CBGTPy: An extensible cortico-basal ganglia-thalamic framework for modeling biological decision making

Matthew Clapp[†3] Jyotika Bahuguna[†3] Cristina Giossi[†1,2], Jonathan E. Rubin[*4,5], Timothy Verstynen[*3,4], Catalina Vich[*1,2]

1 Departament de Matemàtiques i Informàtica, Universitat de les Illes Balears, Palma, Spain.
2 Institute of Applied Computing and Community Code, Palma, Spain.
3 Department of Psychology & Neuroscience Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, United States of America.
4 Center for the Neural Basis of Cognition, Pittsburgh, Pennsylvania, United States of America.
5 Department of Mathematics, University of Pittsburgh, Pittsburgh, Pennsylvania, United States of America.

[†] These authors contributed equally to this work.

* jonrubin@pitt.edu (JR); * timothyv@andrew.cmu.edu (TV); * catalina.vich@uib.es (CV)

Abstract

Here we introduce CBGTPy, a virtual environment for designing and testing goal-directed agents with internal dynamics that are modeled off of the cortico-basal-ganglia-thalamic (CBGT) pathways in the mammalian brain. CBGTPy enables researchers to investigate the internal dynamics of the CBGT system during a variety of tasks, allowing for the formation of testable predictions about animal behavior and neural activity. The framework has been designed around the principle of flexibility, such that many experimental parameters in a decision making paradigm can be easily defined and modified. Here we demonstrate the capabilities of CBGTPy across a range of single and multi-choice tasks, highlighting the ease of set up and the biologically realistic behavior that it produces. We show that CBGTPy is extensible enough to apply to a wide range of experimental protocols and to allow for the implementation of model extensions with minimal developmental effort.

Author summary

We introduce a toolbox for producing biologically realistic simulations of the cortico-basal ganglia-thalamic pathways during a variety of experimental tasks. The purpose is to foster the theory-experiment cycle, offering a tool for generating testable predictions of behavioral and neural responses that can be validated experimentally, in a framework that allows for simple updating as new experimental evidence emerges. We outline how our toolbox works and demonstrate its performance on a set of normative cognitive tasks.

1 Introduction

1.1 Background

With the rise of fields like cognitive computational neuroscience [1], there has been a resurgence of interest in building biologically realistic models of neural systems that capture prior observations of biological substrates and generate novel predictions at the cellular, systems, and cognitive levels. In many cases, researchers rely on off-the-shelf machine learning models that use abstracted approximations of biological systems (e.g., rate-based activity, rectified linear unit gating, learning by backpropagation) to simulate properties of neural circuits [2–4]. For researchers interested primarily in cortical sensory pathways, these systems work well enough at making behavioral and macroscopic network predictions [5], but they often fail to provide biologically realistic predictions about underlying cellular dynamics that can be tested *in vivo*. In addition, the designs of these off-the-shelf networks are typically meant for engineering purposes, with features that differ significantly from the actual neural substrates of cognition. Their focus on cortical pathway structure can also be poorly suited for considering the computational properties of the vast majority of the neurons in the brain, which lie outside of the neocortex [6]. Of course there are a wealth of biologically realistic simulations of non-cortical pathways in the literature, yet these are often designed to address very narrow behaviors and lack flexibility for testing predictions across multiple experimental contexts.

Here we present a scientifically-oriented tool for creating artificial agents that emulates the control of information streams during decision making in mammalian brains. Specifically, our approach mimics how cortico-basal ganglia-thalamic (CBGT) networks are hypothesized to regulate the evidence accumulation process as agents evaluate response options. The goal of this tool, called CBGTPy, is to provide a simple and easy-to-use spiking neural network simulator that emulates the structural and functional properties of CBGT circuits in a variety of experimental environments. CBGTPy is not meant to engineer fast goal-directed agents that can be deployed in a wide variety of engineering contexts. Instead, the core aim of CBGTPy is to enable researchers to derive neurophysiologically-realistic predictions about basal ganglia dynamics under hypothesis-driven manipulations of experimental conditions.

A key advantage of our CBGTPy framework is that it separates most properties of the behaving agent from the parameters of the environment, such that experimental parameters can be tuned independently of the agent properties and vice versa. We explicitly distinguish the agent (Section 2.3.1) from the environment (Section 2.3.2). The agent generates two behavioral features – action choice and decision time – that match the behavioral data typically collected in relevant experiments and affords users the opportunity to analyze the simultaneous activity of all CBGT nuclei under experimental conditions. The flexibility of the environment component in CBGTPy allows for the simulation of both simple and complex experimental paradigms, including learning tasks with complex feedback properties, such as volatility in action-outcome contingencies and probabilistic reward scenarios, as well as rapid action control tasks (e.g., the stop signal task). On the biological side, CBGTPy achieves realism in terms of aspects of the underlying network pathways and dynamics, a dopamine-dependent plasticity rule [7], and the capacity to emulate targeted stimulation of specific CBGT nuclei (e.g., optogenetic stimulation). CBGTPy also allows the easy addition of novel pathways, as well as modification of network and synaptic parameters, so as to enable modeling new developments in the CBGT anatomy as they emerge in the literature. After a brief review of the CBGT pathways in the next subsection, in Section 2 we provide a full description of the structure, use, and input parameters of CBGTPy. In Section 3, we go on to present examples of its performance on a variety of standard cognitive tasks, before turning to a discussion in Section 4. Various appendices present additional details about the CBGT model and CBGTPy toolbox including the implementation of synaptic plasticity and a guide for CBGTPy installation.

1.2 Overview of the CBGT pathways

While many of the circuit-level details of the basal ganglia (BG) pathways are complex, with new cell types and connections being discovered with increased frequency as our biological tools improve, the consensus conceptualization of the canonical BG circuits has long remained

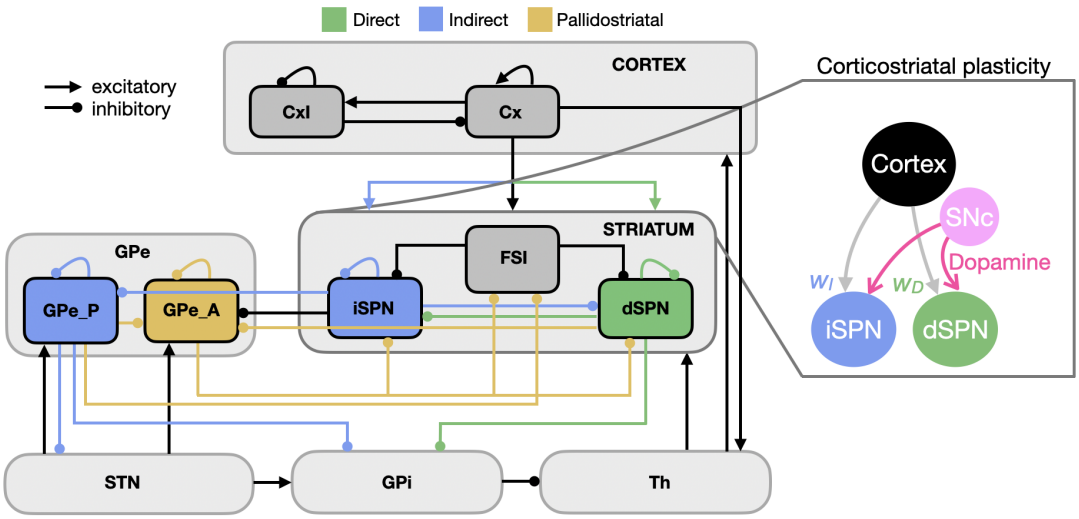

Figure 1: **Overview of the CBGT network.** The connectivity and cellular components of each CBGT channel in a CBGTPy agent. Direct pathway connections are shown in green, indirect pathway connections are shown in blue, and pallidostriatal connections are represented in gold, with circles marking the postsynaptic sites of inhibitory connections and arrows for excitatory connections. Dopaminergic feedback signals associated with rewards following actions induce plastic changes in corticostriatal synapses. Presently the pallidostriatal pathways (gold colors) are only considered for the stop signal task.

stable [8–10]. Much of the theoretical work on this topic relates to a profoundly influential framework for representing the passage of signals through the subcortical circuits, which describes the network as a collection of feed-forward pathways, starting from a cortical input source and flowing to output units that project to certain thalamic nuclei and other subcortical targets [11–15] (Figure 1). In the *direct pathway*, cortical inputs drive a subpopulation of spiny projection neurons (dSPNs) in the striatum. These dSPNs send inhibitory projections directly to basal ganglia output units, which we refer to as the globus pallidus internal segment (GPi), but which can be tuned to represent other outputs such as the substantia nigra pars reticulata (SNr) to suit a user's interests. As these output units are comprised of GABAergic neurons that are suppressed by inhibition from the dSPNs, the traditional view posits that the direct pathway may act to facilitate action selection by disinhibiting populations downstream from the BG.

The *indirect pathway* starts with cortical inputs to a second striatal subpopulation of spiny projection neurons (iSPNs). Like dSPNs, iSPNs send inhibition to the globus pallidus, specifically its external segment (GPe). Unlike the GPi, however, the GPe is not itself an output unit. GABAergic GPe efferents project to the GPi/SNr, to another region called the subthalamic nucleus (STN), and back to the striatum itself. The GPe feedback signals to the striatum, part of what is called the pallidostriatal circuit, relies both on prototypical (GPeP) and arkypallidal (GPeA) GPe cells [16] and is not considered part of the indirect pathway [17]. The STN relays signals to the GPi/SNr, but via glutamatergic rather than GABAergic synapses. Although complex in its structural organization, the "indirect" pathway framework produces a simple prediction: the net effect of cortical activation of iSPNs will be to inhibit GPe neurons. As a result, GPi/SNr neurons are directly disinhibited, and STN neurons are also disinhibited which yields a surge in excitation to GPi/SNr that further enhances their firing and hence the suppression of downstream targets.

