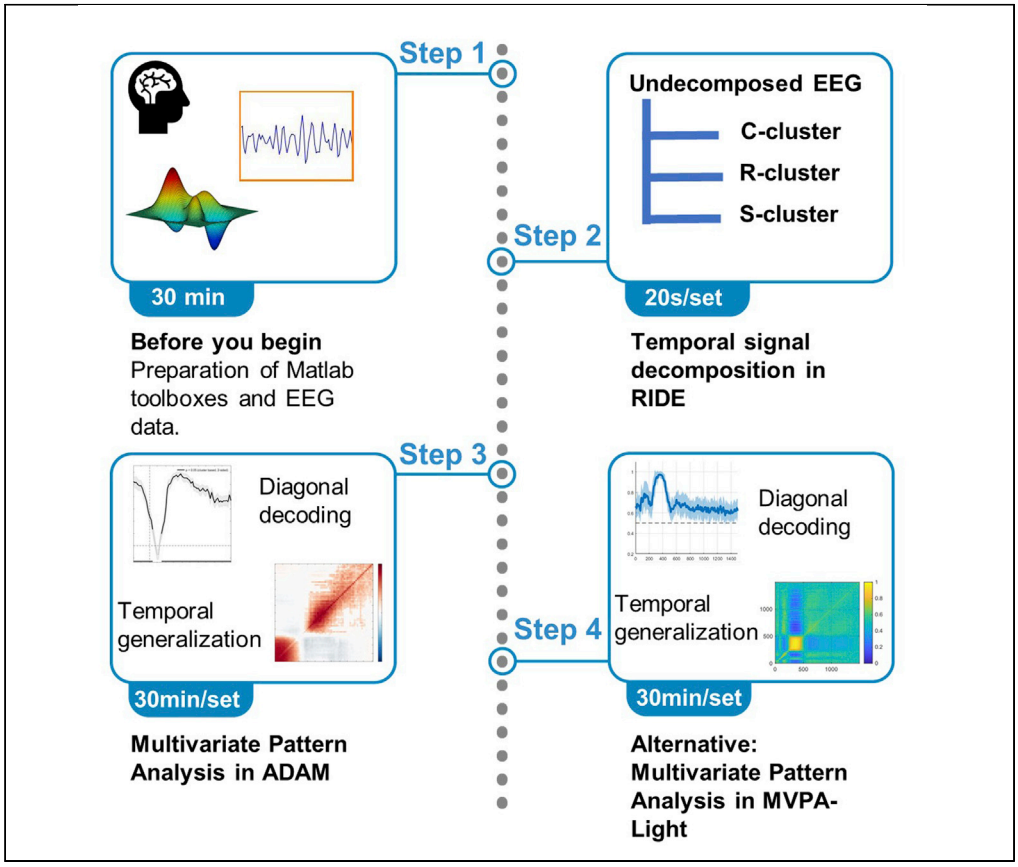
STAR Protocols

CellPress
OPEN ACCESS

Protocol

Protocol to decode representations from EEG data with intermixed signals using temporal signal decomposition and multivariate pattern-analysis



The electroencephalogram (EEG) is one of the most widely used techniques in cognitive neuroscience. We present a protocol showing how to combine a temporal signal decomposition approach (RIDE, Residue iteration decomposition) with multivariate pattern analysis (MVPA) to obtain insights into the temporal stability of representations coded in distinct informational fractions of the EEG signal. In this protocol, we describe pre-processing of human EEG data, followed by the set-up and use of MATLAB-based toolboxes for RIDE and MVPA analysis.

Ádám Takács, Shijing Yu, Moritz Mückschel, Christian Beste

adam.takacs@uniklinikum-dresden.de (Á.T.)
moritz.mueckschel@uniklinikum-dresden.de (M.M.)
christian.beste@uniklinikum-dresden.de (C.B.)

Highlights

A protocol for decoding temporally decomposed EEG signal

Steps for Residue iteration decomposition (RIDE) and handling the decomposed data

Steps for subsequent multivariate pattern analysis (MVPA) with different toolboxes

Recommendations for combined RIDE-MVPA research applications

Takács et al., STAR Protocols 3, 101399
June 17, 2022
https://doi.org/10.1016/j.xpro.2022.101399

STAR Protocols

CellPress
OPEN ACCESS

Protocol

Protocol to decode representations from EEG data with intermixed signals using temporal signal decomposition and multivariate pattern-analysis

Ádám Takács,[1,2,3,*] Shijing Yu,[1,2] Moritz Mückschel,[1,2,4,*] and Christian Beste[1,2,*]

[1]Cognitive Neurophysiology, Department of Child and Adolescent Psychiatry, Faculty of Medicine, TU Dresden, Schubertstrasse 42, 01309 Dresden, Germany

[2]University Neuropsychology Center, Faculty of Medicine, TU Dresden, Schubertstrasse 42, 01309 Dresden, Germany

[3]Technical contact

[4]Lead contact

*Correspondence: adam.takacs@uniklinikum-dresden.de (Á.T.), moritz.mueckschel@uniklinikum-dresden.de (M.M.), christian.beste@uniklinikum-dresden.de (C.B.)
https://doi.org/10.1016/j.xpro.2022.101399

SUMMARY

The electroencephalogram (EEG) is one of the most widely used techniques in cognitive neuroscience. We present a protocol showing how to combine a temporal signal decomposition approach (RIDE, Residue iteration decomposition) with multivariate pattern analysis (MVPA) to obtain insights into the temporal stability of representations coded in distinct informational fractions of the EEG signal. In this protocol, we describe pre-processing of human EEG data, followed by the set-up and use of MATLAB-based toolboxes for RIDE and MVPA analysis. For complete details on the use and execution of this protocol, please refer to Petruo et al. (2021).

BEFORE YOU BEGIN

The electroencephalogram (EEG) is one of the most widely used techniques in cognitive neuroscience. The EEG signal and event-related processes in this signal reflect the combination of various signals from different sources (Huster et al., 2015; Nunez et al., 1997; Stock et al., 2017). Likewise, distinct fractions of information are concomitantly coded in the EEG and important to consider, e.g., in the context of goal-directed behavior (Dippel et al., 2017; Folstein and Van Petten, 2008; Takacs et al., 2020b). This is also reasonable in light of overarching theoretical concepts of cognitive functions stating that perceptual processes, motor processes and the relationship between these are stored in a common representational format (Hommel, 2009). When taking such "common coding" concepts seriously, it is necessary to consider the existence of separable codes in the neurophysiological (EEG) signal (Takacs et al., 2020a, 2020b). This has been shown to be achievable using residue iteration decomposition (RIDE) (Ouyang et al., 2015a). However, likewise the concept of "representations" is central in contemporary cognitive neuroscience. Many theoretical frameworks deal with how representations are being formed, retrieved and operated with. Thus, it is also necessary to consider the representational content of such signals more closely. The dynamics of the representational content can be examined using multivariate pattern analysis (MVPA) (Carlson et al., 2011; King and Dehaene, 2014; Prochnow et al., 2021; Takacs et al., 2020a, 2021).

It is likely that any of this concomitantly coded information shows a distinct profile in how representations are activated and deactivated. Combining an EEG signal decomposition with MVPA is therefore useful to shift the focus from where in the brain a cognitive function can be observed to how several aspects of information are concomitantly represented and enable goal-directed behavior.

Any research relying on "common coding" principles may benefit from such a method because this method enables a fine-grained analysis of the different constituents of commonly coded information. The protocol presented is useful for all occasions where neurophysiological processes in the EEG are temporally locked to external events and where it is not possible to separate processes based on different spatial information (i.e., based on the scalp topography), using independent component analysis (ICA). ICA is only effective when there are independent source components that have different spatial weights and fluctuate independently in time (Ouyang et al., 2015a). Whenever the applicability of ICA is limited (e.g., because there are sources with highly correlated source activity but different scalp distributions, or sources with uncorrelated activity but highly similar scalp topography activity) (Groppe et al., 2009), the presented procedure is still usable.

We present a protocol combining RIDE and MVPA, as recently published by Petruo et al. (2021) and explain how RIDE can be combined with MVPA to obtain insights into the temporal stability of processes coded in distinct informational fractions of the EEG signal.

For MVPA analysis, we first present the ADAM MVPA toolbox (Fahrenfort, 2020; Fahrenfort et al., 2018), which allows performing multivariate analyses on EEG or MEG data using backward decoding (BDM) and forward encoding models (FEM).

As an alternative to ADAM, we also present the MVPA Light toolbox (Treder, 2020).

> ***Note:*** All these toolboxes require Matlab. The current protocol describes how to use temporally decomposed EEG data, as derived from RIDE, for classification and temporal generalization analysis. However, this protocol cannot substitute comprehensive manuals and tutorials dedicated to explaining the wide variety of signal decomposition and MVPA functions and parameters.

> ⚠ CRITICAL: The code presented here is shortened to the core functionality. The code does not take into account all conditions and does not loop through all participants. Further details of these methods are available for RIDE (Ouyang et al., 2015a, 2015b), ADAM MVPA toolbox (Fahrenfort, 2020; Fahrenfort et al., 2018), and MVPA Light toolbox (Treder, 2020).

Institutional oversight

The participants whose anonymized data have recently published by Petruo et al. (2021) were treated in accordance with the declaration of Helsinki and the study was approved by the ethics committee of the TU Dresden.

Data collection

The combination of temporal decomposition and MVPA can be desirable in many EEG experiments. Separately, both RIDE and MVPA have requirements on the input data, that can be used to determine the optimal sample size, the task design (e.g., trial number, trial length), number of sensors (i.e., EEG channels), and the necessary steps of pre-processing. Performing MVPA (classification and temporal generalization) on RIDE-decomposed data does not have additional requirements.

