更新时间：2025年7月5日

\*

天工云枢(SkyNexus)团队

MetaBenchmark-GUI代码说明文档

**目录**

[第一章 代码测试说明 3](#_Toc21949)

[1.1运行环境配置 3](#_Toc26332)

[1.1.1安装基于Qt的PySide6-GUI制作平台 3](#_Toc15820)

[1.1.2安装Benchopt基准测试平台 3](#_Toc5633)

[1.2代码目录结构 4](#_Toc17026)

[1.3 代码功能使用指导 4](#_Toc30992)

[1.3.1 GUI可视化界面 4](#_Toc20199)

[1.3.2 Benchmark基准测试 5](#_Toc6096)

[1.3.3 Analyze数据集分析 9](#_Toc30974)

[1.3.4 Simulation Online模拟在线仿真测试 11](#_Toc4136)

[第二章 基准测试（Benchmark）模块说明 13](#_Toc3898)

[2.1 评估与分析模块（MetaBenchmark-GUI.objective） 14](#_Toc9798)

[2.1.1 Classes: Objective 14](#_Toc16015)

[2.2 算法设计模块（MetaBenchmark-GUI.solvers） 14](#_Toc20185)

[2.2.1 Solver: SSVEP-Decomposition 14](#_Toc32311)

[2.2.2 Solver: P300-Decomposition 16](#_Toc19549)

[2.2.3 Solver: MI-Decomposition 17](#_Toc5721)

[2.3 数据获取模块（MetaBenchmark-GUI.datasets） 18](#_Toc32651)

[2.3.1 Datasets: SSVEP datasets、P300 datasets、MI datasets 18](#_Toc311)

# 第一章 代码测试说明

本章为MetaBenchmark-GUI平台的使用部分，主要用于对GUI界面功能及其操作方法、benchmark基准测试功能及其操作方法、数据集分析功能及其操作方法、模拟在线仿真测试功能及其操作方法四大模块的介绍与说明。

本平台包含：

①主界面(Home)：包含直达MetaBCI Github源、MetaBCI Github源中子平台文件夹、天津大学官网直链，一键调用Brainstim刺激界面，平台功能流式展示台等。

②数据分析界面(Analyze)：包含数据集选择，数据集参数选择，数据集参数表呈现，FFT分析图呈现等。

③模拟在线仿真界面(Simulation Online)：包含数据集选择，在线模拟控制，模拟结果呈现等。

④基准测试界面(Benchmark)：包含范式选择、模型选择、数据集选择、执行控制等。

接下来，将对MetaBenchmark-GUI平台的的使用方法进行说明。

## **1.1**运行环境配置

要安装MetaBenchmark-GUI，首先创建一个新的conda环境，然后激活它。Python版本要求为3.11.9。

conda create -n MetaBenchmark-GUI python=3.11.9

conda activate MetaBenchmark-GUI

或者在配置完成MetaBCI后激活所配置的对于MetaBCI的环境。

以下安装步骤都需要在配置完成MetaBCI后进行。

### **1.1.1**安装基于Qt的PySide6-GUI制作平台

运行以下命令安装PySide6的最新完整版本：

pip install "PySide6-Fluent-Widgets[full]" -i https://pypi.org/simple/

### **1.1.2**安装Benchopt基准测试平台

运行以下命令安装benchopt的最新版本：

pip install -U benchopt

也可以使用最新的开发版本。为此，请改为运行：

pip install git+https://github.com/benchopt/benchopt.git

## **1.2**代码目录结构

metabci

├── benchmarkGUI

│ └── GUI

│ └── core

│ └── ui

│ └── main.py

│ └── ……

│ └── benchmark

│ └── algorithm

│ └── benchmark\_utils

│ └── datasets

│ └── solvers

│ └── output

│ └── objective.py

│ └── benchmark\_run.py

│ └── README.md

├── brainstim

├── brainda

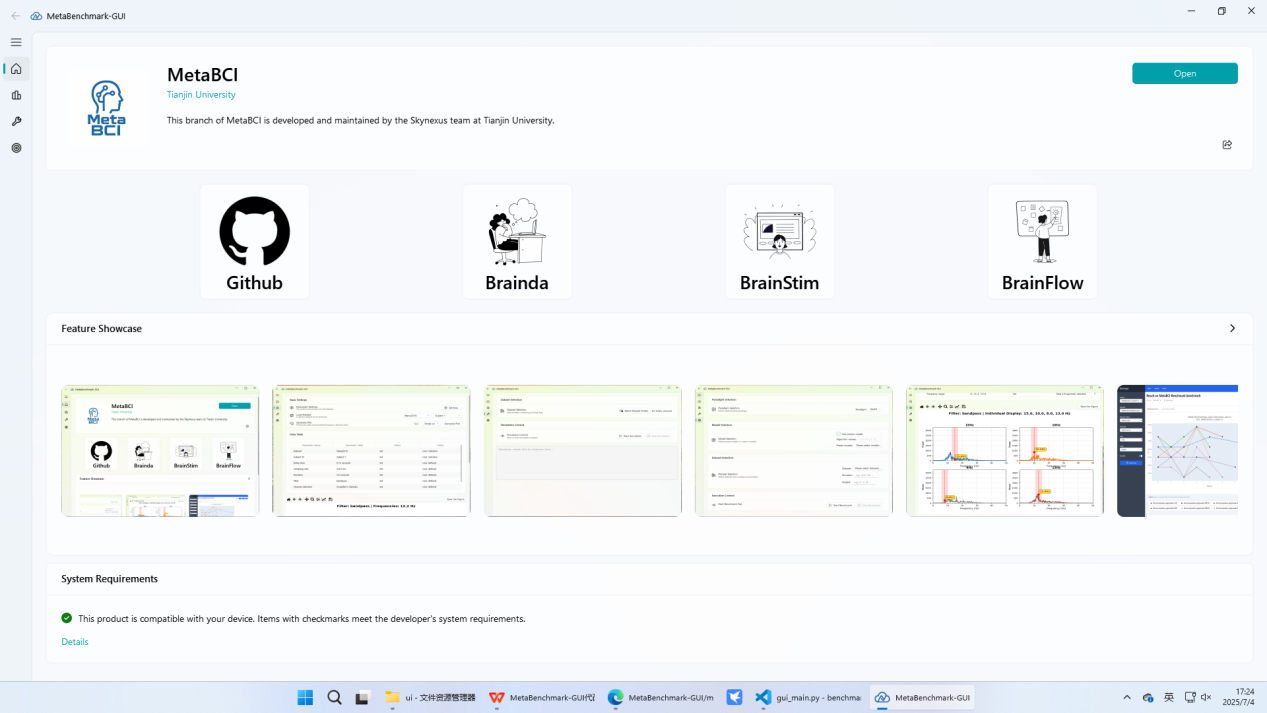
├── brainflow

## **1.3** 代码功能使用指导

### **1.3.1** GUI可视化界面

GUI可视化界面集成了MetaBCI开源平台架构的Brainstim、Brainda、Brainflow三大子平台。使用GUI可视化平台，能够便捷调用MetaBCI中的各部分功能，实现"数据处理-算法调用-设备交互"全流程可视化。