A key piece for adapting this framework to action selection is the concept of action channels. These channels represent putative parallel pathways, each responsible for whether a specific action (perhaps a specific muscle contraction or limb movement, perhaps the performance of a more complete action comprising multiple movements) is implemented or suppressed [13]. Thus,

the framework conceptualizes the basal ganglia as a collection of independent action channels [13], with evidence accumulation or other processing occurring in parallel across channels, until this competition results in victory for one action, while the others are suppressed (i.e., winner-take-all selection [18]). Outside of these canonical pathways, however, a third *hyperdirect pathway* stands ready to relay excitation from cortex directly to the STN, providing a proposed mechanism for reactive stopping that can abruptly interrupt or block all actions [14].

Beyond action selection, the CBGT pathways are also notable for their critical role in learning. Nigrostriatal pathways send dopaminergic projections directly to the striatal SPNs [19], where the dSPNs and iSPNs predominantly express D1 and D2 dopamine receptors respectively [20]. Phasic dopamine signals differentially modulate D1 and D2 pathways in response to post-action feedback [21], sculpting corticostriatal synapses over time to effectively promote or inhibit actions in order to maximize future returns. The critical learning signal arises from a discrepancy between a received reward and the expected reward, known as the reward prediction error (RPE), which appears to be encoded in the firing of dopaminergic neurons in the substantia nigra pars compacta (SNc) [22]. Corticostriatal neural plasticity depends on the levels and timing of dopamine signals, which leads to computational models of reinforcement learning that integrate the RPE concept [7, 21, 23, 24]; such a model has even been used to suggest the computational underpinning for the temporal profile of the dopamine signal [25].

Of course, the reality of the CBGT pathways is more complicated than would expected from the simple canonical model. For example, from the onset of a decision process to the execution of an action, dSPN and iSPN activity has been observed to co-vary, contrasting with the idea that the two subpopulations activate at different times as simple "go" or "no-go" switches respectively [26–29]. In addition, activity of these pathways has been found to impact the kinematics of a movement that is made, rather than or in addition to which action is chosen [30–34]. Continuous stimulation of the STN with deep brain stimulation in individuals with parkinsonism has been shown to fundamentally change BG output [35], impacting impulsiveness without compromising the selection of learned actions. Finally, the idea that hyperdirect pathway activation of the STN acts as a brake on planned actions via direct activation of the GPi has come into doubt as new cell types (e.g., arkypallidal cells in the GPe) and new connections (e.g., GPe arkypallidal and thalamic projections to the striatum), have been recognized as playing critical roles in stopping [17, 36–38].

These recent findings have suggested that the simple concepts of rigidly parallel feedforward BG pathways may be outdated, and part of the motivation for CBGTPy is to provide a tool for developing and exploring more nuanced, updated theories of CBGT dynamics as new discoveries are made. Indeed, achieving a full understanding of CBGT circuit-level computations requires the development of theoretical models that can adapt with and complement the rapidly expanding empirical evidence on CBGT pathways. The fundamental goal of the CBGTPy toolbox is to provide a framework for this rapid theoretical development, which balances biological realism with computational flexibility and extensibility.

2 The toolbox

The core of the CBGTpy toolbox comprises an implementation of a spiking model CBGT network tuned to match known neuronal firing rates and connection patterns that have been previously used to study various aspects of basal ganglia function in cognitive tasks [39–42]. The CBGT network model is composed of 6 different regions/nuclei shown in Figure 1: a cortical component, segregated into excitatory (Cx) and inhibitory (CxI) subpopulations; striatum, containing dSPNs, iSPNs, and also fast-spiking interneurons (FSI); GPe, which is divided into prototypical (GPeP) and arkypallidal (GPeA) subpopulations; subthalamic nucleus (STN); GPi; and a pallidal-receiving thalamic component (Th), which receives input from GPi and Cx and projects to cortical and striatal units. The details of the implementation of this network, including all relevant equations and parameter values, are provided in S1 Appendix.

CBGTPy allows for the simulation of two general types of tasks that cover a wide vaeriety of behavioral experiments typically used in neuroscience research. One of these is a discrete decision making task (*n-choice task*) in which the activity of the CBGT network results in the selection of one choice among a set of options. If plasticity is turned on during simulations, phasic

dopamine, reflecting the reward prediction error, is released at the corticostriatal synapses to modify their efficacy, biasing future decisions. We note that the inclusion of a biologically-realistic, dopamine-based learning mechanism, in contrast to the error gradient and backpropagation schemes present in standard artificial agents, represents an important feature of the model in CBGTPy. We present the details of this learning mechanism in S2 Appendix.

The second of these tasks is a stop signal (*stop signal task*) paradigm, where the network must control the execution or suppression of an action, following the onset of an imperative cue. Here activity of the indirect and pallidostriatal pathways, along with simulated hyperdirect pathway control, determines whether a decision is made within a pre-specified time window. The probability of stop and the relevant RT distributions can be recorded under different parameters related to the stop signal. However, in an extended version, the stop signal task will also be contingent on the reward and the dopaminergic plasticity that will influence future behavior.

2.1 Agent-Environment Paradigm

We have adopted an environment-agent implementation architecture, where the internal properties of the CBGT network (the agent) are largely separated from the external properties of the experiment (the environment). Interaction between the agent and environment is limited in scope, as shown in Figure 2, and occurs only at key time points in the model simulation. The core functionality of the agent is the mapping from stimuli to decisions and the implementation of post-decision changes (e.g., synaptic strength updates) to the CBGT network, while the environment serves to present stimuli, cues, and rewards.

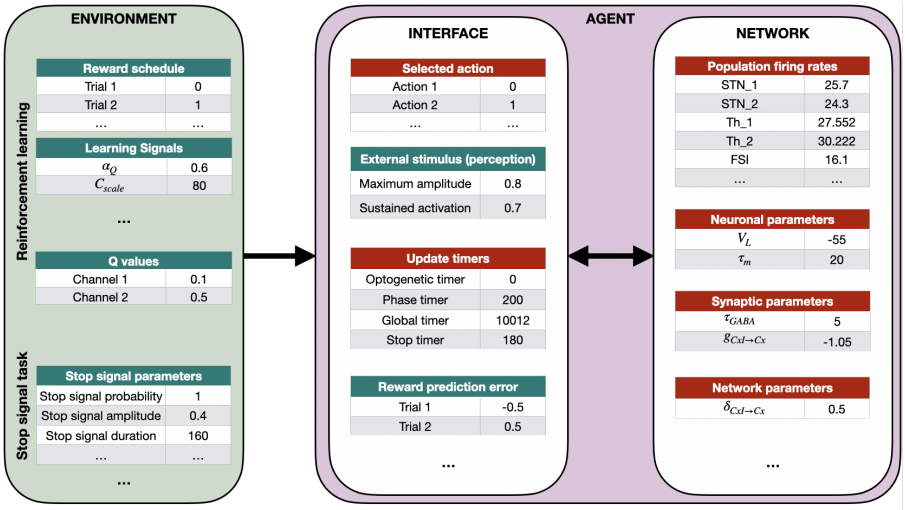

Figure 2: **Example of interactions between the environment and the agent (CBGT network).** The segregation of tasks between the environment and agent allows for independent modification of both. Tables with green title box depict some of the variables that can be easily modified by the user (see Section 2.3). We have divided these variables into two sections: agent-related (Section 2.3.1) and environment-related (Section 2.3.2). The arrow from the environment to the agent block is unidirectional because once the stimulation starts it's not possible to change the environment. The arrow between the interface and network is bidirectional because they are always in constant interaction with each other, the interface controlling the simulation while the actual agent runs the CBGT network.

CBGTPy uses a data-flow programming paradigm, in which the specification of computing steps is separated from the execution of those steps [43]. Internally, the initialization and simulation of the agent-environment system is divided into a large number of specialized functions, each addressing specific tasks. These code blocks are then organized into sequences, referred to as pipelines. Only after a pipeline is constructed is it executed, transforming any input data into output data. The use of pipelines allows for individual code blocks to be rearranged, reused, and modified as necessary, leading to efficient code reuse.

One of the main benefits of the data-flow design is its synergy with the Ray multiprocessing library for Python. Ray operates on a client-server model and allows for the easy distribution of tasks and worker processes based on the available resources [44]. While the sequence of steps for running a simulation can be constructed locally, those same steps can be distributed and performed remotely on the Ray server. As a result, CBGTPy directly supports running on any system that can support a Ray server, which includes high-core-count computing clusters, while maintaining the exact same interface and ease-of-use as running simulations on a local machine.

In the following subsections, we explain all the details of the toolbox by separately describing the agent and environmental components that can be changed by the user. The CBGTPy toolbox can be found in the Github repository `https://github.com/CoAxLab/CBGTPy/tree/main`. The instructions to install it and the list of functions contained in the toolbox can be found in S3 Appendix and S4 Appendix, respectively.