We present this protocol based on a study, in which EEG datasets from N=86 healthy young adults were used (Petruo et al., 2021). To facilitate a hands-on approach, we provide a subsample of N=20 participants (OSF | Example datasets for STAR Protocol). Participants completed a switching task that measured cognitive flexibility. In the task, a cued and a memory-based block followed each other, 198 trials in both. Before the experiment, participants were instructed to follow three task rules interchangeably. For further details, including specific instructions, task sets, and timing of the paradigm, see Petruo et al. (2021). Importantly, the application of the current protocol does not require more detailed information on the task or theoretical knowledge on cognitive flexibility.

Optional: For the interested reader, we recommend the method sections of previous papers that successfully combined RIDE and MVPA to study stimulus-response binding (Takacs et al., 2020a), response-response binding (Takacs et al., 2021) and inhibitory control (Prochnow et al., 2021).

Note: We use EEG data that was recorded in a dimly lit room with 60 Ag/AgCl electrodes, placed in equidistant positions in an elastic cap (EasyCap, Germany). BrainAmp amplifiers and the Recorder software (Brain Products, Germany) were used to collect data at 500 Hz sampling rate. The ground and reference electrodes were located on the following coordinates: $\theta = 58$, $\varphi = 78$ and $\theta = 90$, $\varphi = 90$, respectively. The current protocol can be used with any available research-grade EEG system.

Pre-processing of EEG data

Following data collection, we pre-process the EEG data in BrainVision Analyzer 2 software package (Brain Products, Germany). The pre-processing pipeline (1) follows previous studies that used the same task (Petruo et al., 2017, 2018; Wolff et al., 2016), (2) and ensured that requirements of the RIDE are met (see Preparation one: setting up RIDE).

Note: MVPA does not necessarily benefit from the applied pre-processing (van Driel et al., 2021). Furthermore, most pre-processing steps that are crucial for MVPA can be performed in ADAM (Fahrenfort et al., 2018), see also defining the configuration settings. However, given the lack of standards of pre-processing EEG data for MVPA (Takacs et al., 2020a) and the added value of the RIDE-MVPA combination, we recommend using pre-processed data. Specifically, we conduct filtering, ICA-decomposition for ocular and cardiovascular artifact correction, segmenting, baseline-correction, and residual artifact rejection before RIDE decomposition:

1. Down-sample EEG recordings to 256 Hz.
2. Filter the data with a band-pass filter of 0.5–20 Hz and a notch filter at 50 Hz (slope of 48 dB/oct).
3. Inspect the data manually to remove breaks and larger technical artifacts.
4. Use an ICA decomposition (infomax algorithm) to remove blinking, horizontal/vertical eye movements and cardiovascular artifacts.
5. Create segments locked to the target stimulus, separately for the cue-based and the memory-based blocks; for task repetition and task switching conditions; and for the different task sets.
   a. Include only correctly responded trials. Segments start -200 ms before target presentation and last 1,000 ms after that.

Note: Neither the current protocol nor the original study (Petruo et al., 2021) introduced results that are specific for task sets (i.e., subconditions). However, we recommend segmenting the EEG data to the smallest meaningful units for potential troubleshooting.

6. Run an automated artifact rejection to remove the remaining artifacts (criteria: signal amplitudes higher or lower than ± 200 μV; activity smaller than 0.5 μV for at least 200 ms; higher than 200 μV difference between two consecutive peaks within a 200 ms window).
7. Transform the segmented data to current source density (CSD, a spherical Laplace operator in which n = 4 splines and m = 10 Legendre polynomials; Lambda = 1×10^{-5}).
8. Baseline-correct the segments based on the activity preceding the target presentation (-200 to 0 ms).
9. Export the pre-processed EEG data (including all 60 channels) from BrainVision Analyzer as single-trial, single-subject time-locked data.

⚠ CRITICAL: The pre-processing steps were introduced here as an example. The choice of pre-processing should depend on the properties of the task, sample size, recording instruments, and the recording environment. Please, adjust the steps accordingly.

Preparation: Setting up toolboxes

Timing: 30 min

Follow these steps to install all required toolboxes in Matlab.

Note: Step 15 is optional.

10. Check requirements.
    a. A reasonably up-to-date PC running on Windows 10. Other operating systems may work but have not been tested.
    b. At least 8 GB of memory are recommended, more is better.
    c. A recent version of Matlab (The MathWorks, Inc.) not older than Matlab 2012b is required. Older versions may work but were not tested.
    d. Matlab toolboxes: 'Image processing toolbox", 'signal processing toolbox" and 'statistics toolbox".
    e. A recent version of EEGLAB not older than version 13.
    f. A recent version of FieldTrip not older than version 2015.
11. Install EEGLAB (Delorme and Makeig, 2004).
    a. Download the latest version of EEGLAB from https://eeglab.org/download/.
    b. Extract the archive to a folder of your choice.
    c. Install data import extension if necessary: https://eeglab.org/others/EEGLAB_Extensions.html#data-import.
12. Install FieldTrip (Oostenveld et al., 2010).
    a. Download the latest version of FieldTrip from https://www.fieldtriptoolbox.org/download/.
    b. Extract the archive to a folder of your choice.
13. Install RIDE (Ouyang et al., 2015b).
    a. Download the latest version of the RIDE toolbox from http://cns.hkbu.edu.hk/RIDE.htm.
14. Install ADAM toolbox (Fahrenfort et al., 2018).
    a. Download the latest version of ADAM from http://www.fahrenfort.com/ADAM.htm. Alternatively, you can access the files from https://github.com/fahrenfort/ADAM.
    b. Extract the archive to a directory of your choice.
    c. Open the 'startup.m' file in the 'install' subdirectory. The first lines of code should look like this:

```
% path definitions
ft_path = 'C:\TOOLBOXES\fieldtrip'; % Note Mac and Linux use forward slashes /
instead of \
eeglab_path = 'C:\TOOLBOXES\eeglab';
adam_path = 'C:\TOOLBOXES\ADAM-master';
```

    d. Replace the paths specified for the FieldTrip toolbox ('ft_path'), EEGLAB ('eeglab_path') and ADAM toolbox ('adam_path') with the actual paths on your computer.
    e. Save and close the 'startup.m' file.
15. Alternatives: install MVPA-Light toolbox (Treder, 2020).
    a. Download the MVPA-Light toolbox from: https://github.com/treder/MVPA-Light.

16. Set up the example code and data (optional).
    a. Download the example code and datasets from https://osf.io/4qgp2/.
    b. Extract the folders to a location of your choice.
    c. Matlab users may follow the protocol described below in a step-by-step fashion by running the provided codes.
    d. Use "Example code/star_protocol_RIDE.m" and "Example_data/RIDE" for step Temporal signal decomposition in RIDE.
    e. Use "Example code/star_protocol_concatenate.m" and "Example_data/Concatenate" for step Import the RIDE-decomposed data.
    f. Use "Example code/star_protocol_ADAM.m" and "Example_data/ADAM" for step First-level analysis in ADAM.
    g. Use "Example code/star_protocol_ADAM.m" and "Example_firstlevel_results/ADAM" for step Second-level analyses in ADAM.
    h. Use "Example code/star_protocol_MVPALight.m" and "Example_data/MVPALight" for step Alternative: MVPA in MVPA-Light.

KEY RESOURCES TABLE

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Deposited data		
Raw and analyzed data	(Petruo et al., 2021)	https://osf.io/xuqah/
Participant information (age and gender)	(Petruo et al., 2021)	https://osf.io/xuqah/
Example datasets and the codes	This protocol	Open Science Forum: OSF | Example datasets for STAR Protocol.
Software and algorithms		
Matlab 2019a	The MathWorks, Inc.	RRID: SCR_001622 https://de.mathworks.com/products/matlab.html
BrainVision Recorder	Brain Products	RRID: SCR_016331 https://www.brainproducts.com/productdetails.php?id=21
BrainVision Analyzer	Brain Products	RRID: SCR_002356 http://brainproducts.com/productdetails.php?id=17
EEGLAB	(Delorme and Makeig, 2004)	RRID: SCR_007292 https://eeglab.org/download
FieldTrip	(Oostenveld et al., 2010)	RRID: SCR_004849 https://www.fieldtriptoolbox.org/download
Residue Iteration Decomposition (RIDE)	(Ouyang et al., 2015b)	RRID: SCR_022174 http://cns.hkbu.edu.hk/RIDE.htm
Amsterdam Decoding and Modeling (ADAM)	(Fahrenfort et al., 2018)	RRID: SCR_022172 https://github.com/fahrenfort/ADAM
MVPA Light	(Treder, 2020)	RRID: SCR_022173 https://github.com/treder/MVPA-Light
Other		
BrainAmp	Brain Products	RRID: SCR_009443 https://www.brainproducts.com/productdetails.php?id=1

STEP-BY-STEP METHOD DETAILS

Temporal signal decomposition in RIDE

Timing: 20 s per dataset, depending on computing power

In this step, we apply the RIDE algorithm to EEG data. First, we import epoched, single-trial EEG data in Matlab using the EEGLAB toolbox and prepare the data for RIDE. Second, we extract the trial-specific reaction times (RT). Third, we set the configuration for the decomposition and run RIDE. Fourth, we extract single-trial decomposed data from the RIDE toolbox output. Finally, we

convert the single-trial data into a data format that can be used by the ADAM toolbox and MVPA-Light toolbox.

Note: If preferred, open "Example code/star_protocol_RIDE.m" in Matlab to follow this section while using the deposited data. Sample datasets for this step are in the folder "Example_-data/RIDE". This folder contains input data for RIDE.

