We evaluated five approaches: (1) no standardization, (2) standardization using the available run time series to each decoded volume, (3) standardization to the first 20 volumes of each mental imagery run, (4) standardization of decoded volumes based on the mean and standard deviation of the perceptual data, and (5) standardization using all the data within a run (as offline z-scoring method) (Figure 5A).

Even though our DecNef standardization approach, utilizing the mean and standard deviation of the training data for the decoder (i.e, perceptual data), demonstrated superior numerical decoding performance, in line with Oblak et al. (Oblak et al., 2019), a repeated measures ANOVA with Greenhouse-Geisser sphericity correction found no significant differences between the various z-scoring techniques [$F(1.833, 31.165) = 2.592$, $p = .095$, $\eta p 2 = .132$]. However, in contrast to their results, a paired t-test comparing the non-standardization and standardization of the data revealed no statistically significant differences [$t = -1.588$, $p = .065$] (Figure 5B).

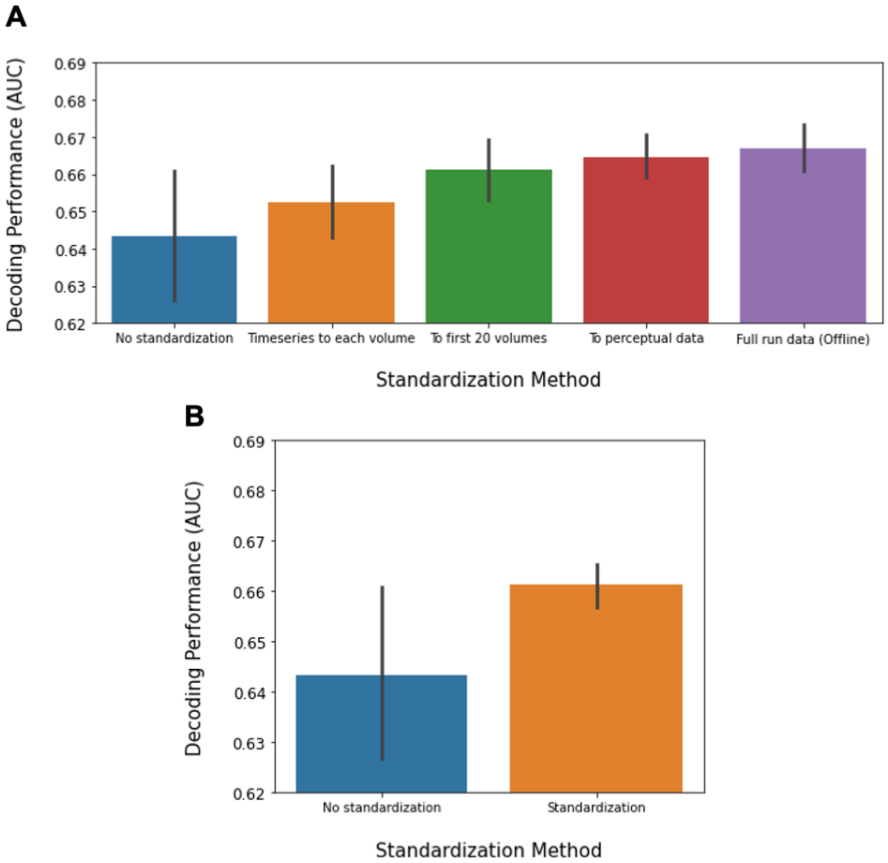

Figure 5. *(A) Assessing the impact of various z-scoring techniques on cross-domain generalization from perception to mental imagery within our bilateral fusiform ROI across participants. In*

sequential order: 1 No standardization, 2 — Standardization using the available run time series to each decoded volume, 3 — Standardization to the initial 20 volumes of mental imagery runs, 4 — Standardization based on mean and standard deviation of perceptual data, and 5 — Standardization using all data within a run (offline z-scoring method). (B) Comparison of decoding performance with and without standardization, where standardization represents the averaged performance of the tested methods. Error bars represent 95% CIs according to the Cousineau–Morey method (Cousineau et al., 2005; Cousineau & O'Brien, 2014).