通过metabci\benchmarkGUI\GUI\main.py进入MetaBenchmark-GUI可视化主界面：



MetaBenchmark-GUI可视化界面示意图

使用说明：

1.通过右上角“Open”进入MetaBenchmark-GUI Github源项目。

2.通过中间左一图标“Github”进入MetaBCI Github源项目。

3.通过中间左二图标“Brainda”进入MetaBenchmark-GUI Github源项目中子目录Brainda文件夹下。

4.通过中间右二图标“Brainstim”一键调用MetaBenchmark-GUI Brainstim子平台刺激界面。

5.通过中间右一图标“Brainflow”进入MetaBenchmark-GUI Github源项目中子目录Brainflow文件夹下。

6.界面中间的流式界面显示了本平台的各功能特征。

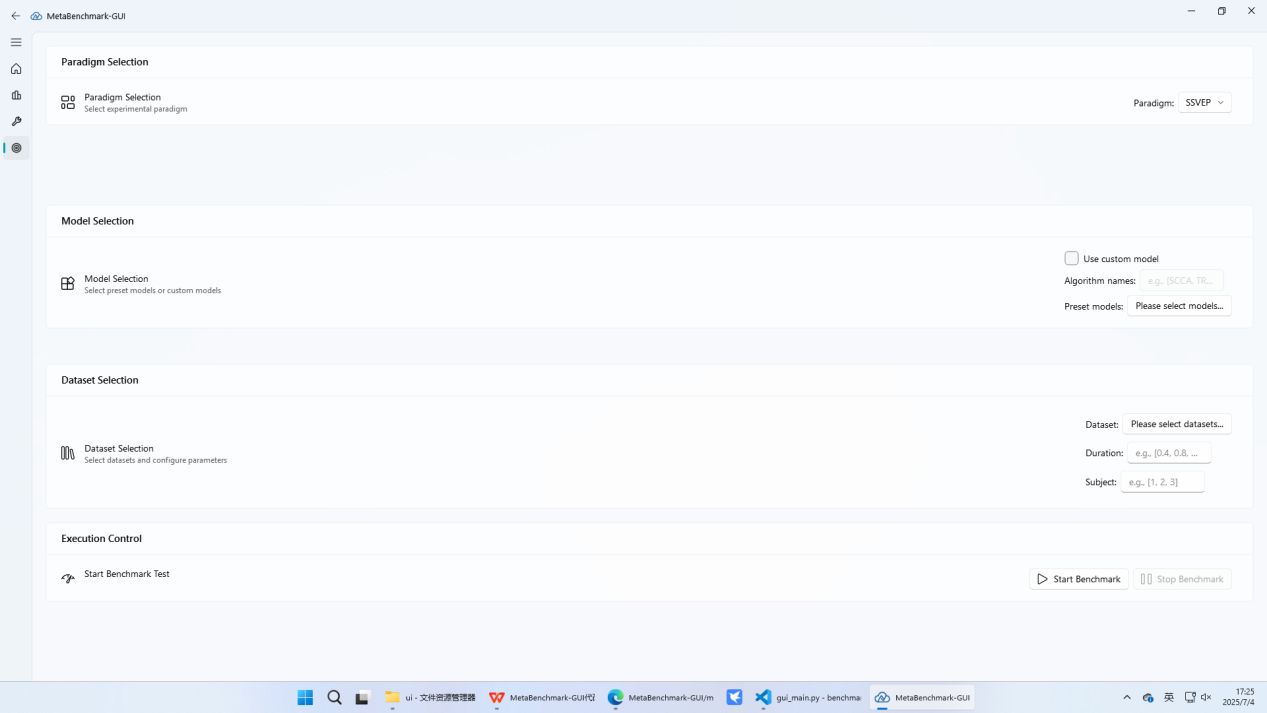
7.主界面左侧栏能够切换不同的功能，分别是主界面(Home)、数据分析界面(Analyze)、模拟在线仿真界面(Simulation Online)、基准测试界面(Benchmark)。

以下是各界面及其功能的详细介绍与使用说明。

### **1.3.2** Benchmark基准测试

MetaBenchmark基准测试基于Benchopt基准测试平台，形成算法性能评估体系。实现对 SSVEP、P300 和 MI 三种脑机交互范式下十余种算法的准确率等关键指标的衡量与评价，并为用户提供自定义算法接口，以便用户能便捷地插入符合格式的对应三种范式的自定义算法进行测试。

进入基准测试界面：



MetaBenchmark-GUI基准测试界面示意图

使用说明：

1.通过“Paradigm Selection”选择进行基准测试的范式：

“SSVEP”:稳态视觉诱发电位。

“P300”:视觉相关诱发电位。

“MI”:运动想象。

2.通过“Model Selection”中的“use custom mode”选择是否需要对自定义算法进行基准测试：  
 “No”:不对自定义算法进行基准测试，自定义算法默认为None。

“Yes”:对自定义算法进行基准测试。添加自定义步骤为：

①将符合格式的相应范式的算法代码文件放入基准测试文件夹Benchmark下的algorithm文件夹

②于algorithm文件夹中的\_\_init\_\_.py代码中加入以下格式的代码——“from .你的代码文件名称 import 基准测试算法类”，如“from .dsp import DSP, DCPM”。

③于基准测试界面中“Algorithm name”输入框中输入自定义算法名称。格式为：基准测试算法类1,基准测试算法类2, ，如：DSP,DCPM, 。

3.通过“Model Selection”中的“Presets models”复选框选择“SSVEP”、“ P300”、“MI”对应范式的预定基准测试算法。此时，选择哪一范式的算法受到第一步中“范式选择”的影响，只能选择第一步中选定范式下的算法：