1. Import EEG data.

RIDE requires epoched single-subject and single-trial data as input. Each dataset should contain trials of one condition and one subject only.

⚠ CRITICAL: Use datasets that are free of any artifacts and are baseline corrected. All channels containing non-brain activity must be discarded beforehand.

Note: The trials are stimulus-locked. The RIDE toolbox requires data in a 3-dimensional matrix, where the first dimension is sampling points, the second is channels, the third is trials (sample x channel x trial).

```
% Reset Matlab search path
restoredefaultpath
% clear workspace
clearvars

% Initialize EEGLAB
cd('PATH/TO/EEGLAB');
eeglab nogui;

% Add RIDE toolbox
addpath('PATH/TO/RIDE_call');

% Exemplary data folder
path = 'PATH/TO/Example_data/RIDE';

% Load example dataset in folder path
[EEG] = pop_loadbv(path,'001_Cue_NUM_rep.vhdr');

% Permute data matrix if the dimensions do not follow the order sample x channel x
trial
data = permute(EEG.data,[2,1,3]);
```

a. Import data in BrainVision Core Data Format 1.0 (exported from Brain Vision Analyzer using EEGLAB and EEGLAB bva-io plugin.
b. Permute the data matrix into the required input format sample x channel x trial as described above.

2. Extract reaction times.

Note: To separate the R component related signal, a vector containing the trial-specific reaction times (RTs) in milliseconds must be passed to RIDE. The length of this vector

must be equal to the size of the third dimension (i.e., trial dimension) of the EEG data matrix.

a. Extract the RTs from the event information variable available in EEGLAB after importing the EEG data.

⚠ CRITICAL: In EEGLAB, the locking point in each trial is coded by "TLE" in the 'EEG.event.type' field. The script looks for the first occurrence of a response-type event representing the response (i.e., "S1" or "S2" for left or right keypress). The trial-specific RTs are computed using the number of data points between the specific TLE event and the response event.

```
%% Extract RT

% Get all time point 0 (TLE) marker in EEG header file
trial_t0 = find(ismember({EEG.event.type},'TLE'));

% Initialize rt variable to number of stimulus time point 0 events in
% the event table.
rt = zeros(numel(trial_t0),1);

% Loop every trial
for trl = 1:numel(trial_t0)
    % Find first occurence of S1 or S2 event following the TLE event
    rt_pos = find(ismember({EEG.event(trial_t0(trl)+1:end).type},{'S  1','S
2'}),1);
    % Compute difference in sample points between TLE and response event.
    % Convert to ms using the sampling rate specified by EEG.srate
    rt(trl) = (EEG.event(trial_t0(trl)+rt_pos).latency -
EEG.event(trial_t0(trl)).latency) * 1000/EEG.srate;
end
```

3. Configure and run RIDE.

All parameters used for RIDE are passed in a struct variable 'cfg'. See the essential parameters below. For a comprehensive list, please refer to the RIDE manual available on the RIDE website (http://cns.hkbu.edu.hk/RIDE.htm).

a. Configure the parameters.

i. cfg.comp.name: A cell variable specifying the components to be extracted. Decompose the signal into three components 'S', 'C' and 'R'. 'S' refers to ERP component clusters that are locked to the stimulus onset and 'R' refers to component clusters locked to the RT. 'C' refers to a central component cluster not locked to either stimulus or response. For details, please refer to Ouyang et al. (2015b).

ii. cfg.comp.twd: Ouyang et al. (2015b) uses time window functions to optimize the decomposition of the components. For each component, a time window in ms must be specified that constrains the decomposition to the time window where the specific component is supposed to occur. The latencies of the typical event-related potentials as determined by visual inspection of the data provide a good orientation here. The S-component time window should cover ERPs associated with perceptual and attentional processes. The C-component time window should span response selection-associated potentials. The

time window of the R-component covers processes directly associated with the response and is specified relative to the trial-specific RT.

Note: Specify the time windows as used by Petruo et al. (2021).

iii. cfg.comp.latency: The RIDE decomposition aims to estimate the latency of the C-component. Therefore, the C-component is specified as 'unknown' here. The latencies of the S-component are assumed to be 0, the latencies of the R component are given by the RTs.
iv. cfg.samp_interval: The temporal resolution of the data in ms, which is the temporal difference between two consecutive data points.
v. cfg.epoch_twd: The time window of each trial in ms, which is the time of the first and last sample point in each trial.

b. Create the RIDE configuration by calling the function 'RIDE_cfg' and passing the 'cfg' struct variable.
c. Start the decomposition by calling the function 'RIDE_call' and passing the 'data' matrix variable and the 'cfg' configuration variable.

```
%% Set RIDE parameters
cfg = [];

% Component names
cfg.comp.name = {'s','c','r'};

% Time windows of components, [min max] in ms
cfg.comp.twd =  {[0,600],[200,900],[-300,300]};

% Latency information
cfg.comp.latency = {0,'unknown',rt};

% Single value – temporal resolution of the data, in ms.
% E.g. 3.9063 ms at 256 Hz
cfg.samp_interval = 1000/EEG.srate;

% 1*2 vector – trial time window, in ms
% E.g. [-100,1000];
cfg.epoch_twd = [EEG.xmin*1000, EEG.xmax*1000];

% Create RIDE configuration
cfg = RIDE_cfg(cfg);

% Run RIDE
results = RIDE_call(data,cfg);
```

4. Extract single-trial, decomposed component data.

The output variable of RIDE_call ('results') contains, among others, the fields 's', 'c', 'r'. These represent the decomposed, latency-corrected and averaged S-, C- and R-components.

⚠ CRITICAL: For later MVPA analysis, we need data on the single-trial level, which is not part of the default RIDE output. The single-trial data can be extracted with the RIDE toolbox

function 'move3'. This function shifts every single trial 3-D data (sample x channel x trial) by a relative lag.

a. Subtract the move3 transformed C and R component data from the undecomposed single-trial data (to yield the S cluster data). The output is single-trial, decomposed data.

Note: In contrast to the averaged 'RIDE_call' output, we do not apply latency shift here.

```
%% Extract single trial data

% Determine number of trials
trial_num = size(data,3);

% Get single trial S component, subtract move3 transformed C and R
results.stS = data - move3(results.c(:, :, ones(1, trial_num)),
round(results.latency_c/cfg.samp_interval)) ...
    - move3(results.r(:, :, ones(1, trial_num)), round((rt-
median(rt))/cfg.samp_interval));

% Get single trial C component, subtract move3 transformed S and R
results.stC = data - results.s(:, :, ones(1, trial_num)) - ...
    move3(results.r(:, :, ones(1, trial_num)), round((rt-
median(rt))/cfg.samp_interval));

% Get single trial R component, subtract move3 transformed S and C
results.stR = data - results.s(:, :, ones(1, trial_num)) - ...
    move3(results.c(:, :, ones(1, trial_num)),
round(results.latency_c/cfg.samp_interval));
```

5. Save single-trial data in EEGLAB format.
   a. Use the original EEGLAB 'EEG' variable.
   b. Overwrite the 'data' field with the specific RIDE decomposed component data.

Note: In the following example, move the C-component data matrix back into the original EEG variable and save the EEG variable as EEGLAB .set file.

   c. Repeat this step also for S and R component data.
   d. Apply a baseline correction to the data before exporting.

The reason for this is that the data may not be properly baseline-aligned after the RIDE decomposition. For MVPA, applying baseline correction is highly recommended. In the example, we use the EEGLAB function 'pop_rmbase' with a time window of -200 to 0 ms relative to the locking point.

Optional: The ADAM toolbox also provides several preprocessing functions, including baseline correction.

⚠ CRITICAL: During the detection of the RTs, make sure to detect the correct response event. Depending on the type of experimental paradigm or segmentation, multiple response events may exist or response events may be missing for specific trials (e.g., if the segmentation contains hit trials but also miss trials).

```
%% Save in EEGLAB format

% C component
% Assign single trial C component to original EEGLAB dataset 'EEG'
% permute results matrix -> channel * datapoint * trial
EEG.data = permute(results.stC,[2,1,3]);

% Perform baseline correction
EEG = pop_rmbase(EEG, [-200 0]);

% Save C component single trial data as .set file
save(fullfile(path, '001_Cue_NUM_rep_stC.set'),'EEG');

% Do the same now for single-trial S and R component
% S component
EEG.data = permute(results.stS,[2,1,3]);
EEG = pop_rmbase(EEG, [-200 0]);
save(fullfile(path, '001_Cue_NUM_rep_stS.set'),'EEG');

% R component
EEG.data = permute(results.stR,[2,1,3]);
EEG = pop_rmbase(EEG, [-200 0]);
save(fullfile(path, '001_Cue_NUM_rep_stR.set'),'EEG');
```

MVPA in ADAM

Timing: 30 min per dataset, depending on the computing performance

In this step, we conduct a set of diagonal decoding (i.e., classification across time) and temporal generalization analyses on the decomposed EEG data by using the ADAM toolbox (Fahrenfort et al., 2018). After the RIDE decomposition, single-trial data are available for RIDE S-, C- and R-component. In the following section, we analyze the C-component data. First, we import the RIDE-decomposed datasets into FieldTrip format. Second, we describe how to run first- and second-level analyses in ADAM. The code example below is based on examples provided by Fahrenfort et al. (2018) and uses the same parameters as the ones deposited by Petruo et al. (2021).

Note: We only present an exemplary analysis of certain classes/conditions of the RIDE C-component here. Except for folder names, none of the other aspects of the following steps is specific to a given RIDE component type. Thus, the order of the different components used for MVPA is irrelevant. Similarly, if RIDE was defined with a lower (i.e., without R-component) or higher number (i.e., two C-components) of components, that should also work just as described in the current example. That is, an iteration with the R-component data can be left out or the C-component can be run twice (first with C1-component and then with C2-component).