Training participants using our customized classification pipeline for DecNef, we were able to observe reliable learning effects across DecNef training in our recent study (Margolles et al.. 2023) (see Figure 6). Participants progressively improved in their ability to induce brain activity patterns that were similar to the target of DecNef (i.e. the representation of Dog targets). This observation indicates that the PyDecNef pipeline is successful. The above observation is also newsworthy because previous DecNef studies using similar amount of training sessions our study (Koizumi et al., 2016, Cortese et al., 2016; Taschereau-Dumouchel et al., 2018, Amano et al., 2016) did not detect reliable learning effects in self-regulating brain activity towards the desired pattern, despite observing DecNef effects on behavioral performance (but see Shibata, Watanabe, Sasaki, and Kawato, 2011, who observed learning effects albeit with a much bigger amount of training sessions).

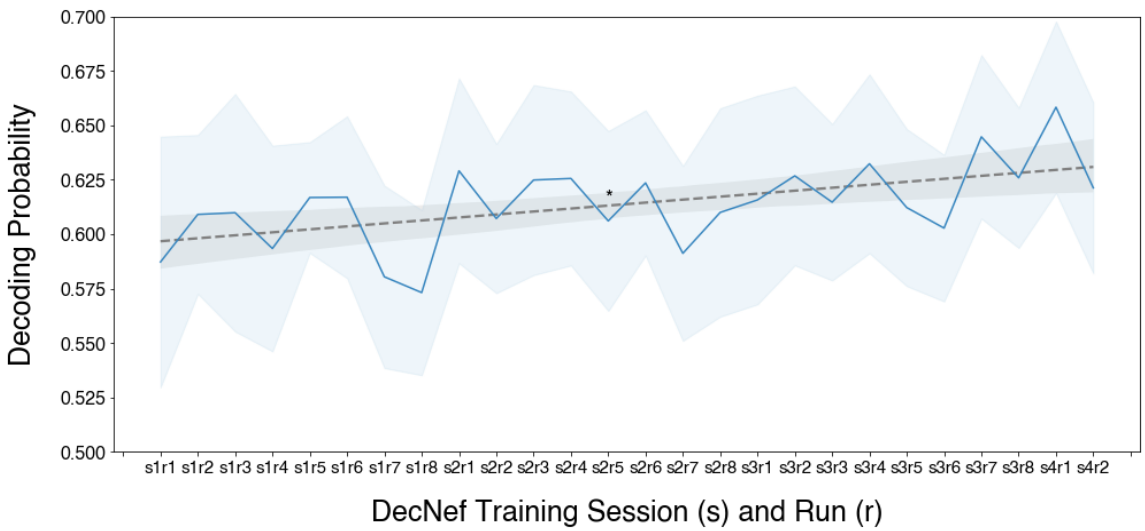

Figure 6. *Learning effects throughout DecNef training in Margolles et al.. (2023). The figure represents the average decoding probability of the target class (i.e. Dogs) in the bilateral fusiform cortex across participants, with a reliable learning trend. * p = .019. The error bars indicate the 95% confidence interval, which was adjusted using the Coiseneau-Morey method (Coiseneau & O'Brien, 2012).*

In summary, PyDecNef offers better calibrated and more fine-grained decoded neurofeedback signals, compared to the logistic regression models used in previous DecNef studies

that tended to give highly binomial classification probabilities as feedback (Bai et al., 2021). We believe that these improvements in the classification pipeline may promote participant's ability to identify more subtle variations in neural changes through feedback, thereby improving their ability for self-regulation during DecNef training. The developments presented here represent a step forward towards enhancing the precise targeting of neural representations by DecNef.

Open Science Statement

The scripts and documentation of the Python library for decoded neurofeedback (PyDecNef) can be found at: https://pedromargolles.github.io/pyDecNef/.

Acknowledgements

P.M acknowledges support from an FPI grant from the Spanish Ministry of Economy and Competitiveness. P.E. acknowledges support from the Basque Government PREDOC grant. D.S. acknowledges support from the Basque Government through the BERC 2022-2025 program, from the Spanish Ministry of Economy and Competitiveness, through the 'Severo Ochoa' Programme for Centres/Units of Excellence (CEX2020-001010-S) and also from project grant BrainMetaLoop PID2019-105494GB-I00. The funders had no role in study design, data collection and analysis, decision to publish or preparation of the manuscript.

References

Amano, K., Shibata, K., Kawato, M., Sasaki, Y., & Watanabe, T. (2016). Learning to associate orientation with color in early visual areas by associative decoded fMRI neurofeedback. *Current Biology, 26(14)*, 1861-1866.

Bai, Y., Mei, S., Wang, H., & Xiong, C. (2021). Don't just blame over-parametrization for over-confidence: Theoretical analysis of calibration in binary classification. In International conference on machine learning (pp. 566–576). PMLR

Cortese, A., Amano, K., Koizumi, A., Kawato, M., & Lau, H. (2016). Multivoxel neurofeedback selectively modulates confidence without changing perceptual performance. *Nature communications, 7(1)*, 13669.