“SSVEP”：可多选。提供算法如下：ECCA、FBECCA、SCCA、FBSCCA、ItCCA、FBItCCA、TtCCA、FBTtCCA、MsetCCA、FBMsetCCA、MsetCCAR、FBMsetCCAR、TDCA、FBTDCA、TRCA、FBTRCA、TRCAR、FBTRCAR。

“P300”：可多选。提供算法如下：DCPM、SKLDA、LDA、STDA。

“MI”：可多选。提供算法如下：CSP、 FBCSP、 MultiCSP、 FBMultiCSP、 DSP、 FBDSP、 SSCOR、 FBSSCOR。

提示：CSP和FBCSP两个二分类算法不支持bnci2014001、eegbci、schirrmeister2017、weibo2014四种多分类任务的数据集。

4.通过“Dataset Selection”选择“SSVEP”、“P300”、“MI”对应范式的数据集。此时，选择哪一范式的数据集受到第一步中“范式选择”的影响，只能选择第一步中选定范式下的数据集：

“SSVEP”：可多选。提供数据集如下：Wang2016、Nakanishi2015、BETA。选择步骤如下：

①在“Duration”输入框中根据提示输入当前数据集时长。格式为：[提示范围内的一个数, ]，如：[0.2, 0.4, ]。

“Wang2016”范围为np.arange(0.2, 5, 0.2)，

“Nakanishi2015”范围为np.arange(0.5, 4, 0.5)，

“BETA”范围为np.arange(0.2, 2, 0.2)。

②在“Subject”输入框中根据提示输入当前数据集受试者代号。格式为：[提示范围内的一个数, ]，如：[1, 2, ]。

“Wang2016”范围为range(1, 36)，

“Nakanishi2015”范围为range(1, 11)，

“BETA”范围为range(1, 71)。

“P300”：可多选。提供数据集如下：Cattan\_P300。选择步骤如下：

①在“Duration”输入框中根据提示输入当前数据集时长。格式为：[提示范围内的一个数, ]，如：[0.2, 0.4, ]。

“Cattan\_P300”范围为np.arange(0.3, 0.7, 0.1)。

②在“Subject”输入框中根据提示输入当前数据集受试者代号。格式为：[提示范围内的一个数, ]，如：[1, 2, ]。

“Cattan\_P300”范围为range(1, 13)。

“MI”：可多选。提供数据集如下：AlexMI、munichmi、bnci2014001、PhysionetMI、schirrmeister2017、weibo2014。选择步骤如下：

①在“Duration”输入框中根据提示输入当前数据集时长。格式为：[提示范围内的一个数, ]，如：[0.2, 0.4, ]。

“AlexMI”范围为np.arange(0.5, 3.0, 0.5)，

“munichmi”范围为np.arange(0, 7, 0.5)，

“bnci2014001”范围为np.arange(2, 6, 0.2)，

“PhysionetMI”范围为np.arange(0, 3, 0.5)，

“schirrmeister2017”范围为np.arange(0, 4, 0.5)，

“weibo2014”范围为np.arange(3, 7, 0.5)。

②在“Subject”输入框中根据提示输入当前数据集受试者代号。格式为：[提示范围内的一个数, ]，如：[1, 2, ]。

“AlexMI”范围为range(1, 9)，

“munichmi”范围为range(1, 11)，

“bnci2014001”范围为range(1, 10)，

“PhysionetMI”范围为range(1, 110)，

“schirrmeister2017”范围为range(1, 15)，

“weibo2014”范围为range(1, 11)。

③提示：当选择bnci2014001、eegbci、schirrmeister2017、weibo2014数据集时，因其为多分类问题，无法选择CSP和FBCSP两个二分类算法

5.通过“Execution control”中“Start Benchmark”按钮开始选定算法和数据集的基准测试、通过“Stop Benchmark”按钮停止基准测试并重置之前所有选择。运行结果会自动在HTML仪表板中显示，功能如下：

“Part 1: Header”：第一部分包含基准元数据：基准标题和用于运行它的机器的规格。右侧是一个下载按钮，用于将基准的结果下载为.plack文件。

“Part 2: Figure”：第二部分是绘制整个基准运行中跟踪指标的主要图。它的标题显示了目标和数据集名称以及生成图的相应参数。将鼠标悬停在图形上会使模式栏出现在右侧。这可用于与图形交互，例如放大和缩小绘图的特定区域。

“Part 3: Legend”：第三部分图例将每条曲线映射到求解器。单击图例项会隐藏/显示其相应的曲线。同样，双击图例项会隐藏/显示所有其他曲线。最后，将鼠标悬停在图例项上会显示一个工具提示，其中包含有关求解器的详细信息。

“Part 4: Sidebar”：第四部分是功能复选框。前两个下拉菜单**“dataset”**和**“objective”**允许选择基准设置。

**“dataset”**下拉列表包含基准测试中包含的所有数据集及其参数，**“objective”**下拉列表与之相同。

**“objective metrics”**显示了基准运行过程中所有跟踪的指标，这些指标是目标中定义的，与“objective”中evaluate\_result()函数返回的数值相对应。将鼠标悬停在问号上会显示一个工具提示，其中包含有关目标的详细信息。

“SSVEP”：当选择的范式类别为SSVEP时，指标①算法正确率ACC，指标②算法信息传输率ITR。

“P300”：当选择的范式类别为P300时，指标①算法每次迭代单次正确率single\_acc，指标②算法每次迭代平均正确率mean\_acc，指标③算法最终正确率final\_acc。

“MI”：当选择的范式类别为MI时，指标①算法每次迭代单次正确率single\_acc，指标②算法每次迭代平均正确率mean\_acc，指标③算法最终正确率final\_acc，指标④算法信息传输率ITR。