6. Import the RIDE-decomposed data.

Note: Matlab users can open "Example code/star_protocol_concatenate.m" to follow this section while using the deposited data. Sample datasets as input for this step are in the folder "Example_data/Concatenate".

⚠ CRITICAL: ADAM requires single-trial datasets in EEGLab or FieldTrip format as input (Fahrenfort et al., 2018). All experimental conditions that should be analyzed using MVPA must be stored in one dataset. Since each RIDE output file contains only trials of one experimental condition, we need to concatenate the datasets first. To be able to associate each trial with its condition, we add a trial information field [1 × number of trials], coding each trial with a condition-specific numerical value. For example, the first condition is coded as 1, the second as 2, and so on. In ADAM it is possible to define classes based on several numeric codes defined in this step.

a. Import the RIDE decomposed output into FieldTrip data format using the FieldTrip function 'ft_preprocessing'.
b. Append the datasets using the function 'ft_appenddata'.
c. Add information on the trial-class association by adding a FieldTrip 'trialinfo' field (dimension number of trials x 1) to the FieldTrip EEG dataset.
d. The aggregated dataset is saved as a Matlab file. Sample datasets for this step can be found in the folder "Example_data/Concatenate".

```
% Initialize fieldtrip
% cd('PATH\TO\FIELDTRIP');
cd('PATH/TO/FIELDTRIP');
ft_defaults;

% Rxamplary data folder
path = 'PATH/TO/Example_data/Concatenate';

% Load EEGLAB .set files using fieldtrip ft_preprocessing function
cfg = [];
cfg.dataset = fullfile(path, '001_Cue_NUM_rep_stC.set');
ft_data1 = ft_preprocessing(cfg);

cfg = [];
cfg.dataset = fullfile(path, '001_Cue_PAR_rep_stC.set');
ft_data2 = ft_preprocessing(cfg);

cfg = [];
cfg.dataset = fullfile(path, '001_Cue_FONT_rep_stC.set');
ft_data3 = ft_preprocessing(cfg);

% Append both datasets into new variable 'data'
cfg = [];
% Do not keep sampleinfo, it would be inconsistent
cfg.keepsampleinfo = 'no';
data = ft_appenddata(cfg,ft_data1,ft_data2,ft_data3);

% Add trialinfo field
data.trialinfo = [ones(numel(ft_data1.trial),1); ones(numel(ft_data2.trial),1)*2;
ones(numel(ft_data3.trial),1)*3];

% Save aggregated data variable as mat file
save(fullfile(path, '001_aggregated.m'),'data');
```

7. Perform the first-level analysis.

Configure the first-level analysis, i.e., single-subject level, in ADAM.

Note: Sample data as input for the following steps can be found in the folder "Example_data/ADAM". Matlab users can run the code star_protocol_ADAM. Check the toolbox paths in the startup file of ADAM beforehand.

a. Define the input files:
   i. Specify the datasets for subjects 1–4. More subjects can be added by adding additional lines.

Note: Exemplary datasets of N=20 subjects are available for this stage. Subject IDs are arbitrarily given between 001 and 023 during recruitment. In case of no-shows, incomplete task performance, equipment failure, etc., the ID number was skipped and not re-used.

```
%% First level analysis

% Reset Matlab search path
restoredefaultpath;

% Run ADAM startup script in install subfolder
cd('PATH/TO/ADAM/install')
startup

% INPUT FILES can be defined one by one
filenames = {
    '001_aggregated'
    '002_aggregated'
    '005_aggregated'
    '006_aggregated'
    };
```

b. Define the classes.
   i. Define the relevant conditions as classes. This is essential to train a classifier to differentiate between experimental conditions based on the decomposed EEG data. In this example, we classify C-component data by differentiating between Task repetition trials in the cued block and Task repetition in the memory block:

```
All_Cue_rep_stC = [1 2 3]; % Assign condition codes 1 to 3
All_Mem_rep_stC = [7 8 9];
```

Note: Similarly, the cue versus memory block effect can be computed for Task repetition. The code examples below specify all the sub-conditions and their combinations. We recommend using the smallest meaningful units (conditions) in the experiment to define the classes. This can be used later for secondary analyses, sanity checks, troubleshooting, etc.

Optional: All the class names below refer to the data type ("_stC" as single-trial C-component data). This might seem redundant since in our case, C-component, S-component, and R-component datasets were created separately for each subject. We have also kept the three

component types separately when the segmented file formats were created as input files for ADAM. Therefore, it is not possible to mix up data and classes from different components. However, in this case, a redundant naming convention can be useful to easily recognize different parts of the code that would be otherwise largely indistinguishable.

```
%%DEFINITION OF CLASSES
%Blocks defined as Cue and Mem
%Trial types defined as rep (repetition) and switch
%Specific task sets (subconditions): NUM, PAR, FONT
%st_C refers to single trial C-component data
Cue_NUM_rep_stC = [1];
Cue_PAR_rep_stC = [2];
Cue_FONT_rep_stC = [3];
Cue_NUM_switch_stC = [4];
Cue_PAR_switch_stC = [5];
Cue_FONT_switch_stC = [6];
Mem_NUM_rep_stC = [7];
Mem_PAR_rep_stC = [8];
Mem_FONT_rep_stC = [9];
Mem_NUM_switch_stC = [10];
Mem_PAR_switch_stC = [11];
Mem_FONT_switch_stC = [12];
All_Cue_rep_stC = [1 2 3];
All_Cue_switch_stC = [4 5 6];
All_Mem_rep_stC = [7 8 9];
All_Mem_switch_stC = [10 11 12];
```

c. Configure the first-level analysis. Define the general configuration parameters in this step. Non-essential parameters are marked as "optional".
   i. Create an empty variable 'cfg'.
   ii. cfg.class_spec: Specify classes. In the example code below, we will classify the observed EEG activity as (1) belonging to a cued task repetition or (2) a memory-based task repetition.
   iii. cfg.datadir: Specify input files location. In our example, input files were stored separately for each component type.
   iv. cfg.filenames: Specify input file names. In this example, the input files were specified in the variable 'filenames' in a previous step.
   v. cfg.outputdir: Specify output folder for the results of the first-level analysis.
   vi. cfg.model (optional): Specify the selected MVPA model.

Note: The 'BDM' backward decoding model is used to predict the experiment's condition (class) based on the neurophysiological pattern. To build the predictive model, ADAM employs Linear Discriminant Analysis (LDA) as a default setting. Another option for model selection would be the forward encoding model (FEM). From the application point of view, BDM should be used to investigate a categorical relationship between EEG data and classes, while FEM should be used in case of a continuous relationship between classes and the neurophysiological data. In the current protocol, the aim is to classify behaviors of Task repetition and Task switching by differentiating between cued and memory-induced situations. Response selection prompted by a cue or by an internal memory are discrete categories, therefore,

BDM was selected. To perform FEM in ADAM, see (Fahrenfort, 2020). BDM is the default setting in ADAM, therefore, this line is optional in the code.

vii. cfg.raw_or_tfr (optional): Perform classification on time-domain data ('raw') or on time-frequency data ('tfr'). The default setting in ADAM is 'raw'. If 'tfr' is selected, ADAM will perform a time-frequency decomposition prior to the classification using the FieldTrip toolbox. The current protocol does not cover this type of analysis, in case of interest, please refer to (Fahrenfort et al., 2018). Of note, choosing 'tfr' will significantly increase the processing time required for classification.

viii. cfg.nfolds (optional): Define the number of folds. Using a value of 5, the classifier will be trained on 80% of the data, and tested on the remaining 20% of the data, iterating this process until all data points have been tested (5-fold training). The average of the consecutive test folds will be used as a final performance index. The number of folds can be increased up to the number of trials in the input file, however, it is not a common practice to use more than ten folds. The default setting is 10, that is, the classifier will be trained on 90% of the data and tested on the remaining 10% of the data. In the current protocol, a 5-fold configuration yielded good classification accuracy and stable generalization patterns. We recommend testing this parameter for every research project.

Note: It is not possible to provide an optimal number for the different research applications. As a general recommendation, if a lower number of folds yields classification around the chance level, try to use more folds.

ix. cfg.class_method (optional): Select the performance metric. Here, we choose the area under the curve (AUC). AUC is the default setting in ADAM, therefore, this line can be skipped. However, since this parameter is essential for interpreting the results, we recommend keeping this line for clarity. This measure originates from signal detection theory and refers to the area under the receiver operating characteristic. That is, when cumulative true positive rates are plotted against the cumulative false-positive rates, the total area covered will determine the AUC value. AUC is the default setting in ADAM. Other included options are accuracy, d', hit rate, and false alarm rate (Fahrenfort et al., 2018).

x. cfg.crossclass: Compute temporal generalization. If 'yes' is specified, a complete matrix will be computed between training and testing time points. If this parameter is 'no' (default setting), only a subset of this data will be available (i.e., when testing and training on the same time point).