Cohen, O., Malach, R., Koppel, M., & Friedman, D. (2018). Brain-voyant: A general purpose machine learning tool for real-time fmri whole-brain pattern classification. In 2018 international joint conference on neural networks (ijcnn) (pp. 1–8). IEEE

Debettencourt, M. T., Cohen, J. D., Lee, R. F., Norman, K. A., & Turk-Browne, N. B. (2015). Closed-loop training of attention with real-time brain imaging. *Nature Neuroscience, 18(3)*, 470-475.

Goebel, R. (2012). Brainvoyager—past, present, future. *Neuroimage, 62(2)*, 748–756.

Koush, Y., Ashburner, J., Prilepin, E., Sladky, R., Zeidman, P., Bibikov, S., . . . Van De Ville, D. (2017). Opennft: An open-source python/matlab framework for real-time fmri neurofeedback training based on activity, connectivity and multivariate pattern analysis. *NeuroImage, 156,* 489–503.

Koizumi, A., Amano, K., Cortese, A., Shibata, K., Yoshida, W., Seymour, B., ... & Lau, H. (2016). Fear reduction without fear through reinforcement of neural activity that bypasses conscious exposure. *Nature human behaviour, 1(1)*, 0006.

Kumar, M., Anderson, M. J., Antony, J. W., Baldassano, C., Brooks, P. P., Cai, M. B., . . . Huberdeau, D., et al. (2021). Brainiak: The brain imaging analysis kit. *Aperture neuro, 1(4)*

Lorenz, R., Monti, R. P., Violante, I. R., Anagnostopoulos, C., Faisal, A. A., Montana, G., & Leech, R. (2016). The Automatic Neuroscientist: A framework for optimizing experimental design with closed-loop real-time fMRI. *NeuroImage, 129*, 320-334.

Majima, K. (2015). Python implementation of sparse multinomial logistic regression (smlr). https://github.com/KamitaniLab/smlr. GitHub.

Margolles, P., Elosegi, P., Mei, N., & Soto, D. (2023). Unconscious manipulation of conceptual representations with decoded neurofeedback impacts search behaviour. bioRxiv, 2023-07.

Mathôt, S., Schreij, D., & Theeuwes, J. (2012). OpenSesame: An open-source, graphical experiment builder for the social sciences. *Behavior research methods, 44*, 314-324.

Miyawaki, Y., Uchida, H., Yamashita, O., Sato, M.-a., Morito, Y., Tanabe, H. C., . . . Kamitani, Y. (2008). Visual image reconstruction from human brain activity using a combination of multiscale local image decoders. *Neuron, 60(5)*, 915–929

Oblak, E. F., Sulzer, J. S., & Lewis-Peacock, J. A. (2019). A simulation-based approach to improve decoded neurofeedback performance. NeuroImage, 195, 300–310.

Sato, J. R., Basilio, R., Paiva, F. F., Garrido, G. J., Bramati, I. E., Bado, P., . . . Moll, J. (2013). Real-time fmri pattern decoding and neurofeedback using friend: An fsl-integrated bci toolbox. *PLoS One, 8(12)*, e81658

Scharnowski, F., Hutton, C., Josephs, O., Weiskopf, N., & Rees, G. (2012). Improving visual perception through neurofeedback. *Journal of Neuroscience, 32(49)*, 17830-17841.

Shibata, K., Watanabe, T., Sasaki, Y., & Kawato, M. (2011). Perceptual learning incepted by decoded fMRI neurofeedback without stimulus presentation. *Science, 334(6061)*, 1413-1415.

Taschereau-Dumouchel, V., Cortese, A., Chiba, T., Knotts, J. D., Kawato, M., & Lau, H. (2018). Towards an unconscious neural reinforcement intervention for common fears. *Proceedings of the National Academy of Sciences, 115(13)*, 3470-3475.

Watanabe, T., Sasaki, Y., Shibata, K., & Kawato, M. (2017). Advances in fmri real-time neurofeedback. *Trends in Cognitive Sciences, 21(12)*, 997–1010.

Yamashita, O., Sato, M.-a., Yoshioka, T., Tong, F., & Kamitani, Y. (2008). Sparse estimation automatically selects voxels relevant for the decoding of fmri activity patterns. *NeuroImage, 42(4),* 1414–1429.