2.2 Setting up a simulation

One of the objectives of CBGTPy is to enable end users to easily run simulations with default experimental setups. Furthermore, users can specify parameter adjustments with minimal effort, more specifically to a `configuration` variable, which we describe in greater detail in the following sections.

The following list contains a mandatory set of instructions to be executed in order to implement the entire process associated with running a simulation. These instructions will proceed with a default set of parameters. We also provide two example notebooks (n-choice task and stop signal task), that can be found in the repository and that include these steps and commands.

- Start the Ray server.
- Import all relevant functions.
- Create the main pipeline.
- Create `configuration` dictionary with default values.
- Run the simulation.
- Extract relevant data frames (e.g., firing rates, reaction times, performance).
- Save variables of interest as pickle files.
- Plot variables of interest (e.g., firing rates and reward data frames).

We explain each of these steps in detail. Note that changing the local IP node (e.g., when the underlying Wi-Fi/LAN network has changed) requires stopping the previous instance of the Ray server and restarting it with the newly assigned IP. We also explain how to shut down the Ray server at the end of this section.

Start the Ray server. On the command line, run Ray server to execute the head node and obtain the local IP node, in the following way:

```
ray start --head --port=6379 --redis-password="cbgt"
```

This command should list a local IP along with a port number. Hence, to initiate a ray client that connects to the server started above, the user will have to substitute the local IP node, obtained from the previous command line, in place of `<local ip node>` in

```
ray start --address=<local ip node>:6379 --redis-password="cbgt"
```

Any port number that is free to use in the machine can be used. Here port number 6379 is used, which is the default port number for Ray.

Import relevant functions. All the relevant imports can be imported with the following commands:

```
import pandas as pd
import numpy as np
import cbgt as cbgt
import pipeline_creation as pl_creat
import plotting_functions as plt_func
import plotting_helper_functions as plt_help
import postprocessing_helpers as post_help
```

Create the main pipeline. Here the user can choose to run either the n-choice task or the stop signal task by assigning a variable `experiment_choice` [1]. Depending on the choice of this variable a relevant pipeline is created. A pipeline consists of all the modules required to run a task and returns a pipeline object that can be used.

For a basic n-choice task, the `experiment_choice` needs to be set as

```
experiment_choice = "n-choice"
```

while for a basic stop signal task, it is set as

```
experiment_choice = "stop-signal"
```

In both cases, the user also should decide how many choices/action channels the current instance of CBGTPy should create and run, using the variable `number_of_choices`:

```
number_of_choices = 2
```

While a usual decision making task is run with `number_of_choices` $= 2$, it can also be run for an arbitrary number of choices (i.e., `number_of_choices` ≥ 1). The change in the number of choices and corresponding action channels requires scaling of some of the parameters in order to ensure maintenance of the same amount of input to certain shared CBGT nuclei irrespective of the number of action channels. We explain these scaling schemata in S5 Appendix. Please note that some parameters have to be appropriately updated in the configuration variables (e.g., Q data frame, channel names) according to the `number_of_choices` selected. We explicitly mention which parameters should be updated with the number of choices as we describe them below, and we include example notebooks in the repository for different cases.

The pipeline is created with commands

```
pl_creat.choose_pipeline(experiment_choice)
pl = pl_creat.create_main_pipeline(runloop=True)
```

Create `configuration` dictionary with default values. The `configuration` variable is a dictionary in which some parameters take internally set default values, whereas some need to be assigned values to run a simulation. A minimal `configuration` variable consists of the following parameters, the details of which are described in S1 Table, S2 Table, S3 Table, S4 Table, S5 Table, S6 Table, and explained separately in Section 2.3.

```
configuration = {
    "experimentchoice": experiment_choice,
    "inter_trial_interval": None, # default = 600ms
    "thalamic_threshold": None, # default 30sp/s
    "movement_time": None, # default sampled from N(250,1.5)
    "choice_timeout": None, # default 1000
    "params": None,
    "pops": None,
    "receps": None,
    "base": None,
    "dpmns": None,
    "d1": None,
```

[1] If the variable "experiment_choice" is not set, the pipeline creation gives an error because of the ambiguity of which experiment to run.

```
    "d2": None,
    "channels": pd.DataFrame([["left"], ["right"]], columns=["action"]),
    "number_of_choices": number_of_choices,
    "newpathways": None,
    "Q_support_params": None,
    "Q_df": None,
    "n_trials": 3,
    "volatility": [1,"exact"],
    "conflict": (1.0, 0.0),
    "reward_mu": 1,
    "reward_std": 0.1,
    "maxstim": 0.8,
    "corticostriatal_plasticity_present":True,
    "record_variables": ["weight", "optogenetic_input"],
    "opt_signal_present": [True],
    "opt_signal_probability": [[1]],
    "opt_signal_amplitude": [0.1],
    "opt_signal_onset": [20.],
    "opt_signal_duration": [1000.],
    "opt_signal_channel": ["all"],
    "opt_signal_population": ["dSPN"],
    "sustainedfraction": 0.75
}
```

Note that the parameter `corticostriatal_plasticity_present` doesn't have to be introduced in the configuration dictionary when running the stop signal task (see reference on the example notebook).

Run the simulation At this stage, the user can choose the number of cores to be used (`num_cores`) and the number of parallel simulations that should be executed with the same `configuration` variable but a different random seed (`num_sims`). The simulation is performed by filling in values for these variables in the following command and executing it:

```
results =
cbgt.ExecutionManager(cores=num_cores).run(
                        [pl]*num_sims, [configuration]*num_sims)
```

The random seed controls the initial conditions of the network, such as precisely which neurons connect together, under the constraint of the given connection probability and the baseline background activity of the CBGT nuclei. The simulation returns a `results` object, which is a dictionary containing all the data produced by the model, typically organized into data frames [45].

Extract relevant data frames Once the `results` object has been returned, specific variables and tables of interest can be extracted (see S1 Fig). All the variables available can be listed by accessing the keys of the variable `results`, which is done by executing the following command:

```
results[0].keys()
```

All environmental variables passed to the simulation can also be accessed here for cross-checking. Some additional data frames related to the simulation are also returned.

One of these data frames is `results[0]["popfreqs"]`, which returns the population firing rate traces of all nuclei, with each neuronal subpopulation as a column and each time bin of simulated time as a row (see S2 Fig). This data frame can be addressed directly by executing its name in a command line.

Another relevant data frame is `datatables[0]`, which contains a list of chosen actions, optimal actions, reward outcomes, and decision times for all of the trials in the simulation (see S3 Fig). When running multiple simulations in parallel (i.e., `num_sims` > 1), `datatables[i]` is returned, where i indicates the corresponding thread. This data frame can be extracted by first executing the command

```
datatables = cbgt.collateVariable(results,"datatables")
```

and next typing `datatables[i]` on the command line, to access the results of the ith simulation.

The Q-values for each action, updated according to the q-learning rule (details in S2 Appendix), are available in `results["Q_df"]`, where each column corresponds to one of each possible choice (see S4 Fig). These data frames are designed for easy interpretation and use in later data processing steps.

CBGTPy also provides a function to extract some of these data frames in a more processed form. The specific command for the n-choice task is given by

```
firing_rates, reward_q_df, performance, rt_dist, total_performance =
plt_help.extract_relevant_frames(results, seed, experiment_choice)
```

where `firing_rates` provides a stacked up (pandas command `melt`) version of the `results[0]["popfreqs"]` that can be used in the seaborn.catplot() plotting function, `reward_q_df` compiles data frames for reward and q-values, `performance` returns the percentage of each decision choice, `rt_dist` returns the reaction time distribution for the simulation, and `total_performance` compares the `decision` and `correctdecision` in `datatables[i]` and calculates the performance of the agent. Depending on the experiment choice, the function returns relevant data frames. Note that for the stop signal task, the data frames returned are just `firing_rates` and `rt_dist`.

The time-dependent values of the recorded variables can also be extracted for both the n-choice task and the stop signal task using the following command:

```
recorded_variables =
post_help.extract_recording_variables(results,
                                      results[0]["record_variables"],
                                      seed)
```

Presently, for the n-choice task, CBGTPy only allows recording the variable `"weight"` or `"optogenetic_input"`. The former can be used to track the evolution of corticostriatal weights during a n-choice experiment. The variable `"optogenetic_input"` can be recorded and plotted to check if the optogenetic input was applied as intended to the target nuclei. An example of plotting these data frames is included in the example python notebook. In addition, for the stop signal task, CBGTPy also allows the recording of the variable `"stop_input"`, which can be used to check if the stop signal inputs were applied correctly to the target nuclei.

Save variables of interest as pickle files All the relevant variables can be compiled together and saved in a single pickle file. The following command saves the object `results` with additional data frames of `popfreqs` and `popdata` into a pickle file `network_data` in the current directory:

```
cbgt.saveResults(results, "network_data", ["popfreqs","popdata"])
```

Basic plotting functions (plot firing rates and reward data frame) CBGTPy provides some basic plotting functions. The `firing_rates` data frame from the above functions can be passed to function `plot_fr` which returns a figure handle that can be saved (see Figure 3), as follows:

```
FR_fig_handles = plt_func.plot_fr(firing_rates, datatables, results,
                                  experiment_choice, display_stim)
```

In addition to the `firing_rates` data frame, the plotting function `plot_fr` requires the `datatables` (also extracted along with `firing_rates` data frame), the original `results` variable, `experiment_choice` and `display_stim`. The `experiment_choice` ensures that relevant nuclei are plotted and the `display_stim` is a boolean variable that can be set to True/False. When set to True, the stimulation information (e.g., optogenetic or stop signal) is indicated over all trials during which the stimulation was applied. Note that, for a longer simulation, this may slow the plotting function, because the function checks if a stimulation is applied every trial before indicating in the figure. The stop signal stimulation is indicated as a bright horizontal red bar above the stimulated nuclei firing traces (e.g., Fig 7), whose onset and offset depend on the stop signal onset and duration specified. The optogenetic stimulation is indicated as a blue bar for excitatory stimulation and yellow for inhibitory stimulation (e.g., Fig 8) with onset and offset according to the onset and offset applied for the optogenetic signal.