Note: It is recommended to set cfg.crossclass to 'yes', since this step does not increase computing time significantly, however, it provides more options for subsequent analyses (Fahrenfort et al., 2018). In the current protocol, performing a temporal generalization analysis on RIDE decomposed data was the main goal, therefore, the whole matrix was calculated.

xi. cfg.channelpool (optional): Select number and types of channels. 'ALL NOSELECTION' selects all available channels for decoding, which is also the default setting. The number of channels will partially determine the number of features in decoding. Generally, a larger number of features can increase the success of classification. Features that contribute to the classification will receive a larger weight than features with a low contribution. However, if it is assumed that certain areas should contribute more to the model, the relevant channels can be preselected in this step, which in turn, can further boost the classification performance. In the case of task switching, there was no such hypothesis, therefore, 'ALL NOSELECTION' was used to allow the model a non-biased channel weighting and to keep the number of features on maximum, at least for the channels. Of note, results of the first-level analysis will be saved in a folder named after the channel

selection. That is, the results of the first-level analysis will be located in a new subfolder called 'ALL NOSELECTION'.

xii. cfg.resample (optional): Perform resampling. To reduce computation time, we down-sample the data to 55 Hz. Importantly, lowering the sampling rate inevitably leads to information loss, which might affect the accuracy of the classification. We have tested this possibility and re-calculated some of our results on 256 Hz. Since there was no observable difference between the models of 55 Hz and 256 Hz data, we have kept the 55 Hz results (Petruo et al., 2021). For new research projects, we recommend a similar approach. Initial exploration on low-resolution data and a subsequent validation with a higher resolution can provide the optimum between computation time and classification accuracy. The default setting is 'no'.

xiii. cfg.erp_baseline (optional): Perform baseline correction. Here, baseline correction was applied already in a previous step. The default setting is 'no'. To run baseline correction in ADAM, add this parameter to the provided code and specify the beginning and the end of the baseline time window in seconds. For example, a 200 ms baseline period before the stimulus presentation would be cfg.erp_baseline = [-.2, .0].

Note: Baseline correction is a linear transformation, users can freely choose when to apply it (before or during ADAM).

```
% GENERAL ANALYSIS CONFIGURATION SETTINGS
cfg = []; % clear the config variable
cfg.class_spec{1} = cond_string(All_Cue_rep_stC);
cfg.class_spec{2} = cond_string(All_Mem_rep_stC);

cfg.datadir = 'PATH/TO/Example_data/ADAM'; % input folder
cfg.filenames = filenames; % specifies filenames
cfg.outputdir = 'PATH/TO/Example_data/ADAM/ALL_REP_CueMem_C'; % output location

cfg.model = 'BDM'; % backward decoding ('BDM')
cfg.raw_or_tfr = 'raw';
cfg.nfolds = 5; % use 5 folds
cfg.class_method = 'AUC'; % Use AUC performance measure
cfg.crossclass = 'yes'; % compute temporal generalization
cfg.channelpool = 'ALL_NOSELECTION'; % channel selection to
cfg.resample = 55;
```

Pause point: Double-check the configuration.

d. Start the classification by running the function 'adam_MVPA_firstlevel'. The output i.e., results of the first-level analysis are stored in the specified output folder.

```
% run first level analysis
adam_MVPA_firstlevel(cfg);
```

8. Perform the second-level analyses.

First, calculate the group-level decoding performance when training and testing were performed on the same time points. This is also known as diagonal decoding. Training and testing data are

separated and iterated according to the 'cfg.nfolds' settings in Configure and run first-level analysis. Second, perform a temporal generalization analysis, which tests the generalizability of the classifier to other time points. In secondary or group-level analyses.

Note: ADAM applies *t*-tests across the individual datasets to compare classification performance against a reference level (Fahrenfort et al., 2018). Classification performance was defined during the first-level analysis in 'cfg.class_method'. Here, we use AUC, therefore, the reference is the chance level of a binary choice (AUC = 0.5). For more details about AUC and alternative performance metrics, see 'cfg.class_method' in the step Configure and run first-level analysis. Input files are provided in "Example_firstlevel_results_ADAM". The related code in the depository is star_protocol_ADAM.m.

a. Perform diagonal decoding.
   i. Evaluate the success of the decoding process by calculating AUC values on the group level. Specifically, the 'adam_compute_group_MVPA' function extracts the results of the first level analyses and calculate the available group statistics on them.
   ii. Define the input folder with the parameter 'cfg.startdir'.

Note: In our example, C-component results of the first-level analysis contrasting Cue and Memory block Task Repetition classes ("Example_firstlevel_results_ADAM/Task repetition"). Running the code example below opens a pop-up window that allows the selection of subfolders. Choose the folder that contains the "ALL_NOSELECTION" subfolder. The subfolder "All_NOSELECTION" was generated according to the 'cfg.channelpool' setting. Importantly, if the files of the first level results need to be moved, make sure that they are still located directly in a folder that is named according to the used 'cfg.channelpool' setting. Troubleshooting: Problem 3: Cannot load data in the second-level analysis of ADAM.

   iii. Specify the time window of the analysis in the field 'cfg.timelim' (optional). For the sample data, select the entire trial length, starting from the baseline period (-200 ms–1,000 ms). Time intervals have to be defined as milliseconds.

⚠ CRITICAL: This step is important if the input files contain larger segments than the interest of the group-level analysis. If the entire segment lengths in the input data are to be analyzed, this line can be skipped.

   iv. Specify correction method for multiple comparisons. Importantly, any group-level analysis is subject to the multiple comparison problem. ADAM has two methods for controlling multiple comparisons: cluster-based permutation testing and False Discovery Rate (FDR). In the first option, significant *t*-tests in adjacent time points constitute a cluster that is equivalent to the sum of the individual *t*-values in that cluster. This process is then iterated according to the 'cfg.iterations' setting. After the iteration, observed cluster sizes can be compared against the null distribution of cluster sizes under random permutation, which then allows the computation of the corrected *p*-values. If FDR was chosen, the correction takes into account the expected proportion of false discoveries (for details, see Fahrenfort et al., 2018). Previous MVPA studies that used RIDE-decomposed EEG data all used cluster-based permutation testing, therefore, for the example datasets, select 'cluster_based' in 'cfg.mpcompcor_method'.

Note: The selection of correction methods should depend on the consideration of the researcher. There is no known aspect of classifying RIDE data that would necessitate either cluster-based or FDR correction.

v. Set the number of iterations (optional). Since the default value is also 1000, the command line of 'cfg.iterations' can be left out completely without changing the group-level results.

Note: Generally, high iteration numbers lead to more accurate cluster-based *p*-values, however, more iterations would significantly increase computation time. Therefore, if initial explorations are needed with the parameters of the decoding analysis, iterations between 250-500 can be used to obtain an estimate of the group-level classification performance. After the parameter space has been set, the final results can be evaluated with 1,000 or more iterations. If the results show spurious clusters, the iteration number might need to be increased above 1000.

vi. Specify the time dimensions for training and testing. The parameter 'cfg.reduce_dims' should be used to specify our interest within the matrix of training and testing time points. In the example below, 'diag' limits the group-level extraction of decoding performance to the case when the classifier was trained and tested on the same points (i.e., the diagonal axis of the testing X training time points matrix). Results are saved in a structure array type of variable and named as 'mvpa_stats' in our example (see section expected outcomes).

vii. Visualize the results with adam_plot_MVPA. This command will prompt a pop-up window in which group-level AUC values (y-axis) are depicted along with the analyzed time window (x-axis, as earlier defined in cfg.timelim) and compared with the corrected significance level. Customized plots can also be generated in ADAM (for details, see Fahrenfort et al., 2018).

```
%% Second level analysis

% Diagonal decoding
% Train and test on the same time point:
cfg = [];
cfg.startdir = 'PATH/TO/Example_firstlevel_results_ADAM
';
cfg.timelim = [-200 1000];
cfg.mpcompcor_method = 'cluster_based';
cfg.iterations = 1000;
cfg.reduce_dims = 'diag'; % diagonal decoding
mvpa_stats = adam_compute_group_MVPA(cfg);

% Plot result
cfg = [];
adam_plot_MVPA(cfg,mvpa_stats);
```

b. Temporal generalization.

Apply the group-level analysis to the whole matrix of testing and training time points. This method is called temporal generalization and its function is to test the generalizability of the classifier to other time points (Fahrenfort et al., 2018; Grootswagers et al., 2016; King and Dehaene, 2014).

Note: It is also possible to specify a time window and/or frequency range for training and testing data. These functions are out of the scope of the current protocol, for their description, see Fahrenfort et al. (2018).

```
% Temporal generalization
% Use all dimensions of the level-1 result
cfg = [];
cfg.startdir = 'PATH/TO/Example_firstlevel_results_ADAM
';
cfg.timelim = [-200 1000];
cfg.mpcompcor_method = 'cluster_based';
cfg.iterations = 1000;
cfg.reduce_dims = '';
mvpa_stats = adam_compute_group_MVPA(cfg);

% Plot result
cfg = [];
adam_plot_MVPA(cfg,mvpa_stats);
```

Alternative: MVPA in MVPA-Light

Timing: 30 min per dataset, depending on the computing performance

The classification across time and temporal generalization analyses on the decomposed EEG data can be alternatively performed using the MVPA-Light toolbox (Treder, 2020). MVPA Light has a larger selection and customization of classifiers and numerous options of performance metrics than ADAM. Furthermore, it includes the option of a searchlight analysis, which is useful for the source estimation of the classification results. Since this is not a feature of ADAM, we recommend MVPA Light for research questions that concern neural source localization.

Note: Similar to ADAM, single-trial single-subject data is required for MVPA-Light. All MVPA analyses (classification across time, temporal generalization) are performed in two levels: single-subject level (level 1) and group level (level 2). In the following section, use the same example of ADAM which classifies C-component data by differentiating between task repetition trials in the cued block and task repetition trials in the memory block. The following steps include single-subject level classification across time, temporal generalization, and corresponding group-level statistics and visualization. To classify S-component and R-component data and to differentiate other conditions, please replace the dataset with the respective RIDE-decomposed dataset and follow the same steps.

Optional: Use "Example code/star_protocol_MVPALight.m" and "Example_data/MVPALight.