**“scale”**将x和y轴比例设置为线性或对数。

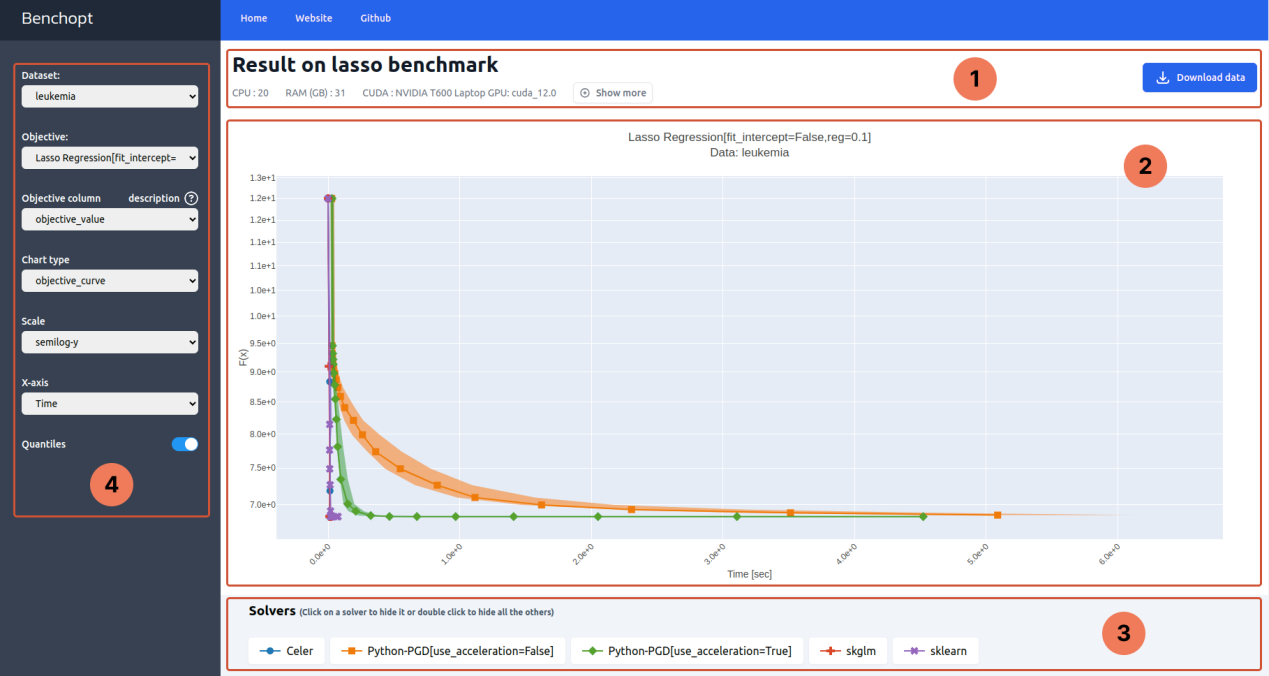
**“X-axis”**更改X轴上绘制的数量，从而将度量绘制为时间、迭代或公差的函数。下拉菜单中迭代和容差的可用性取决于求解器使用的采样策略。此外，它们的解释取决于求解器。

“SSVEP”：采样策略为“run\_once”,“Figure”图示为离散点图。

“P300”：采样策略为“iteration”,“Figure”图示①final\_acc为离散点图，②其余指标为连续曲线。

“MI”：采样策略为“iteration”,“Figure”图示①final\_acc为离散点图，②其余指标为连续曲线。

**“Quantiles”**是一个开关，用于在多次重复运行基准测试时显示/隐藏第95-5个分位数。



MetaBenchmark-GUI基准测试输出示意图

### **1.3.3** Benchmark基准测试报错解决方法

①报错点：UnicodeDecodeError: 'gbk' codec can't decode byte 0xff in position 47: illegal multibyte sequence

解决方法：找到当前环境下的路径：

C:\Users\dell\.conda\envs\MetaBenchmark-GUI\Lib\site-packages\benchopt\utils\shell\_cmd.py

修改shell\_cmd.py中方法\_run\_shell()中第66行代码：

exit\_code, output = subprocess.getstatusoutput(f"{SHELL} {tmp.name})"

为：

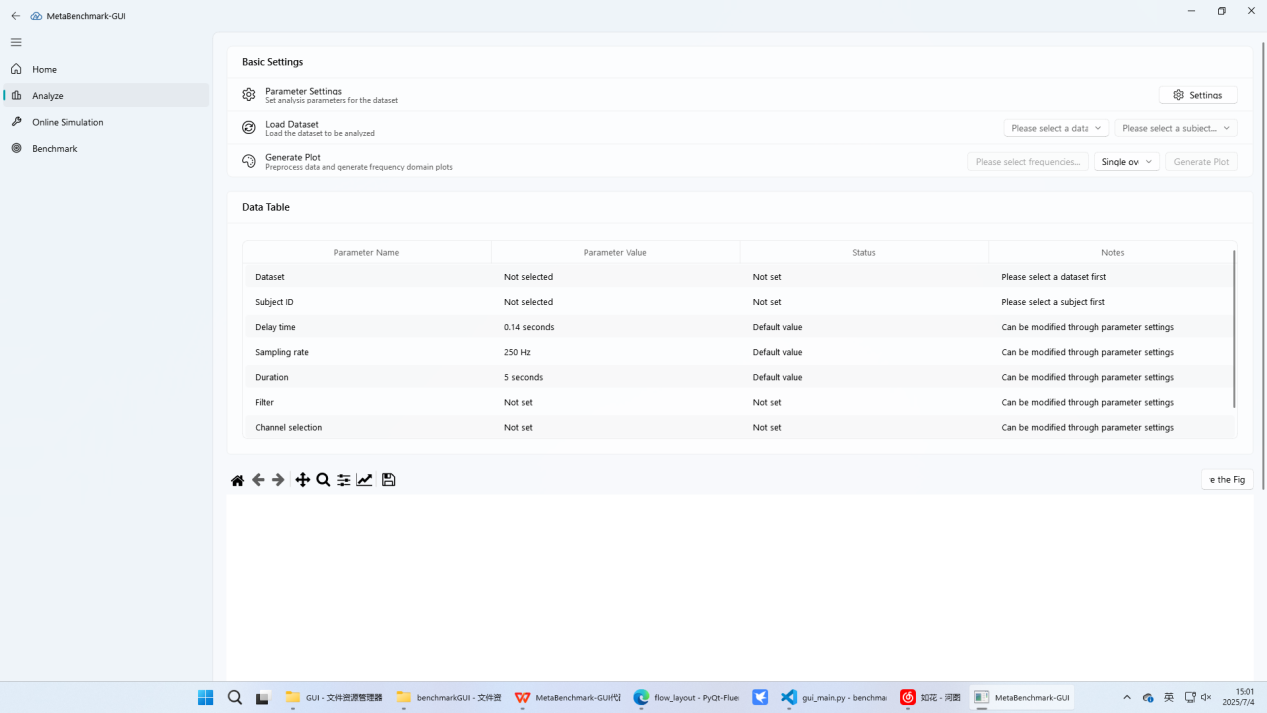
exit\_code, output = subprocess.getstatusoutput(f"{SHELL} {tmp.name}", encoding='latin1')