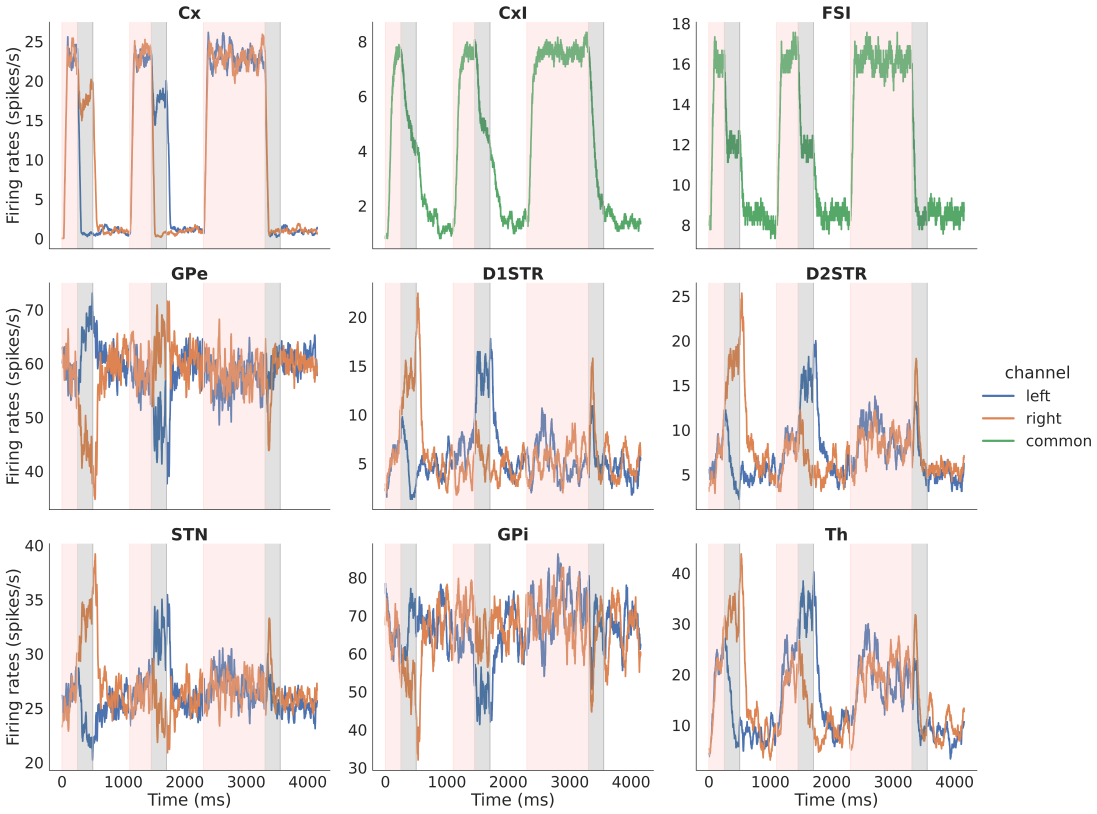

Figure 3: **Example figure showing firing rates for all the nuclei in n-choice task**. Example figure showing firing rates for all nuclei for three consecutive trials on which the model selected a choice in a 2-choice task, color-coded to distinguish times associated with decision making (pink, *decision phase*) and subsequent times of sustained activity in the selected channel (grey, *consolidation phase*) of each trial. The unshaded regions between each pair of trials are the (*inter-trial interval*). In each panel, the blue (orange) trace corresponds to activity in the left (right) action channel. Note that in the third trial, where decision making times out, no sustained activation is applied to the cortical channel (top left subplot) during *consolidation phase* (grey region).

The reward and Q-values data frame can be plotted with the function `plot_reward_Q_df` as follows; note that a figure is shown in the example notebook:

```
reward_fig_handles = plt_func.plot_reward_Q_df(reward_q_df)
```

Shut down Ray server When the user is done working with the network, the Ray server can be shut down with the command

```
ray stop
```

or, if the processes have not all been deactivated, with the command

```
ray stop --force
```

2.3 User level modifications

CBGTPy allows modifications of several parameters that a user can easily perform. All of the parameters in the `configuration` variable can be modified. A modified value is usually in the form of a data frame or a dictionary. The underlying function (`ModifyViaSelector`) in `frontendhelpers.py` iterates through all the features listed in the data frame/dictionary and

updates the default values with the new values passed to the `configuration` variable. If the user wants to use the default value of a parameter, it is essential to specify the parameter value as `None`, as also shown in Section 2.2. We can subdivide the parameters that can be modified into two major classes, which we will discuss separately: a) parameters related to the agent, which we call the agent parameters (Section 2.3.1); b) parameters related to the experimental environment, which we call the environmental parameters (Section 2.3.2).

2.3.1 Agent parameters

General neuron parameters The parameters common to all neurons can be modified using the `params` field in the dictionary `configuration`. A complete list of editable neuronal parameters is listed in S1 Table. For example, type the following command to modify the capacitance (`C`) of all neurons:

```
"params": pd.DataFrame([[30]], columns=["C"]),
```

Population-specific neuronal parameters The neuronal parameters of a specific population can be modified using the field `pops` in the dictionary `configuration`. A complete list of editable parameters is given in S2 Table. In the following example, the membrane time constant (`Taum`) of the neurons in the FSI population is specified:

```
"pops": {"FSI": {"Taum": [60]}},
```

Synaptic parameters The parameters of the synapses (GABA, AMPA or NMDA) can be modified through the field `receps` in the `configuration` variable. A complete list of editable synaptic parameters is given in S3 Table. In the following example, the membrane time constants of AMPA and GABA synapses (`Tau_AMPA`, `Tau_GABA`) are specified:

```
"receps": pd.DataFrame([[100, 100]], columns=["Tau_AMPA", "Tau_GABA"]),
```

Population-specific baseline input parameters Each neuron receives a background input from a random Gaussian process with a specified mean frequency and variance 1. Depending on the nature of the background input, the Gaussian process can be excitatory (AMPA) or inhibitory (GABA). The user can specify the mean frequency, the efficacy, and the number of connections from the background Gaussian process to the neurons in the population with the dictionary `base`. A complete list of editable parameters is given in S4 Table. In the following example, the frequency of the external AMPA inputs (`FreqExt_AMPA`) applied to FSI neurons is specified:

```
"base": {"FSI": {"FreqExt_AMPA": [100]}},
```

Dopamine-specific parameters The dopamine-related parameters can be modified via the `dpmns` parameter, which takes as its input a data frame containing the field name and the names of the parameters to be updated. A complete list of editable parameters is given in S5 Table. In the following example, the dopamine decay rate (`dpmn_tauDOP`) is specified:

```
"dpmns": pd.DataFrame([[5]], columns=["dpmn_tauDOP"]),
```

SPN-specific dopaminergic parameters The SPN-specific dopaminergic parameters for corticostriatal projections to dSPNs and iSPNs can be modified via `d1` and `d2`, respectively, which take as their input a data frame containing the field name and the parameters to be updated. The complete list of editable parameters is given in S6 Table. In the following example, the learning rate (`dpmn_alphaw`) and the maximal value for the corticostriatal weights (`dpmn_wmax`) are specified:

```
"d1": pd.DataFrame([[39.5, 0.08]], columns=["dpmn_alphaw", "dpmn_wmax"]),
"d2": pd.DataFrame([[-38.2, 0.06]], columns=["dpmn_alphaw", "dpmn_wmax"]),
```

New pathways The parameters of a specific pathway can be changed by using the variable `newpathways`. This variable can also be used to add new connections. This takes as its input a data frame that lists the following features of the pathway: source population (`src`), destination population (`dest`), receptor type (`receptor`), channel-specific or common (`type`), connection probability (`con`), synaptic efficacy (`eff`) and the type of the connection (`plastic`), which can be plastic (`True`) or static (`False`). An example is presented in the following command line:

```
"newpathways": pd.DataFrame([["Cx", "GPe", "AMPA", "syn", 0.5, 0.01, False]],
        columns=["src", "dest", "receptor", "type", "con", "eff", "plastic"]),
```

The above command will add a new static (non-plastic) pathway between cortex (Cx) and GPe with connection probability (0.5) and synaptic strength (0.01). Note that, only one pathway at a time can be added in the present version of CBGTPy.

Q-learning algorithm CBGTPy uses Q-learning to track the internal representations of the values of the possible choices, which depend on the rewards received from the environment. The parameters of this algorithm can be modified via the variable `Q_support_params`. The two parameters that can be modified are `C_scale` and `q_alpha`. The former controls the scaling between the change in phasic dopamine and the change in weights, and the latter controls the change in choice-specific q-values with reward feedback from the environment. An example of how to specify these two parameters is presented in the following command line:

```
"Q_support_params": pd.DataFrame([[30, 0.1]], columns=["C_scale", "q_alpha"]),
```

The equations showing the role of these parameters are described in detail in S2 Appendix.