⚠ CRITICAL: The steps 9, 10, and 11 must be performed for all subjects, which can best be done using loops.

9. Import the RIDE-decomposed data.

MVPA-Light uses a 3-D [trials × features × time points] array as data input, where features represent electrode channels or voxels. The trials of all classes must be concatenated in a single variable. The condition associated with a certain trial of the aggregated dataset is coded by a numeric value (1 for the first condition, 2 for the second condition) in the trialinfo field (number of trials × 1).

⚠ CRITICAL: Some classifiers (e.g., Support Vector Machine) rely on the value of these class labels (e.g., 1 as positive, -1 as negative) to evaluate the classification performance. Labels

with other numerical codes rather than 1 and 2 cause distortion of classification performance.

a. Import the RIDE decomposed output into FieldTrip data format using the FieldTrip function 'ft_preprocessing'.
b. Append the datasets using the function 'ft_appenddata'.
c. Add information on the trial-class association by adding the field 'trialinfo' to the FieldTrip EEG dataset.
d. Use the FieldTrip function 'ft_timelock' with the parameter cfg.keeptrials set to 'yes' to permute the data to a 3-D matrix. This function can also be used to make a channel or time window selection (parameter 'cfg.channel' and 'cfg.latency').

Note: For classification across time and temporal generalization, all-time points and all electrode channels are recommended to be included in further MVPA.

```
% Initialize FieldTrip
cd('PATH/TO/FIELDTRIP');
ft_defaults;

% Load EEGLAB .set files using FieldTrip ft_preprocessing function
cfg = [];
cfg.dataset = 'SUBJECT1_CONDITION1.SET'
ft_data1 = ft_preprocessing(cfg);

cfg = [];
cfg.dataset = ' SUBJECT1_CONDITION2.SET'
ft_data2 = ft_preprocessing(cfg);

% Append datasets
cfg = [];
% Do not keep sampleinfo, it would be inconsistent
cfg.keepsampleinfo = 'no';
data = ft_appenddata(cfg,ft_data1,ft_data2);

% Add trialinfo field
data.trialinfo = [ones(numel(ft_data1.trial),1); ones(numel(ft_data2.trial),1)*2];

% Permute to trial*channel*timepoint and select time window
cfg = [];
cfg.keeptrials = 'yes';
cfg.latency = conf.mvpa_time;
data = ft_timelockanalysis(cfg, data.data);
```

10. Set parameters for first-level analysis.

A set of parameters need to be specified in MVPA-Light to perform first-level analysis, i.e., single-subject level analysis. These parameters determine the preprocessing steps, classification algorithms, and the expected outcome metrics. All parameters are assigned to a structure variable 'cfg'.

Note: We only describe a few parameters that are necessary or frequently used in the single-subject level MVPA. For further parameters and hyperparameters please refer to Treder (2020).

a. cfg.preprocess: MVPA-Light provides different pre-processing procedures to adapt datasets before training the classifiers, such as sample averaging, over-/under-sampling to balance the trial numbers of different conditions.

Note: It is recommended to preprocess the data with undersampling or oversampling if the numbers of trials in the two conditions strongly differ.

b. cfg.classifier: This is the most important configuration which decides the algorithm used for MVPA. MVPA-Light provided numerous classifiers with flexible customizing settings for each of them. The default classifier is 'lda' (linear discriminant analysis), which is especially time efficient. For noisy datasets, the more robust classifier 'svm' is recommended. For each classifier, its hyperparameters can also be specified in the field 'cfg.hyperparameter'.

Note: It is strongly recommended to try out different classifiers and hyperparameters to finally decide the proper classifier which fits the specific research goal.

c. cfg.metric: Specify the measures for classification output.

Note: 'Accuracy' represents the fraction correctly predicted the class in each participant. Another important one is 'auc' that represents the area under the curve. Notably, 'auc' is only used for binary classification. Other metrics such as 'confusion', 'precision' can also be used as the output. Several metrics can be included in one calculation through a configuration resembling this: 'cfg.metric' = {'accuracy', 'auc','confusion'}.

d. cfg.cv: Specify the type of cross-validation. Cross validation is also provided in MVPA-Light through configuring 'cv' and other related parameters. The parameter 'cfg.cv' specifies the cross-validation type which can be chosen from 'kfold','holdout', and so on.

Note: When cfg.cv='kfold', all data are split into k folds (the number of k is defined in 'cfg.k' with a default value of 5). In each iteration, one fold is held out and used as testing set, the rest folds are training set. This process is iterated until every fold serves as testing set for once. When cfg.cv='holdout', a fraction of data is held out and used as testing data and the rest are used as training data.The number of fraction is defined by setting 'cfg.p'. Cross validation can also be repeated with new randomly assigned folds when setting 'cfg.repeat' and the final result is the average of all repetitions.

11. Run the first-level analyses. After the parameters and class labels have been properly specified, the first-level analysis for each subject can be performed. MVPA-Light provides different analyses using the function 'mv_classify'.
    a. Use the function 'mv_acrosstime' directly for binary classification.
    b. Use the function 'mv_timextime' for temporal generalization.

⚠ CRITICAL: Both functions should be applied on datasets of single subjects, thus the returned result is individual classification performance.

Note: For the convenience of further statistical analysis, it is recommended to run this function in a loop and store all individual results in a cell array as shown in the code below.

```
% Parameters
cfg              = [];
cfg.preprocess   = {'undersample'};
cfg.classifier   = 'svm';
cfg.metric       = {'auc','accuracy'};
cfg.cv           = 'kfold';
cfg.k            = 5;
cfg.repeat       = 2;

% binary classification across time
[~, results_across{1}] = mv_classify_acrosstime(cfg, data.trial, data.trialinfo);

% temporal generalization
[~, results_time{1}] = mv_classify_timextime(cfg, data.trial, data.trialinfo);
```

12. Run the statistical analyses.

Statistical analyses can be done in MVPA-Light for all MVPA tasks to evaluate the classification performance. Different statistical methods can be employed through configuration to fit the results generated from the subject-level MVPA.

a. Select a metric from the classification outcome. Here we use 'auc' for statistics.
b. Specify statistical methods and requirements in 'cfg_stat.test' and corresponding parameter settings. Statistical analyses should be done in all timepoints for binary classification and temporal generalization.
   i. Select a cluster-based permutation test to identify time points where significant classification performance occurred. For each time point, the statistic design follows the purpose of the experiment design and classification goal.
   ii. Choose a within-subject design and compare the 'auc' of the group with the chance level 0.5 to evaluate the classification performance of the classifier. The following code example shows one way to statistically analyze the classification performance.

Note: This code can be applied to both results of subject-level binary classification and temporal generalization.

```
% configuration for statistics
cfg_stat                 = [];
cfg_stat.metric          = 'auc';
cfg_stat.test            = 'permutation';
cfg_stat.correctm        = 'cluster'; % correction method is cluster
cfg_stat.n_permutations  = 1000;

cfg_stat.clusterstatistic = 'maxsum';
cfg_stat.alpha           = 0.05; % standard significance threshold of 5%

% Level 2 (group level) stats design
cfg_stat.design          = 'within';
cfg_stat.statistic       = 'wilcoxon';
cfg_stat.null            = 0.5;
cfg_stat.clustercritval  = 1.96;
stat_level2 = mv_statistics(cfg_stat, results_across);
```

13. Plot the results.

Use the MVPA-Light's provided built-in functions to visualize the classification performance and corresponding statistics.

a. Calculate a grand average among all subjects using selected metrics.
   i. Use the function 'mv_combine_results'
   ii. Select the proper metric via function 'result_average'.
b. Specify parameters for plotting. The grand average classification performance can be plotted through 'mv_plot_result' for classification across time and temporal generalization using two parameters: the averaged results and timepoints for labeling x-axis.

Note: If visualization of statistical results is required, another parameter 'mask' from the variable 'stat_level2' is also needed. The example of plotting the grand average classification performance and statistics for each MVPA task is shown in the following code example.

```
% calculate the grand average across subjects for binary classification, or
temporal generalization, or searchlight analysis
result_average = mv_combine_results(results, 'average');
average_auc = mv_select_result(result_average, 'auc');

% plot for binary classification or temporal generalization
mv_plot_result(average_auc, selected_timepoints, 'mask',stat_level2.mask);
```

EXPECTED OUTCOMES

Here we only present the expected outcomes using the ADAM toolbox. After conducting the second-level analyses in ADAM, classification performance can be interpreted and optionally further analyzed based on (1) the created plots and (2) variables that contain the exact statistics. The figures below can be generated by using the data in "Example_firstlevel_results_ADAM". First, we inspect the plots. If not specified otherwise, plots are named after the name of the folder that was selected in the pop-up panel during the secondary analyses. That is, these are the folders that contain the 'ALL_NOSELECTION' subfolder. Figure 1 shows the plots depicting the ADAM MVPA analysis results based on the example datasets provided in the depository.

The classification of the example dataset of C-component signal yielded images that are clearly interpretable (Figure 1). The time intervals (x axis) that show higher classification accuracy on the diagonals of panels B and D correspond to the time intervals that show significantly above chance classification in diagonal decoding (panels A and C, respectively). For the interpretation of different temporal generalization patterns, we recommend the review of King and Dehaene (2014). Here, we would like to highlight, that not only above-chance but also below-chance classification is possible. We highlight this possibility in an illustration (Figure 2) derived from the original study (Petruo et al., 2021).

Please, note, that the example of Figure 2 shows not only classification significantly above-chance level (in dark red) but also classification significantly below-chance level (in dark blue). Possible interpretations of this type of result can be found in the original article (Petruo et al., 2021), and in other sources (Carlson et al., 2011; King and Dehaene, 2014).