即可解决。

### **1.3.4 Analyze数据集分析**

Analyze数据集分析基于MetaBCI中Brainda子平台，实现可选本地数据集项目与自选数据集参数的FFT分析，极大地简化FFT分析流程，直观便捷的显示分析结果。

进入数据集分析界面：



MetaBenchmark-GUI数据集分析界面示意图

使用说明：

1.通过“Basic Settings”初始化进行数据集FFT分析的设置：

“Paradigm Settings”:为数据集选择分析参数。包括：

延迟时间(delay time)：单位为秒。

采样率(sampling rate)：单位为赫兹。

数据持续时间(duration)：单位为秒。

滤波：bandpass,lowpass,highpass,notch,butter

导联：Occipital 9 channels, All channels

“Load Dataset”:选择需要分析的数据集及其项目。包括：

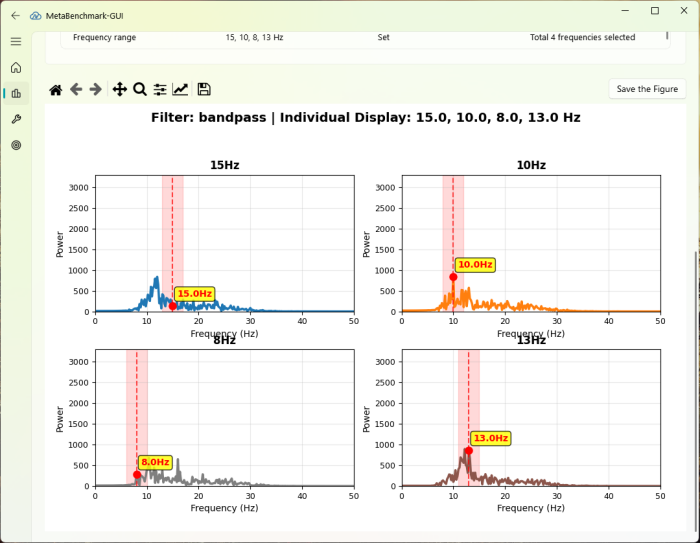
“Wang2016”：Subject1-Subject35

“Nakanishi2015”：Subject1-Subject10

“Select frequencies”:选择需要分析的频率。

2.点击“Generate Plot”生成FFT图形

3.通过“Data table”能够直观的看到分析的相关数据。同时通过下方的图表能直观的观察到相关结果，并且能够对其进行相关操作。

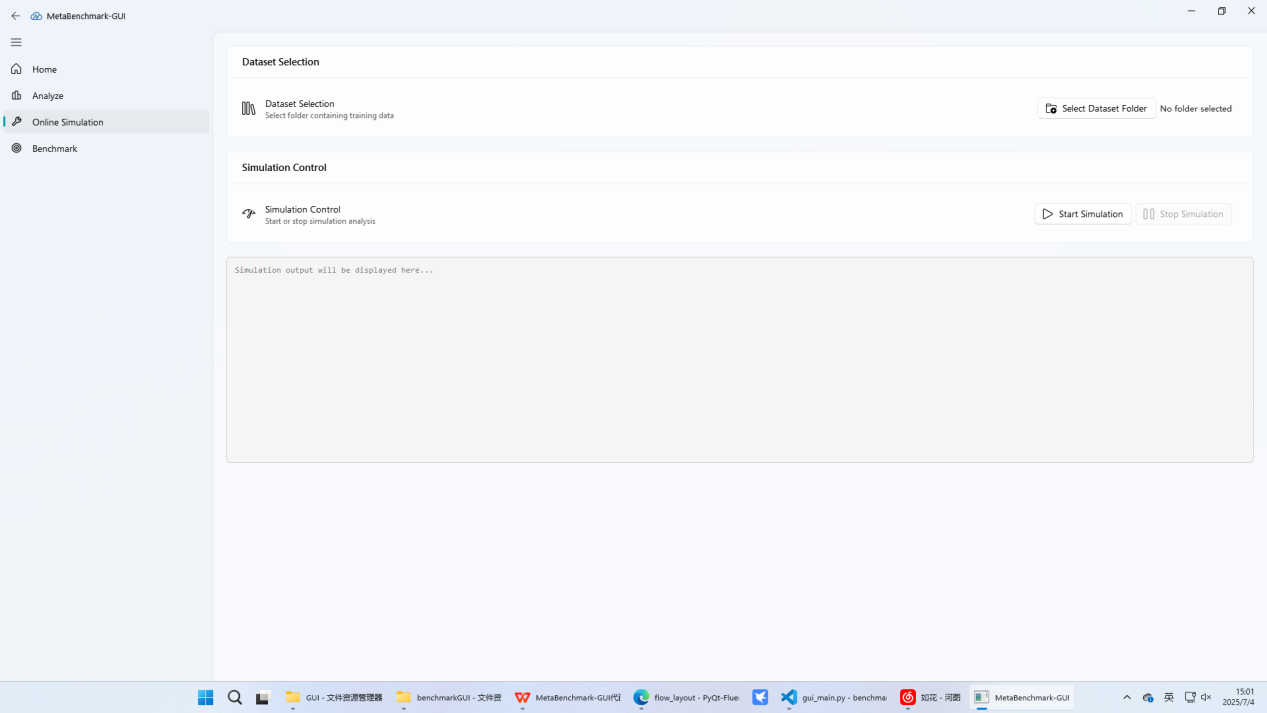


MetaBenchmark-GUI数据集分析输出示意图

### **1.3.5 Simulation Online模拟在线仿真测试**

Simulation Online模拟在线仿真测试基于MetaBCI中Brainflow子平台，实现用离线数据集仿真在线实验，对实验反馈进行动态观察，无需打脑电膏就能观察在线实验可能出现的问题从而及时纠正。