Q-values data frame The choices available to the agent can be initialized with identical values (e.g., 0.5) representing an unbiased initial condition. Alternatively, non-default values for the Q-values data frame (such as values biased towards one choice) can be initialized using the variable `Q_df`. An example of how to specify this variable is presented in the following command line:

```
"Q_df": pd.DataFrame([[0.3, 0.7]], columns=["left", "right"]),
```

Note that this parameter should be updated according to the number of choices specified in the variable `number_of_choices`. The above example shows an initialization for a 2-choice task.

Cortical activity In addition to the background inputs that generate the baseline activity of all of the CBGT nuclei, the cortical component provides a ramping input to the striatal and thalamic populations, representing the presence of some stimulus or internal process that drives the weighing of possible choices. The maximum level of this input can be defined by the parameter `maxstim` and specified using the following command line:

```
"maxstim": 0.8,
```

Corticostriatal plasticity The corticostriatal plasticity in the n-choice experiment can be switched on and off using the `corticostriatal_plasticity_present` parameter. When it is set to `True`, the corticostriatal weights change based on rewards and dopaminergic signals. When it is set to `False`, the simulation proceeds without any update in the corticostriatal weights and the Q-values. The value of this parameter is set to `True` using the following command line:

```
"corticostriatal_plasticity_present": True,
```

Sustained activation to the action channel for the selected choice In order to resolve the temporal credit assignment problem [40], we rely on post-decision sustained activation to keep the selected channel active during the phasic dopaminergic activity [46]. After the choice has been made, following the onset of a trial (*decision phase*), the cortical component of the action channel associated with the selected choice continues to receive inputs, while the unselected channels do not *consolidation phase*. This phase may also represent the movement time of the agent.The assumption here is that this activation provides an opportunity for corticostriatal plasticity that

strengthens the selected choice. The parameter `sustainedfraction` is the fraction of input stimulus maintained during the consolidation phase in the cortical channel corresponding to the action selected by the agent, and it can be specified using the following command line:

```
"sustainedfraction": 0.75,
```

Once the dopamine signal has been delivered, all cortical inputs are turned off for the *inter-trial interval*. See Section 3.1 and Fig 4 for more details on the trial phases.

Thalamic threshold In the default set-up, when the thalamic firing rate of either choice reaches $30\,Hz$, that choice is selected. This threshold can be specified by the user by setting the parameter `thalamic_threshold` in the following way:

```
"thalamic_threshold": 30,
```

2.3.2 Environment parameters

Experiment choice The parameter `experiment_choice` is set at the beginning of the simulation (see Section 2.2). It also needs to be sent as a configuration variable, so that the specific functions and network components relevant to the appropriate experiment are imported.

Inter-trial interval The parameter `inter_trial_interval` allows the user to specify the inter-trial interval duration. The inter-trial interval also corresponds to the duration of *inter-trial-interval* phase of the simulation, where the network receives no external input and shows spontaneous activity. When no value is specified (`None`), a default value of $600\,ms$ is used. The user can set the value of this parameter using the following command line:

```
"inter_trial_interval": 600,
```

Movement time After a choice is made, the chosen action channel receives sustained activation at some fraction (with a default value of 80%) of the initial cortical input strength. As noted in the previous section, this phase of the simulation (*consolidation*) represents the movement time, which is distinct from the reaction time, provides a key window for corticostriatal synaptic plasticity to occur, and remains unaffected by the selected choice. The length of this phase can be controlled with the parameter `movement_time`. The default value of the movement time (when this parameter is set to `None`) is sampled from a normal distribution $\mathrm{N}(250, 1.5)$. However, the user can choose to set it to a constant value by passing a list `["constant", N]`, where `N` represents the constant value of movement time for all trials. The other option is to sample from a normal distribution of specified mean `N` using `["mean", N]`. The movement time can be set to a fixed value as follows:

```
"movement_time": ["constant", 300],
```

Choice time out The parameter `choice_timeout` controls the duration of the time interval in which a choice can be made. The default value of this parameter (when it is set to `None`) is $1000\,ms$. This parameter can be specified as follows:

```
"choice_timeout": 300,
```

Choice labels The data frame `channels` allows the labels for the action channels to be changed. The new labels can be used to access information about the action channels. An example is shown below:

```
"channels": pd.DataFrame([["left"], ["right"]], columns=["action"]),
```

Note that this parameter should be updated according to the number of choices specified in the variable `number_of_choices`. The above example shows an initialization for a 2-choice task.

Number of trials The `n_trials` parameter sets the number of trials to be run within a simulation. Note that this number should be greater than the `volatility` parameter (described in the following paragraph). However, if only 1 trial is to be simulated, then `volatility` parameter should be set to `None`. Examples of how to set this parameter is as follows:

```
"n_trials": 2,

"n_trials": 1,
...
"volatility": [None, "exact"]
```

Volatility The parameter `volatility` indicates the average number of trials after which the reward contingencies switch between the two choices. The volatility parameter is a list consisting of two values, [λ, 'option'], where `option` can be set as `exact` or `poisson`. The λ parameter generates a reward data frame where the reward contingency changes after an average of λ trials. The option `exact` ensures that the reward contingency changes exactly after λ trials whereas the option `poisson` samples the change points from a Poisson distribution with parameter λ. However, note that this parameter cannot be 0 or the total number of trials. To perform a simulation in which the reward contingencies do not change until the end of the simulation, set this parameter to `n_trials`-1 and drop the last trial from the analysis. An example of how to define and specify the volatility is shown in the following command line:

```
"volatility": [2,  "exact"],
```

Please note that for 1-choice task or stop signal task, the volatility parameter is not applicable and hence should be defined differently; specifically, the parameter λ should be set to None as follows:

```
"volatility": [None,  "exact"],
```

Reward probability The parameter `conflict` represents the reward probability of the reward data frame and is defined as a tuple of reward probabilities for the n choices. In the following example, for a 2-choice task, the first reward probability corresponds to the first choice listed in the `channels` parameter (e.g., `"left"`). The reward probabilities for the choices are independent, thereby allowing reward structure to be set in the format (p1, p2) as in the two following examples, representing unequal and equal reward probabilities, respectively:

```
"conflict": (0.75, 0.25),
"conflict": (0.75, 0.75),
```

Note that this parameter should be updated according to the number of choices specified in variable `number_of_choices`. The above example shows an initialization for a 2-choice task. For example, for a 3-choice task the reward probabilities can be defined as:

```
"conflict": (1.0, 0.5, 0.2),
```

Reward parameters The trial-by-trial reward size is generated by a random Gaussian process, with a mean (`reward_mu`) and a standard deviation (`reward_std`) that can be assigned. To simulate binary rewards, choose mean = 1 and standard deviation = 0. An example of how to set these two parameters for rewards that are essentially binary is shown below:

```
"reward_mu": 1,
"reward_std": 0.0,
```

Optogenetic signal If the experiment requires an optogenetic signal to be applied, then this should be indicated in the `configuration` variable by setting `opt_signal_present` to a list of `True`, with each boolean variable corresponding to each nuclei specified in the list of populations to be stimulated, as shown below:

```
"opt_signal_present": [True],
```

The above example shows the case, when only one population is to be stimulated.

Recorded variables CBGTPy allows recording of time-dependent values of both the corticostriatal weights and any optogenetic inputs to any CBGT nuclei that are being stimulated by using the parameter `record_variables`. The former can be used to track the evolution of the weights from cortex to dSPNs or iSPNs during a n-choice experiment. The latter records the optogenetic input applied to the target population and can be recorded for debugging purposes. Both of these variables can be extracted as a data frame by calling the function `extract_recording_variables` as described in Section 2.2. In addition, when running the stop signal task, the stop signal inputs can be recorded as well. Here, we can see an example of how to extract the weights and the optogenetic input:

```
"record_variables": ["weight", "optogenetic_input"],
```

Optogenetic signal probability The `opt_signal_probability` parameter accepts either a float or a list. The float represents the probability of the optogenetic signal being applied in any given trial. Alternatively, a specific list of trial numbers during which optogenetic stimulation should be applied can also be passed, as shown below:

```
"opt_signal_probability": [[0, 1]],
"opt_signal_probability": [1.0],
```

In the above example, a list of trial numbers `[0, 1]` indicates that the optogentic stimulation is applied to trial numbers 0 and 1. A probability of value 1.0 indicates that all the trials will receive the optogenetic stimulation.

Please note, that if more than one nuclei have to be stimulated, the `opt_signal_probability` expects a list of floats (probabilities) or list of list (list of trial numbers).

Optogenetic signal amplitude The amplitude of the optogenetic signal can be passed as a list of float to the parameter `opt_signal_amplitude`. A positive value represents an excitatory optogenetic signal, whereas a negative value represents an inhibitory optogenetic signal. An example of excitation is shown below:

```
"opt_signal_amplitude": [0.1],
```

Optogenetic signal onset The `opt_signal_onset` parameter sets the onset time for the optogenetic signal. The onset time is measured relative to the start of *decision* phase. For example, to specify that optogenetic stimulation will start 20 ms after the *decision* phase begins, the appropriate command is:

```
"opt_signal_onset": [20.],
```

Optogenetic signal duration The duration for which the optogenetic signal is applied can be controlled by the parameter `opt_signal_duration`. This parameter accepts a numeric value in ms as well as the phase names as strings. For example, to apply the optogenetic stimulation for 1000 ms after the signal onset, the command is:

```
"opt_signal_duration": [1000.],
```

However, in order to apply optogenetic stimulation during the whole *decision* phase, the duration variable should be the string "phase 0" as shown below:

```
"opt_signal_duration": ["phase␣0"],
```

This allows the user to specifically target *decision* ("phase 0"), *consolidation* ("phase 1") and *inter-trial interval* ("phase 2") phases with optogenetic stimulation.