After the visual inspection of the results, we can take a look at the quantification of the classification performance. ADAM saves the relevant parameters in a stats array. Figure 3 depicts a screenshot of the overview of the statistical results based on the example dataset.

The most important metrics from the stats array are the first three and the fifth one as listed in Figure 3. All the other variables record the setting of the analyses (e.g., statistical correction method as

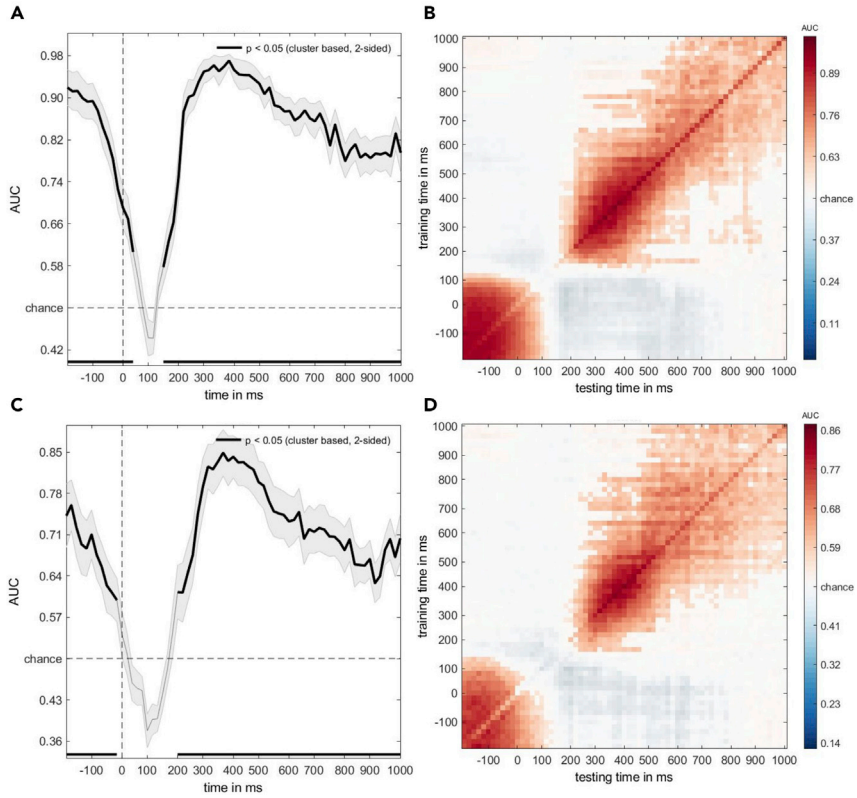


Figure 1. Classification results of cue versus memory trials based on RIDE decomposed C-component EEG data

The upper panel (A and B) shows task the repetition condition, while the lower panel (C and D) shows the task switching condition. Time zero denotes the presentation of the target stimulus.

(A) Area under the curve (AUC) decoding accuracy of the repetition condition when the classifier was trained and tested on the same time points (diagonal decoding). Thicker curve line represents classification performance that is significantly above the chance level.

(B) Temporal generalization plot of the repetition condition when the classifier was trained on a given time point and tested the generalizability of the classifier to other time points. The y axis represents the training time points, while the x axis represents the testing time points. More saturated colors in the matrix indicate good classification performance (i.e., dark red as high AUC value).

(C) Area under the curve (AUC) decoding accuracy of the switching condition when the classifier was trained and tested on the same time points.

(D) Temporal generalization plot of the switching condition when the classifier was trained on a given time point and tested the generalizability of the classifier to other time points.

cluster-based permutation; channel selection as all available ones, etc.). Specifically, ClassOverTime contains the classification performance (in our case, AUCs) on the group level, separately for each time point. Corresponding individual values are available in indivClassOverTime. Similarly, the variability of the classification performance can be seen in StdError and the corrected *p* values in pVals. These metrics can be used to report the exact time intervals of successful classification. Moreover, secondary analyses can be performed on them. For instance, we have selected time intervals based on pVals values to run follow-up source localization analyses in sLORETA (Petruo et al., 2021). Additionally, individual classification performance values from indivClassOverTime have been used to run correlational analyses between behavioral performance and MVPA results (Petruo et al., 2021).

LIMITATIONS

The combination of temporal signal decomposition and decoding methods seem not only feasible but also desirable in areas of human cognition in which multiple aspects of coding are intermixed (Petruo et al., 2021; Prochnow et al., 2021; Takacs et al., 2020a, 2021). This method combination

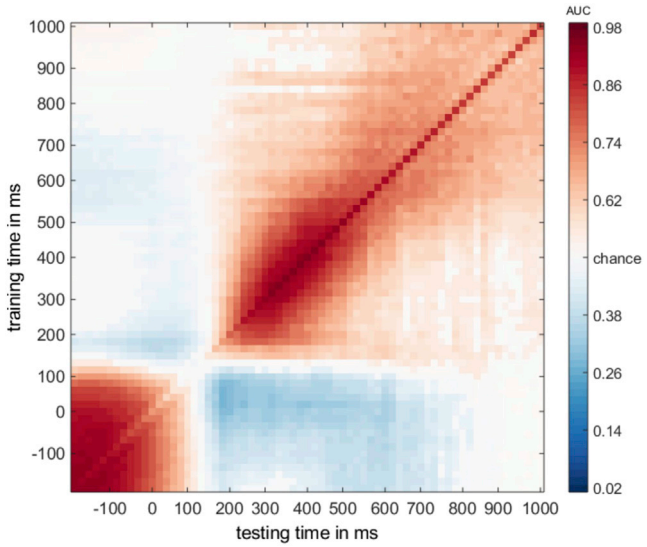


Figure 2. Temporal generalization plot of the switching condition when the classifier was trained on a given time point and tested the generalizability of the classifier to other time points

The y axis represents the training time points, while the x axis represents the testing time points. More saturated colors (dark red or dark blue) in the matrix indicate good classification performance.

has the potential to increase the number of MVPA applications in fields where it is important to disentangle different aspects of the EEG signal. It was suggested, that such distinction can be important cognitive control (Petruo et al., 2021; Prochnow et al., 2021), that has the overlapping motor and non-motor components. However, classes should be defined even more cautiously to decode higher-level, abstract representations. When classes represent higher-order concepts, such as level of effort, switching between goals, or inhibition, researchers have to make sure that lower-level

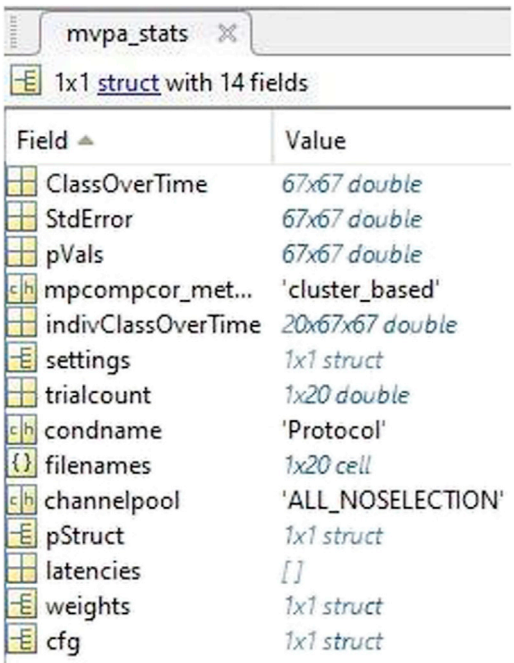


Figure 3. Screenshot from ADAM that highlights the structure of the statistical results

Classification metrics of cue versus memory trials are based on RIDE decomposed C-component EEG data.

aspects of the classes (e.g., physical features of the presented stimuli) do not cause artifacts. This is why it was suggested to define classes based on the smallest meaningful units. Secondary classification analyses that take into account lower-level attributes could exclude the possibility of faulty class distinctions.

Importantly, signal decomposition proved to be useful to enhance the signal-to-noise ratio in the EEG signal. However, removing noise and intermixed signals from the EEG data (e.g., removing stimulus-related S-component signal from the motor-related R-component data) raises the possibility of overfitting. That is, if the classification fits exactly against its training data, MVPA cannot provide meaningful insight into the respective representational dynamics. For instance, an almost perfect classification (i.e., 0.99 or close to 0.99 AUC) could suggest overfitting. Of note, lower AUC values do not exclude overfitting completely. Nevertheless, in the case of almost perfect classification, we recommend investigating this issue further. A good starting point could be plotting individual results.

Troubleshooting: Problem 4: Almost perfect classification. Furthermore, the comparison between undecomposed and RIDE-decomposed classifications might be useful even with non-perfect models. Such comparison may help to delineate whether signal decomposition enhanced a class distinction that was already represented in the undecomposed data albeit faintly (Petruo et al., 2021; Takacs et al., 2021), or decomposition introduced a new pattern that was picked up by MVPA (Takacs et al., 2020b). In the second case, cautious interpretation is needed.

The current protocol is limited to analyses of human scalp EEG data. Researchers who wish to enhance their models by introducing multimodal information sources (e.g., EEG and near-infrared spectroscopy; EEG and cardiovascular signal, etc.) will need to explore the feasibility of using RIDE-decomposed EEG in combination with a variety of neurophysiological channels for MVPA.

Notably, RIDE is only one of the available methods for EEG signal decomposition. Temporal decomposition might be performed by using other means, or researchers might want to explore different techniques, such as spatial decomposition of the EEG signal for subsequent MVPA. We do not know of any aspect that would prevent combining MVPA with other types of EEG signal decomposition methods, however, those applications are beyond the scope of this protocol.

TROUBLESHOOTING

Problem 1

Incorrect baseline interval.