进入模拟在线仿真测试界面：

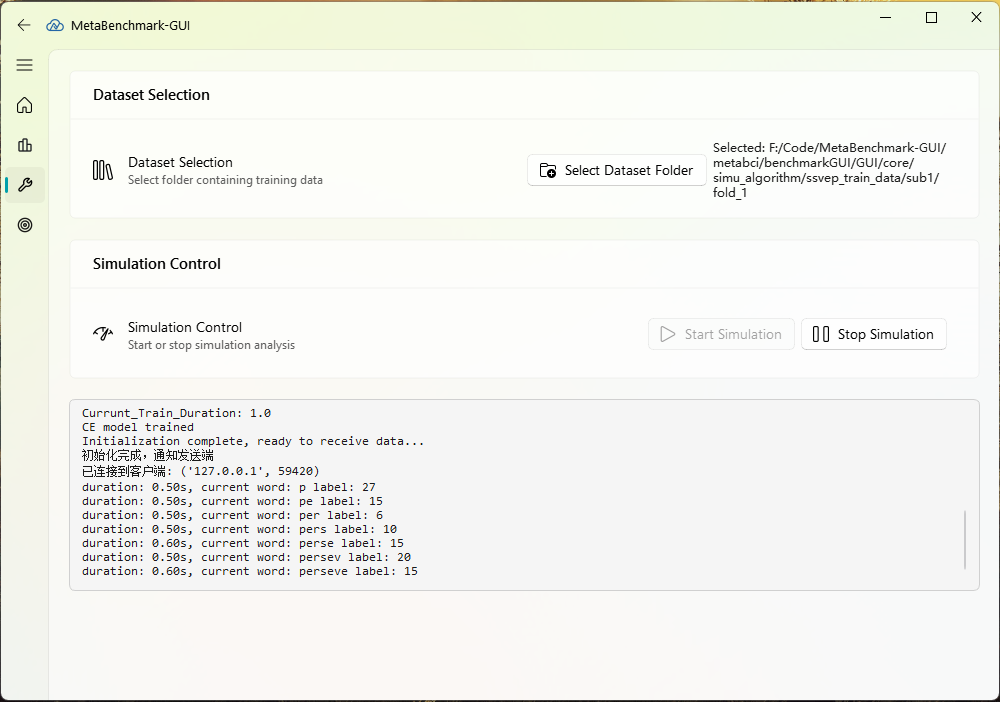


MetaBenchmark-GUI模拟在线仿真测试界面示意图

使用说明：

1.通过“Dataset Selection”选择进行模拟在线仿真测试的数据集及其项目，在\metabci\benchmarkGUI\GUI\core\simu\_algorithm\ssvep\_train\_data\sub1\fold\_1有预定训练集、测试集和标签，能够进行预定模拟在线仿真测试。

2.通过“Simulation Control”中的“Start Simulation”进行模拟在线仿真测试，“Stop Simulation”停止测试。



MetaBenchmark-GUI模拟在线仿真测试输出示意图

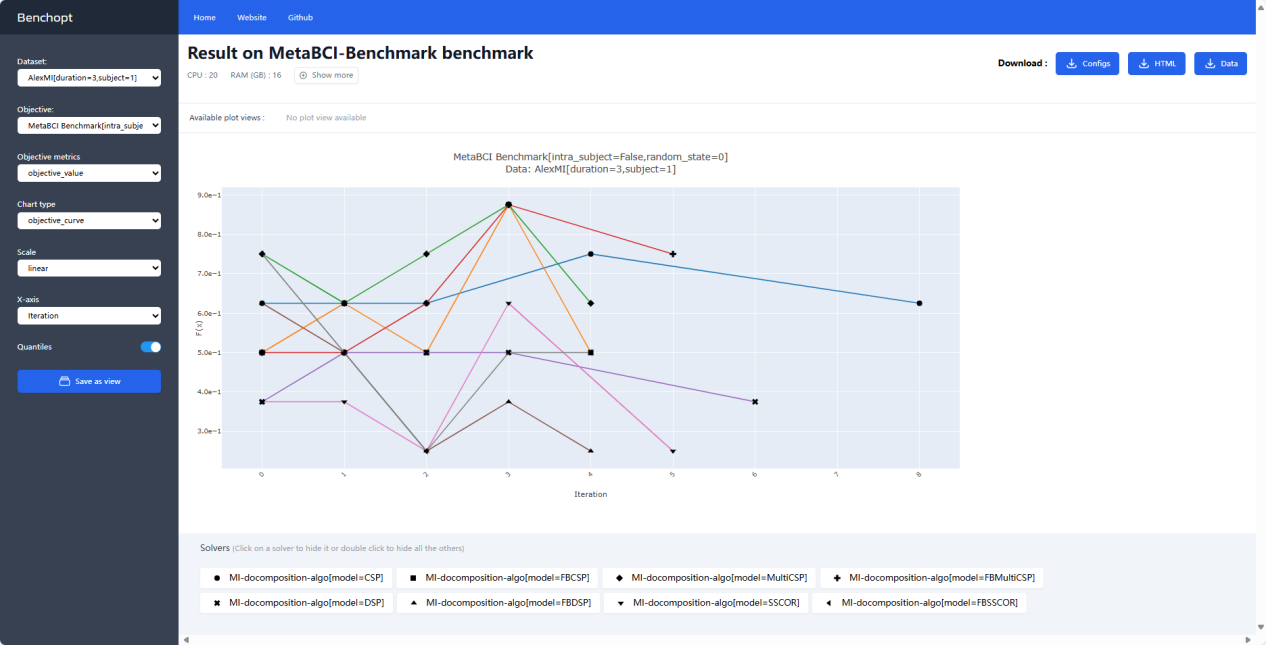
# 第二章 基准测试（Benchmark）模块说明

本章节介绍基准测试(Benchmark)下三大主要的实现函数，主要分为三个功能块，分别由三个程序文件和文件夹实现（评估和分析 objective.py、算法设计solvers文件夹、数据获取datasets文件夹），实现方法和顺序如下图。



benchmark实现方法和顺序示意图

基准测试（Benchmark）是一种对算法性能进行系统性评估的方法。它通过在特定的数据集或问题实例上运行算法，来衡量其效率、准确性和资源消耗等多个方面的表现，是评估和比较不同算法性能的关键手段。MetaBenchmark在脑机接口（Brain computer interface, BCI）领域是一套人为设计的为MetaBCI离线测试定制的基准测试套件，通过三个组件objective、solvers和datasets为三种范式(SSVEP、P300、MI)以及其对应的数据集提供相应算法的准确率衡量与评价。运行结束后，分析结果由默认浏览器中的仪表板窗口显示，如下图所示。