Optogenetic signal channel The user can also control whether the optogenetic signal is applied globally to all action channels (`all`), to a randomly selected action channel (`any`), or to a specific action representation (for instance, `left`). To do so, the parameter `opt_signal_channel` needs to be specified as in the example below:

```
"opt_signal_channel": ["all"],
```

Optogenetic signal population The optogenetic stimulation can be applied to a single or multiple populations in the same simulation. In either case, the population names should be defined as a list. The target population can be specified using the parameter `opt_signal_population`. In the example below, the dSPN population is set as the target population:

```
"opt_signal_population": ["dSPN"],
```

Although the optogenetic-related parameters can be used to mimic the effect of a stop signal manifested as the application of a step input current to a population, the network also has a number of modifiable parameters that are specific to injecting the stop signal to target nuclei via a box-shape current.

Stop signal If the experiment requires a stop signal to be applied, then this should be indicated in the `configuration` variable by setting `stop_signal_present` to `True`, as follows: Note that `stop_signal_present` is a list, so, if more stop signals have to be used, it has to be indicated as follows:

```
"stop_signal_present": [True, True],
```

In the present version of the stop signal task, we apply a stop signal to two nuclei: STN and GPeA.

Stop signal probability The stop signal probability can be specified using the parameter `stop_signal_probability`, which can take a float or a list as input. If a float is introduced, it represents the fraction of trials to which the stop signal is applied. These trials are picked randomly from the total number of trials. Alternatively, if a list is specified, then it must contain the numbers of those trials where the user wants the stop signal to be applied. The same formalism is maintained if more than one stop signal is used. An example follows:

```
"stop_signal_probability": [1.0, 1.0],
```

In the above example, a list of two float values represents a 100% probability of applying stop signal to both populations on every trial.

Stop signal populations More than one population at a time can receive a stop signal. The target populations can be specified using the parameter `stop_signal_population`. In the example below, the STN and GPeA populations are set as the target populations:

```
"stop_signal_populations": ["STN", "GPeA"],
```

All the examples presented below corresponding to the stop signal task are designed taking into account that the stop signal is injected into these two populations.

Stop signal amplitude The amplitude of the stop signal can be passed as a float to the parameter `stop_signal_amplitude`. Note, that this parameter is a list and that every value refers to the corresponding population. The order should follow the order of the populations. For example,

```
"stop_signal_amplitude": [0.4, 0.6]
```

Stop signal onset The parameter `stop_signal_onset` sets the times when the stop signals are injected into the target nuclei, respectively. The onset time is measured with respect to the start of phase *decision*. In the example proposed above, one stop signal stimulation starts $30\,ms$ after *decision* phase begins, while the other starts $60\,ms$ after *decision* phase begins:

```
"stop_signal_onset": [30., 60.]
```

Stop signal duration How long each stop signal is maintained can be controlled using the parameters `stop_signal_duration`. In the following example, two stop signals are applied for periods of $100\,ms$ and $160\,ms$, respectively.

```
"stop_signal_duration": [100., 160.]
```

In order to apply the stop signal as long as the duration of a whole phase, the duration variable should be set to a string containing the name of the phase the user wants to apply the stop signal, as shown below:

```
"opt_signal_duration": ["phase␣0"],
```

This allows the user to specifically target *decision*, *consolidation* and *inter-trial interval* phases with stop signal stimulation.

Stop signal channel The user can also control if the stop signal of a specific population is applied to all action channels (`all`), to a uniformly and randomly picked action channel (`any`), or to a specific action representation (for instance, `left`). These can be specified using the parameters `stop_signal_channel`. An example is presented here:

```
"stop_signal_channel": ["all", "left"]
```

3 Experiments

In this section, we present some details about examples of the two primary experiments that CBGTPy is designed to implement.

3.1 A n-choice task in an uncertain environment

This task requires the agent to select between n choices (e.g., left/right in a 2-choice task). The selection of each choice leads to a reward with a certain probability. Moreover, the reward probabilities associated with each choice can be abruptly changed as part of the experiment. Thus, there are two forms of environmental uncertainties that are associated with this task: a) *conflict*, or the degree of similarity between reward probabilities; and b) *volatility*, or the frequency of changes in reward contingencies. A higher conflict would represent a situation where the reward probabilities are similar whereas a lower conflict represents disparate values of reward probabilities. Conflict is not specified directly; rather, the reward probabilities are explicitly set by the user (Section 2.3.2). An environment with high (low) volatility corresponds to frequent (rare) switches in the reward contingencies. The volatility can be set by the parameter λ, which represents the average number of trials after which reward contingency switches. One can choose between whether the trials are switched after exactly λ trials or the change points are sampled from a Poisson distribution with average length λ (Section 2.3.2).

Using the reward probabilities and volatility, the backend code generates a reward data frame that the agent encounters during the learning simulation. The reward data frame is used in calculating the reward prediction error and the corresponding dopaminergic signals, which modulate the plasticity of the corticostriatal projections.

At the beginning of the simulation, the CGBT network is in a resting phase during which all CBGT nuclei produce their baseline firing rates. When a stimulus is presented, the network enters a new phase, which we call *phase 0* or *decision* phase. We assume that at the start of this stage a stimulus (i.e., either an external stimulus or an internal process) is introduced that drives the cortical activity above baseline. Cortical projections to the striatal populations initiate ramping dynamics there, which in turn impacts activity downstream in the rest of the BG and Th. We also assume that when the mean firing rate of a thalamic population exceeds a designated threshold value of 30 spikes per second (the so-called decision boundary), the CBGT network, and hence the agent, has made a choice. This event designates the end of *phase 0*, and the duration of this phase is what we call the *reaction time*. If a decision is not made within a time window $\Delta_{max}\,ms$ after the start of the phase, then we say that none of the available choices have been selected and the decision is recorded as "none". Such trials can be excluded from further analysis depending on the hypothesis being investigated.

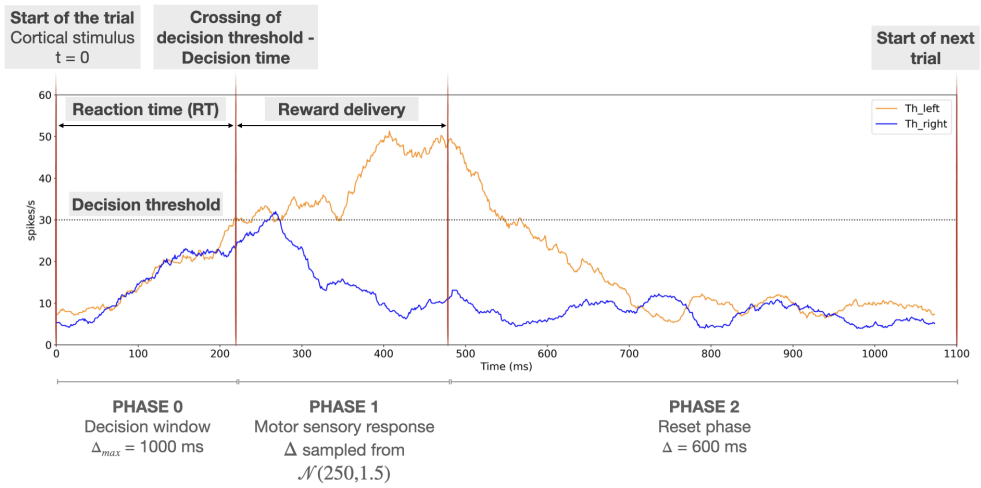

Figure 4: **Representation of the different phases of the simulated decision process.** This sketch represents the simulation of one trial of the 2-choice task in which a left choice is made. The first red vertical thick line, at time 0, represents the time of onset of a ramping stimulus to Cx, which indicates the start of the trial. The second red vertical thick line depicts the decision time (end of *phase 0*). The third red vertical thick line depicts the end of the motor response period associated with the decision (end of *phase 1*). After this line, the reset phase (*phase 2*) starts; this ends after 600 ms (inter trial interval), when a new trial starts (right-hand vertical red line). Orange and blue traces represent the mean thalamic firing rates Th_i for $i \in \{\text{left, right}\}$, respectively, and the horizontal black dotted line highlights the decision threshold.

To allow for selection between n (e.g., two) different choices, i with $i \in \{left, right\}$, we duplicate all CBGT populations except FSI and CxI. This duplication sets up action channels representing the two choices that can influence each other indirectly through the two shared populations and otherwise remain separate over the whole CBGT loop. To distinguish between the firing rate of the populations within channels, we will call them Pop_i, where Pop refers to the corresponding CBGT region and i refers to the channel name (e.g., Cx_{left}, Cx_{right}).