The average amplitudes of the RIDE decomposed data in the baseline interval (output of step 5) differ from 0.

Potential solution

Apply baseline correction.

Apply a baseline correction using the EEGLAB function 'pop_rmbase' on single-trial RIDE output, as described at the end of section temporal signal decomposition in RIDE.

Problem 2

Abnormal behavior of toolbox functions.

After using the ADAM toolbox, several FieldTrip or EEGLAB toolbox functions may not work as expected. For example, the Fieldtrip function 'ft_definetrial' does not accept positive values for the 'trialdef.pre' parameter (all steps).

Potential solution

Reset Matlab search path.

These issues may be caused by version conflicts between toolbox functions. The MVPA-Light toolbox installation package includes several FieldTrip and EEGLAB functions, which may be in conflict with regular EEGLAB/FieldTrip versions. Resetting the Matlab search path by calling the function 'restoredefaultpath' and should solve this.

Problem 3

Cannot load data in the second-level analysis of ADAM.

A typical error message, in this case, is "Error using adam_compute_group_MVPA>drill2data.

Cannot find data, select different location in the directory hierarchy and/or check path settings." (step 8).

Potential solution

Change the level of folder selection.

This problem occurs when the incorrect directory level was selected in the path. ADAM looks for a folder that contains the results of the first-level analysis (e.g., "ALL_NOSELECTION). Make sure that you do not select the folder that contains the first-level result but the one above it in the folder hierarchy.

Problem 4

Almost perfect classification.

Classification performance (output of step 8) is higher than expected (close to 0.99 or higher than in previous research reports)

Potential solution

Inspect single subject results.

In the second-level analysis of ADAM, add the line 'cfg.plotsubjects = true'. If individual results do not show variability, then the design of the original classification is likely faulty.

RESOURCE AVAILABILITY

Lead contact

Further information and requests for resources and reagents should be directed to and will be fulfilled by the lead contact, Ádám Takács (adam.takacs@uniklinikum-dresden.de).

Materials availability

This study did not generate new unique reagents. We provide a subsample of *N*=20 participants (OSF | Example datasets for STAR Protocol) based on data collected by Petruo et al. (2021).

Data and code availability

The example datasets and the codes generated during this protocol are available at the Open Science Forum: OSF | Example datasets for STAR Protocol.

ACKNOWLEDGMENTS

This work was supported by Grants from the Deutsche Forschungsgemeinschaft SFB TRR 265 and FOR 2698.

AUTHOR CONTRIBUTIONS

A.T. and M.M. compiled the example data based on data collected by Petruo et al. (2021). All authors wrote and revised the manuscript.

DECLARATION OF INTERESTS

The authors declare no competing interests.

REFERENCES

Carlson, T.A., Hogendoorn, H., Kanai, R., Mesik, J., and Turret, J. (2011). High temporal resolution decoding of object position and category. J. Vis. *11*, 9. https://doi.org/10.1167/11.10.9.

Delorme, A., and Makeig, S.J. (2004). EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis. Neurosci Methods *134* (1), 9–21. https://doi.org/10.1016/j.jneumeth.2003.10.009.

Dippel, G., Mückschel, M., Ziemssen, T., and Beste, C. (2017). Demands on response inhibition processes determine modulations of theta band activity in superior frontal areas and correlations with pupillometry – implications for the norepinephrine system during inhibitory control. NeuroImage *157*, 575–585. https://doi.org/10.1016/j.neuroimage.2017.06.037.

Fahrenfort, J.J. (2020). Multivariate methods to track the spatiotemporal profile of feature-based attentional selection using EEG. In Spatial Learning and Attention Guidance, Neuromethods, S. Pollmann, ed. (Springer US), pp. 129–156. https://doi.org/10.1007/7657_2019_26.

Fahrenfort, J.J., van Driel, J., van Gaal, S., and Olivers, C.N.L. (2018). From ERPs to MVPA using the Amsterdam decoding and modeling toolbox (ADAM). Front. Neurosci. *12*, 368. https://doi.org/10.3389/fnins.2018.00368.

Folstein, J.R., and Van Petten, C. (2008). Influence of cognitive control and mismatch on the N2 component of the ERP: a review. Psychophysiology *45*, 152–170. https://doi.org/10.1111/j.1469-8986.2007.00602.x.

Grootswagers, T., Wardle, S.G., and Carlson, T.A. (2016). Decoding dynamic brain patterns from evoked responses: a tutorial on multivariate pattern analysis applied to time series neuroimaging data. J. Cogn. Neurosci. *29*, 677–697. https://doi.org/10.1162/jocn_a_01068.

Groppe, D.M., Makeig, S., and Kutas, M. (2009). Identifying reliable independent components via split-half comparisons. NeuroImage *45*, 1199–1211. https://doi.org/10.1016/j.neuroimage.2008.12.038.

Hommel, B. (2009). Action control according to TEC (theory of event coding). Psychol. Res. *73*, 512–526. https://doi.org/10.1007/s00426-009-0234-2.

Huster, R., Plis, S., and Calhoun, V. (2015). Group-level component analyses of EEG: validation and evaluation. Front. Neurosci. *9*, 254. https://doi.org/10.3389/fnins.2015.00254.

King, J.-R., and Dehaene, S. (2014). Characterizing the dynamics of mental representations: the temporal generalization method. Trends Cogn. Sci. *18*, 203–210. https://doi.org/10.1016/j.tics.2014.01.002.

Nunez, P.L., Srinivasan, R., Westdorp, A.F., Wijesinghe, R.S., Tucker, D.M., Silberstein, R.B., and Cadusch, P.J. (1997). EEG coherency: I: statistics, reference electrode, volume conduction, Laplacians, cortical imaging, and interpretation at multiple scalesEEG coherency. Electroencephalogr. Clin. Neurophysiol. *103*, 499–515. https://doi.org/10.1016/S0013-4694(97)00066-7.

Oostenveld, R., Fries, P., Maris, E., and Schoffelen, J.-M. (2010). FieldTrip: open source software for advanced analysis of MEG, EEG, and invasive electrophysiological data. Comput. Intell. Neurosci. *2011*, e156869. https://doi.org/10.1155/2011/156869.

Ouyang, G., Sommer, W., and Zhou, C. (2015a). Updating and validating a new framework for restoring and analyzing latency-variable ERP components from single trials with residue iteration decomposition (RIDE): ERP analysis with residue iteration decomposition. Psychophysiology *52*, 839–856. https://doi.org/10.1111/psyp.12411.

Ouyang, G., Sommer, W., and Zhou, C. (2015b). A toolbox for residue iteration decomposition (RIDE)—a method for the decomposition, reconstruction, and single trial analysis of event related potentials. J. Neurosci. Methods *250*, 7–21. https://doi.org/10.1016/j.jneumeth.2014.10.009.

Petruo, V., Takacs, A., Mückschel, M., Hommel, B., and Beste, C. (2021). Multi-level decoding of task sets in neurophysiological data during cognitive flexibility. iScience *24*, 103502. https://doi.org/10.1016/j.isci.2021.103502.

Petruo, V.A., Mückschel, M., and Beste, C. (2018). On the role of the prefrontal cortex in fatigue effects on cognitive flexibility - a system neurophysiological approach. Sci. Rep. *8*, 6395. https://doi.org/10.1038/s41598-018-24834-w.

Petruo, V.A., Zeißig, S., Schmelz, R., Hampe, J., and Beste, C. (2017). Specific neurophysiological mechanisms underlie cognitive inflexibility in inflammatory bowel disease. Sci. Rep. *7*, 13943. https://doi.org/10.1038/s41598-017-14345-5.

Prochnow, A., Bluschke, A., Weissbach, A., Münchau, A., Roessner, V., Mückschel, M., and Beste, C. (2021). Neural dynamics of stimulus-response representations during inhibitory control. J. Neurophysiol. *126*, 680–692. https://doi.org/10.1152/jn.00163.2021.

Stock, A.-K., Gohil, K., Huster, R.J., and Beste, C. (2017). On the effects of multimodal information integration in multitasking. Sci. Rep. *7*, 4927. https://doi.org/10.1038/s41598-017-04828-w.

Takacs, A., Bluschke, A., Kleimaker, M., Münchau, A., and Beste, C. (2021). Neurophysiological mechanisms underlying motor feature binding processes and representations. Hum. Brain Mapp. *42*, 1313–1327. https://doi.org/10.1002/hbm.25295.

Takacs, A., Mückschel, M., Roessner, V., and Beste, C. (2020a). Decoding stimulus-response representations and their stability using EEG-based multivariate pattern analysis. Cereb. Cortex Commun. *1*, tgaa016. https://doi.org/10.1093/texcom/tgaa016.

Takacs, A., Zink, N., Wolff, N., Münchau, A., Mückschel, M., and Beste, C. (2020b). Connecting EEG signal decomposition and response selection processes using the theory of event coding framework. Hum. Brain Mapp. *41*, 2862–2877. https://doi.org/10.1002/hbm.24983.

Treder, M.S. (2020). MVPA-light: a classification and regression toolbox for multi-dimensional data. Front. Neurosci. *14*, 289. https://doi.org/10.3389/fnins.2020.00289.

van Driel, J., Olivers, C.N.L., and Fahrenfort, J.J. (2021). High-pass filtering artifacts in multivariate classification of neural time series data. J. Neurosci. Methods *352*, 109080. https://doi.org/10.1016/j.jneumeth.2021.109080.

Wolff, N., Roessner, V., and Beste, C. (2016). Behavioral and neurophysiological evidence for increased cognitive flexibility in late childhood. Sci. Rep. *6*, 28954. https://doi.org/10.1038/srep28954.