benchmark输出示意图

## 2.1 评估与分析模块（MetaBenchmark-GUI.objective）

### 2.1.1 Classes: Objective

允许在求解器的迭代过程中监视感兴趣的指标。通常，它允许评估求解器要最小化的目标函数。

***class*** **Objective**(BaseObjective)

**Parameters**

**name**: ***str***

基准测试的名字，用于在CLI中选择目标并显示结果。

**random\_state**: ***list***, ***shape(int)***

设置随机数生成器的种子，保证实验的可重复性。

**intra\_subject**: ***list***, ***shape(bool)***

控制是否进行被试内（intra-subject）分析

**requirements**: ***list***, ***shape(str, )***

指定 benchmark 运行所需的基础依赖包（如 numpy）。这些包会自动通过 conda 或 pip 安装。

**min\_benchopt\_version**: ***str***

指定运行本 benchmark 所需的 benchopt 最低版本。

**Methods**

***set\_data*** (*self, X, y, meta, srate, duration, dataset\_name, special\_data*):

将信息存储在数据集上，以便能够计算目标。

***evaluate\_result***(*self, estimator, solver\_state=None*):

根据求解器的输出计算目标值。参数是Solver.get\_result返回结果字典中的键。

***get\_one\_result*** (*self*):

返回一个可以评估目标的结果。此方法主要用于测试目的，检查是否可以调用Objective.comcomputer方法，以及它是否为benchopt返回兼容的类型。返回的对象将传递给Objective.comcomputer。

***get\_objective*** (*self*):

返回求解器的目标参数。

## 2.2 算法设计模块（MetaBenchmark-GUI.solvers）

### 2.2.1 Solver: SSVEP-Decomposition

定义 SSVEP 解码算法的基准测试求解器，支持多种 SSVEP 解码算法。

***class* Solver** *(BaseSolver)*

**Parameters**

**name**: ***str***

SSVEP 求解器的名字，用于在CLI中选择目标并显示结果。

**model**: ***list***, ***shape(None|str,)***

预定算法模型，可在CLI中选择或使用代码运行。

**custom\_model**: ***list***, ***shape(None|str,)***

自定义算法模型，提供用户自定义算法的测试。默认为None。

**module\_name**: ***list***, ***shape(str,)***

预定算法和用户自定义算法的模块来源，可在CLI中选择或使用代码运行。预定算法来源于"decomposition"模块，用户自定义算法放入"algorithm"模块。

**padding\_len**: ***list***, ***shape(None|int,)***

默认为***None***， TDCA/FBTDCA 算法的必须输入参数。

**requirements**: ***list***, ***shape(str, )***

指定 benchmark 运行所需的基础依赖包。这些包会自动通过 conda 或 pip 安装。

**sampling\_strategy**: ***str***

用于控制每个solver的运行方式。"run\_once" 表示：对于每组输入数据和参数组合，这个solver只会被调用一次（即只训练/评估一次）。 "iteration" 进行多次采样、画出运行多次的连续曲线。

**Attributes**

**model**: ***list***, ***shape(None|str,)***

预定算法模型，可在CLI中选择或使用代码运行。

**custom\_model**: ***list***, ***shape(None|str,)***

自定义算法模型，提供用户自定义算法的测试。默认为None。

**module\_name**: ***list***, ***shape(str,)***

预定算法和用户自定义算法的模块来源，可在CLI中选择或使用代码运行。预定算法来源于"decomposition"模块，用户自定义算法放入"algorithm"模块。

**padding\_len**: ***list***, ***shape(None|int,)***

默认为***None***， TDCA/FBTDCA 算法的必须输入参数。

**Methods**

***set\_objective***(*self, X, y, srate, special\_data=None*)*:*

为求解器准备参数并确定算法模型。

***run***(*self, \_* )*:*

使用给定的stop\_val调用求解器。此函数不应返回后续调用get\_result时将检索到的参数。SSVEP中采用"run\_once" 的迭代策略。

***get\_result***(*self*)*:*

返回上一次运行计算的参数。参数应作为字典返回至Objective.evaluate\_result进行评估。

### 2.2.2 Solver: P300-Decomposition

定义P300解码算法的基准测试求解器，支持多种 P300解码算法。

***class* Solver** *(BaseSolver)*

**Parameters**

**name**: ***str***

P300求解器的名字，用于在CLI中选择目标并显示结果。

**model**: ***list***, ***shape(None|str,)***

预定算法模型，可在CLI中选择或使用代码运行。

**custom\_model**: ***list***, ***shape(None|str,)***

自定义算法模型，提供用户自定义算法的测试。默认为None。

**module\_name**: ***list***, ***shape(str,)***

预定算法和用户自定义算法的模块来源，可在CLI中选择或使用代码运行。预定算法来源于"decomposition"模块，用户自定义算法放入"algorithm"模块。

**requirements**: ***list***, ***shape(str, )***

指定 benchmark 运行所需的基础依赖包。这些包会自动通过 conda 或 pip 安装。

**sampling\_strategy**: ***str***

用于控制每个solver的运行方式。"run\_once" 表示：对于每组输入数据和参数组合，这个solver只会被调用一次（即只训练/评估一次）。 "iteration" 进行多次采样、画出运行多次的连续曲线。

**Attributes**

**model**: ***list***, ***shape(None|str,)***

预定算法模型，可在CLI中选择或使用代码运行。

**custom\_model**: ***list***, ***shape(None|str,)***

自定义算法模型，提供用户自定义算法的测试。默认为None。

**module\_name**: ***list***, ***shape(str,)***

预定算法和用户自定义算法的模块来源，可在CLI中选择或使用代码运行。预定算法来源于"decomposition"模块，用户自定义算法放入"algorithm"模块。