The presentation of a stimulus to the cortical population is simulated by increasing the external input frequency in all copies of the cortical Cx populations that ramp to a target firing rate I_{target}. The ramping current $I_{ramp}(t)$ is calculated as below:

$$I_{ramp}(t) = I_{ramp}(t - dt) + 0.1\big[I_{target}(t) - I_{ramp}(t - dt)\big]$$

where dt is the integrator time step, and the external input frequency also changes according to

$$f_{ext,x}(t) \leftarrow f_{ext,x}(t) + I_{ramp}(t).$$

After a Th population reaches the threshold and hence a decision is made, the ramping input to Cx is extinguished and a subsequent period that we call *phase 1* or *consolidation* phase begins, which by default is sampled from a normal distribution $\mathcal{N}(\mu = 250ms, \sigma = 1.5ms)$ but can also be fixed to a given duration. This phase represents the period of the motor implementation of the decision. Activity during *phase 1* strongly depends on what happened in *phase 0* such that, if a decision i occurred at the end of phase 0, then Cx_i will be induced to exhibit sustained activity [46] (see S2 Appendix), while in any non-selected action channels, the cortical activity returns to baseline. If no decision has been made by the network (within a time window Δ_{max} ms, with a default value of 1000 ms), then no sustained activity is introduced in the cortex (see Figure 3, 3rd choice, the top left subplot showing cortical activity).

Finally, each trial ends with a reset phase of duration 600 ms (although this can be adjusted by the user), which we call *phase 2* or the *inter-trial interval*, when the external input is removed and the network model is allowed to return to its baseline activity, akin to an inter-trial interval.

A visualization of the decision phases is shown in Figure 4, where two different options, *right* and *left*, are considered. The blue trace represents the thalamic activity for the *right* channel, Th_{right}, while the orange trace represents that for the *left* channel, Th_{left}. At the end of *phase 0*, we can see that Th_{left} reaches the decision threshold of 30 spikes per second before Th_{right} has done so, resulting in a left choice being made. During phase 1, the Th_{left} activity is maintained around 30 spikes per second by the sustained activity in Cx_{left}.

In this task, critical attention should be paid to *phase 0*, as this represents the process of evidence accumulation where the striatal activity of both channels ramps until one of the thalamic populations' firing rates reaches the threshold of $30\,Hz$. To be largely consistent with commonly used experimental paradigms, the maximal duration of this phase is considered to be $\Delta_{max} = 1000\,ms$ such that, if the agent makes no decision within 1000 ms, the trial times out and the decision is marked as "none". These trials can be conveniently removed from the recorded data before analysis. If a decision is made, then the simulation proceeds as though a reward is delivered at the beginning of phase 1 – that is, during the motor sensory response – such that this phase represents the plasticity phase, where the choice selected in *phase 0* is reinforced with a dopaminergic signal. During this phase, the cortical population of the selected channel receives 75% of the maximum cortical stimulus applied during the ramping phase. This induces sustained activity that promotes activity-dependent plasticity as described in S2 Appendix. The activity-dependent plasticity rule strengthens (weakens) the corticostriatal weight to dSPNs (iSPNs) of the selected channel. CBGTPy allows for the specification of other parameters such as learning rate, maximum weight values for the corticostriatal projections and dopamine-related parameters (see details with examples in Section 2.3).

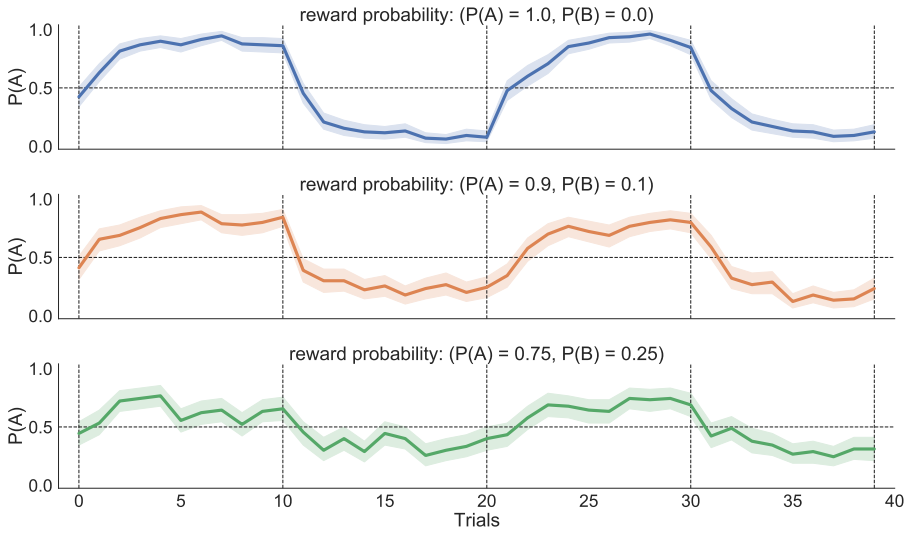

Figure 5: **Probability of choosing the more rewarded action (e.g., A or B) for different levels of conflict in a 2-choice task.** The reward contingencies flip between the two choices every 10 trials (marked by vertical dashed lines), at which point the probability of choosing the more rewarded option drops below chance. The probability of choosing A is high in 1st and 3rd block (where A is rewarded with varying levels of probabilities), however the probability to choose A drops in the 2nd and 4th block (where B is rewarded). Performance degrades in general as conflict increases, but sensitivity to change points drops. The performance was averaged over 50 random seeds for each conflict level.

In a baseline network, the selection probabilities are at chance level (i.e., 50% for a 2-choice task). However, as the network gets rewarded for its choices, the dopamine-dependent plasticity strengthens the corticostriatal projection to the dSPN population of each rewarded choice, thereby increasing the likelihood that it will be selected in the future. CBGTPy allows for probabilistic reward delivery associated with each option, as well as switching of these probabilities between the two actions (Figure 5). When such a change point occurs, the previously learned action now elicits a negative reward prediction error, forcing the network to unlearn the previously learned

choice and learn the new reward contingency.

3.2 A stop signal task

The stop signal task represents a common paradigm used in cognitive psychology and cognitive neuroscience for the study of reactive inhibition [47]. In this task, participants are trained to respond as fast as possible after the presentation of a "Go" cue. Sometimes the "Go" cue is followed by the presentation of a "Stop" cue, which instructs subjects to withhold their decision and hence, if successful, prevents any corresponding movement before it begins.

Imaging and electrophysiological studies in humans, rats, and monkeys agree in reporting that STN neurons become activated in response to a stop signal [14], providing a fast non-selective pause mechanism that contributes to action suppression, through the activation of the cortical hyperdirect pathway [37, 48]. However, this mechanism, by itself, mostly fails to inhibit locomotion, appearing to be not selective and long-lasting enough to prevent a late resurgence of the evidence accumulation process and thus to guarantee a complete cancellation of the execution of the motor response [36]. A complementary slower but selective mechanism is thought to be provided by the activation of arkypallidal neurons in the GPe in response to an external stimulus that instructs the network to brake the ongoing motor planning process [17, 36, 37]. According to this idea, a long-lasting action inhibition results from the activation of pallidostriatal GABAergic projections.

To reproduce these mechanisms and to simulate the interruption of the action selection process, we inject two independent external excitatory currents directly into STN and GPeA neurons during a typical CBGT simulation. The currents injected as a step function cause an increase in the firing rates of the target nuclei. Both of these external currents are defined using parameters that can be modified in an easy and user-friendly way, without requiring any familiarity or advanced knowledge of the details of the implementation (see Section 2.3 for further details and examples).

The following list of parameters characterizes each one of these currents:

(a) `amplitude`, specifying the magnitude of the stimulation applied;

(b) `population`, specifying the CBGT region or sub-population being stimulated. According to Mallet et al. [36], to reproduce reactive stopping, STN and GPeA are the nuclei that should be stimulated.

(c) `onset`, specifying the onset time of the stimulation, with respect to the trial onset time (i.e., the beginning of *phase 0*);

(d) `duration`, defining the duration of the stimulation. Since the length of *phase 0* is not fixed and is dependent on how long it takes for the thalamic firing rates to reach the decision threshold (30Hz), this parameter should be set carefully;

(e) `probability`, determining the fraction of trials on which the stop signal should be introduced;

(f) `channel`, defining which action channels are stimulated.

A more detailed description of all the parameters can be found in Section 2.3.2.

The characterization of the different network phases described in Section 2 slightly changes when performing the stop signal task (see Figure 6). During the decision phase (*phase 0*, which in this task lasts for a maximum of 300 ms) the stop signals are directly presented to the target populations by injecting independent external currents. The user can choose the moment of the injection by manipulating the variable `stop_signal_onset_time`. These signals are maintained active for a period equal to `stop_signal_duration`, with a magnitude equal to the `stop_signal_amplitude`. The values of all the aforementioned parameters do not need to be the same for all of the stop signals used. At this stage, two possible outcomes follow: (a) the presentation of the stop signals doesn't affect the network, which still manages to choose an action; or, (b) the network is not able to make an action after the presentation of the stop signal, and *phase 0* ends with no action triggered, which condition we named `stop`. The former option could be due to different reasons, amongst which the strength of the stop signal is not enough to

prevent the network from triggering an action or the evaluation process still has enough time to recover after the stop signal stimulation and the thalamic firing rates can reach the decision threshold (e.g., $30\,Hz$) within the permitted time of the decision window.