**Methods**

***set\_objective***(*self, X, y, srate, special\_data, meta*)*:*

为求解器准备参数并确定算法模型。

***\_stopping\_criterion***(*self*)*:*

使用NoCriterion运行固定迭代。

***\_prepare\_iteration\_data***(*self, k*)*:*

准备第k次迭代的数据，为run函数做准备。

***run***(*self, n\_iter* )*:*

使用给定的stop\_val调用求解器。此函数不应返回后续调用get\_result时将检索到的参数。P300中采用"iteration" 的迭代策略。

***get\_result***(*self*)*:*

返回上一次运行计算的参数。参数应作为字典返回至Objective.evaluate\_result进行评估。

### 2.2.3 Solver: MI-Decomposition

定义MI解码算法的基准测试求解器，支持多种MI解码算法。

***class* Solver** *(BaseSolver)*

**Parameters**

**name**: ***str***

MI求解器的名字，用于在CLI中选择目标并显示结果。

**model**: ***list***, ***shape(None|str,)***

预定算法模型，可在CLI中选择或使用代码运行。

**custom\_model**: ***list***, ***shape(None|str,)***

自定义算法模型，提供用户自定义算法的测试。默认为None。

**module\_name**: ***list***, ***shape(str,)***

预定算法和用户自定义算法的模块来源，可在CLI中选择或使用代码运行。预定算法来源于"decomposition"模块，用户自定义算法放入"algorithm"模块。

**requirements**: ***list***, ***shape(str, )***

指定 benchmark 运行所需的基础依赖包。这些包会自动通过 conda 或 pip 安装。

**sampling\_strategy**: ***str***

用于控制每个solver的运行方式。"run\_once" 表示：对于每组输入数据和参数组合，这个solver只会被调用一次（即只训练/评估一次）。 "iteration" 进行多次采样、画出运行多次的连续曲线。

**Attributes**

**model**: ***list***, ***shape(None|str,)***

预定算法模型，可在CLI中选择或使用代码运行。

**custom\_model**: ***list***, ***shape(None|str,)***

自定义算法模型，提供用户自定义算法的测试。默认为None。

**module\_name**: ***list***, ***shape(str,)***

预定算法和用户自定义算法的模块来源，可在CLI中选择或使用代码运行。预定算法来源于"decomposition"模块，用户自定义算法放入"algorithm"模块。

**Methods**

***set\_objective***(*self, X, y, srate, special\_data, meta*)*:*

为求解器准备参数并确定算法模型。

***\_stopping\_criterion***(*self*)*:*

使用NoCriterion运行固定迭代。

***\_prepare\_iteration\_data***(*self, k*)*:*

准备第k次迭代的数据，为run函数做准备。

***run***(*self, n\_iter* )*:*

使用给定的stop\_val调用求解器。此函数不应返回后续调用get\_result时将检索到的参数。MI中采用"iteration" 的迭代策略。

***get\_result***(*self*)*:*

返回上一次运行计算的参数。参数应作为字典返回至Objective.evaluate\_result进行评估。

## 2.3 数据获取模块（MetaBenchmark-GUI.datasets）

### 2.3.1 Datasets: SSVEP datasets、P300 datasets、MI datasets

数据集定义了可以传递给目标的内容。更具体地说实现了一种方法，输出一个字典，该字典作为关键字参数\*\*data传递给set\_data函数。

2.3.1.1 SSVEP datasets

***class*** **Dataset** (*BaseDataset*)

**Parameters**

**name**: ***str***

数据集的名字，用于在CLI中选择目标并显示结果。

**duration**: ***list***, ***shape(int,)***

选择ssvep 脑电信号的时长，用于后续数据的截取。

**subject**: ***list***, ***shape(int,)***

选择受试者项目编号。

**requirements**: ***list***, ***shape(str, )***

指定 benchmark 运行所需的基础依赖包。这些包会自动通过 conda 或 pip 安装。

**Attributes**

**duration**: ***list***, ***shape(int,)***

选择ssvep 脑电信号的时长，用于后续数据的截取。

**subject**: ***list***, ***shape(int,)***

选择受试者项目编号。

**Methods**

***get\_data*** (*self*):

将数据返回给目标。

2.3.1.2 P300 datasets

***class*** **Dataset** (*BaseDataset*)

**Parameters**

**name**: ***str***

数据集的名字，用于在CLI中选择目标并显示结果。

**duration**: ***list***, ***shape(int,)***

选择ssvep 脑电信号的时长，用于后续数据的截取。

**subject**: ***list***, ***shape(int,)***

选择受试者项目编号。

**requirements**: ***list***, ***shape(str, )***

指定 benchmark 运行所需的基础依赖包。这些包会自动通过 conda 或 pip 安装。

**Attributes**

**duration**: ***list***, ***shape(int,)***

选择ssvep 脑电信号的时长，用于后续数据的截取。

**subject**: ***list***, ***shape(int,)***

选择受试者项目编号。

**Methods**

***get\_data*** (*self*):

将数据返回给目标。

2.3.1.3 P300 datasets

***class*** **Dataset** (*BaseDataset*)

**Parameters**

**name**: ***str***

数据集的名字，用于在CLI中选择目标并显示结果。

**duration**: ***list***, ***shape(int,)***

选择ssvep 脑电信号的时长，用于后续数据的截取。

**subject**: ***list***, ***shape(int,)***

选择受试者项目编号。

**requirements**: ***list***, ***shape(str, )***

指定 benchmark 运行所需的基础依赖包。这些包会自动通过 conda 或 pip 安装。

**Attributes**

**duration**: ***list***, ***shape(int,)***

选择ssvep 脑电信号的时长，用于后续数据的截取。

**subject**: ***list***, ***shape(int,)***

选择受试者项目编号。

**Methods**

***get\_data*** (*self*):

将数据返回给目标。

2.3.1.4 metabci\_utils

动态的导入自定义的模块或者函数，使他们能在不同的求解器或者数据集中使用

**Methods**

***create\_ssvep\_paradigm*** (*srate, channels, interval, events*):

创建ssvep范式实例，同时对数据进行滤波。

***gen\_ssvep\_filterbank*** (*srate, n\_bands=3*):

创建ssvep滤波器组，为后续需要滤波器组的算法提供。

***create\_mi\_paradigm*** (*srate, channels, interval, events*):

创建ssvep范式实例，同时对数据进行滤波。

***gen\_mi\_filterbank*** (*srate, n\_bands=3*):

创建ssvep滤波器组，为后续需要滤波器组的算法提供。

***create\_P300\_paradigm*** (*srate, channels, interval, events*):

创建ssvep范式实例，同时对数据进行滤波。

***gen\_P300\_filterbank*** (*srate, n\_bands=3*):

创建ssvep滤波器组，为后续需要滤波器组的算法提供。