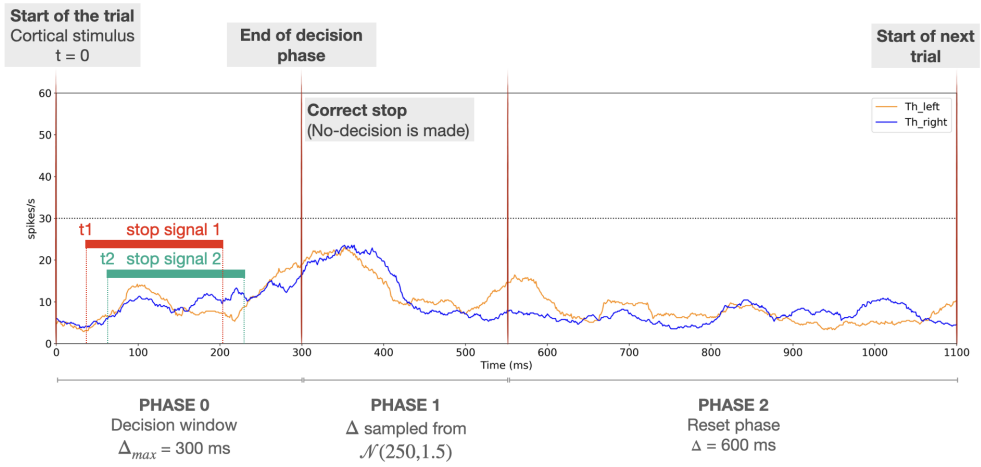

Figure 6: **Representation of the phases of the stop signal task with two action channels.** This sketch represents one trial of the simulation. The first red vertical thick line represents the presentation of the cortical stimulus (causing the cortical ramping), which indicates the start of the trial. Red and green horizontal bold lines depict the presentation of two stop currents, according to the onset time and duration values chosen by the user. In this example, the two stop signals are considered to be applied with different onset times t_1 and t_2, respectively. The second red vertical thick line depicts either the moment when an action has occurred (end of *phase 0*) or that $300\,ms$ has expired and no action has been triggered, so a successful stop has occurred. The third red vertical thick line depicts the end of the motor sensory response phase (end of *phase 1*), when an action is triggered anyway (failed stop). Here, the stop was successful because no action has been triggered (no decision threshold crossing within the decision window), so no motor sensory response is visible. It follows the start of the reset or inter-trial interval phase (*phase 2*), after which a new trial begins. Blue and orange traces represent the mean thalamic firing rates Th_x for $x \in \{\text{right,left}\}$, respectively, and the horizontal black dotted line highlights the decision threshold.

We show an example of stop signal stimulation applied to STN and GPeA populations, independently, in a 1-choice task (Figure 7). The onset of the stimulation applied to STN occurred at $30\,ms$ while that for the stimulation of GPeA was set to $60\,ms$; both signals were applied for a duration of $145\,ms$. Both stimulations were applied in all trials shown. Note that trial number 1 corresponds to a correct stop trial (no decision was made within *phase 0*), whereas the following trial corresponds to a failed stop trial. This is well reflected in the activity of GPi at the end of the decision making window, where the firing traces decrease more for failed stop trial rather than for correct stop.

3.3 Optogenetic stimulation

CBGTPy also allows for simulation of optogenetic stimulation of CBGT nuclei while an agent performs the above tasks (stop signal or n-choice). The optogenetic stimulation is implemented as an excitatory or inhibitory current to all the neurons of a nucleus or subpopulation during a typical CBGT simulation. The stimulation paradigm includes the following parameters:

(a) `amplitude`, the sign of which specifies the nature (positive→excitatory / negative→inhibitory) and the absolute value of which specifies the magnitude of the stimulation applied;

(b) `population`, specifying the CBGT region or sub-population being stimulated;

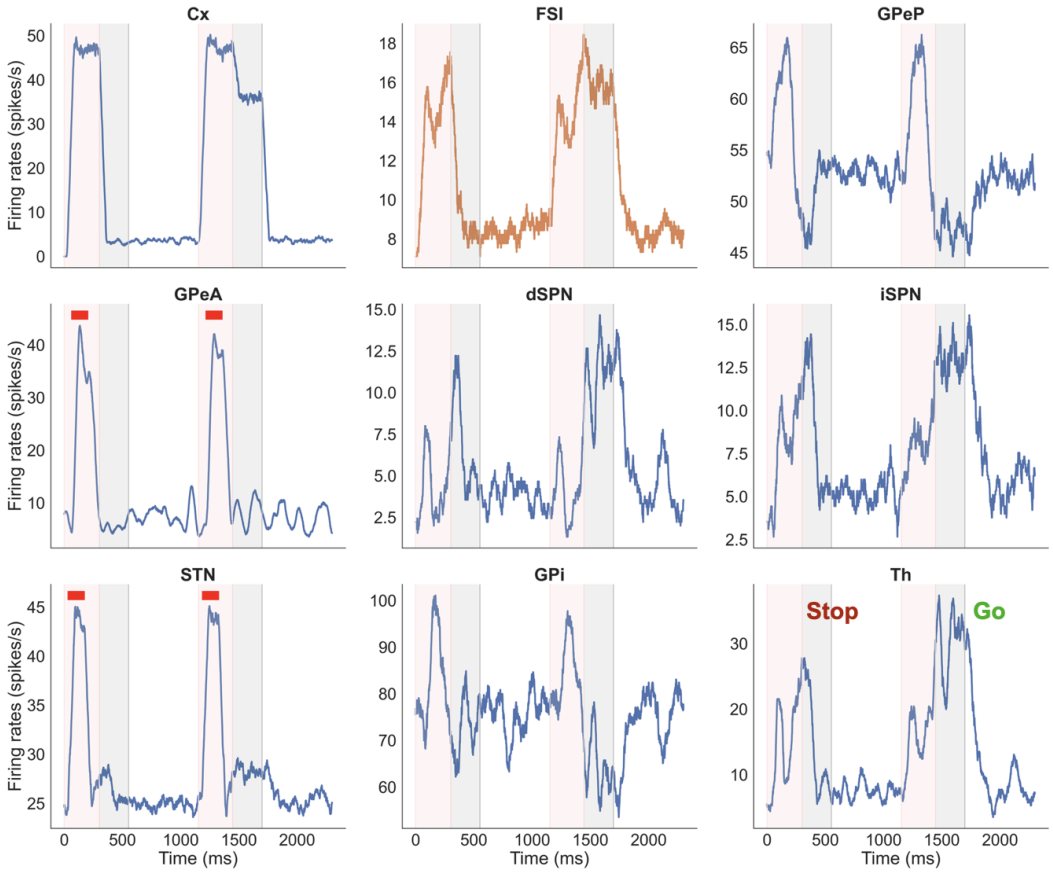

Figure 7: **Example figure showing firing rates for all nuclei for two consecutive stop trials.** Note that the simulation has been run in a 1-channel regime and two stop currents have been applied to STN and GPeA, respectively (see thick red bars). Segments of the simulation are color-coded to distinguish times associated with decision making (pink, phase 0) and subsequent times of sustained activity in the selected channel (grey, phase 1), when relevant, in each trial. The unshaded regions between each pair of trials are the inter-trial-interval (phase 2).

(c) `onset`, specifying the onset time of the stimulation;

(d) `duration`, defining the duration of the stimulation;

(e) `probability`, determines the fraction of trials to be stimulated or a list of trial numbers;

(f) `channel`, specifying which action channels are stimulated.

The parameter `population` should be entered as a list of the subpopulations to be stimulated. The parameter `onset` is calculated from the beginning of *phase 0*; for example, if this parameter is 10, then the optogenetic stimulation starts 10 ms after *phase 0* starts. The parameter `duration` controls the duration of the optogenetic stimulation. This parameter either accepts a numeric value in ms or string specifying which *phase* should be stimulated. The numeric value stimulates the list of selected populations from the specified onset until the specified duration. The string (e.g., "phase 0") ensure that the stimulation lasts until the end of specified phase, thereby allowing the user to specifically target the *decision*, *consolidation* or *inter-trial interval* phase. Please note, if an optogenetic configuration results in extending the duration of a phase (e.g., strongly inhibiting dSPN may delay *phase 0*), a time out is specified for every phase to prevent indefinite delay. The default timeouts for *phase* 0,1 and 2 are 1000 ms, 300 ms and 600 ms respectively unless specified by the user.

The parameter `probability` offers the flexibility of either assigning a fraction that determines the fraction of trials (randomly sampled from the full collection of trials) on which the stimulation is to be delivered or entering a list of specific trial numbers. Lastly, the parameter `channel` specifies the name of the action channel, such as "left", onto which the stimulation should be applied. This parameter also accepts two additional options, "all" or "any", the former of which leads to the application of a global stimulation to the same population in all channels and the latter randomly selects a channel each trial for stimulation.

We show an example of optogenetic stimulation applied to a list of iSPN and dSPN populations in a 3-choice task (Figure 8). The configuration for the stimulation was: 1) population: ["iSPN", "dSPN"] 2) amplitude: [0.1, -0.5] 3) duration: ["phase 0", 400.] 4) probability: [[0], [1]] 4) channels: ["all", "all"]. The excitatory stimulation (shown as thick blue bar), with an amplitude 0.1, was applied to the iSPN populations of all the channels (namely A, B, and C) in the 1st trial, for the duration of *phase 0*. An increased activity in the iSPN population (activation of the indirect pathway) causes a choice time out in the 1st trial. In the 2nd trial, an inhibitory stimulation for dSPN (shown as yellow bar) of amplitude -0.5 was applied for $400\,ms$ which results in brief but strong inhibition of dSPNs (direct pathway) thereby delaying the action selection. This can be observed by comparing the *decision* phase in the 3rd trial, where no such manipulation was imposed.

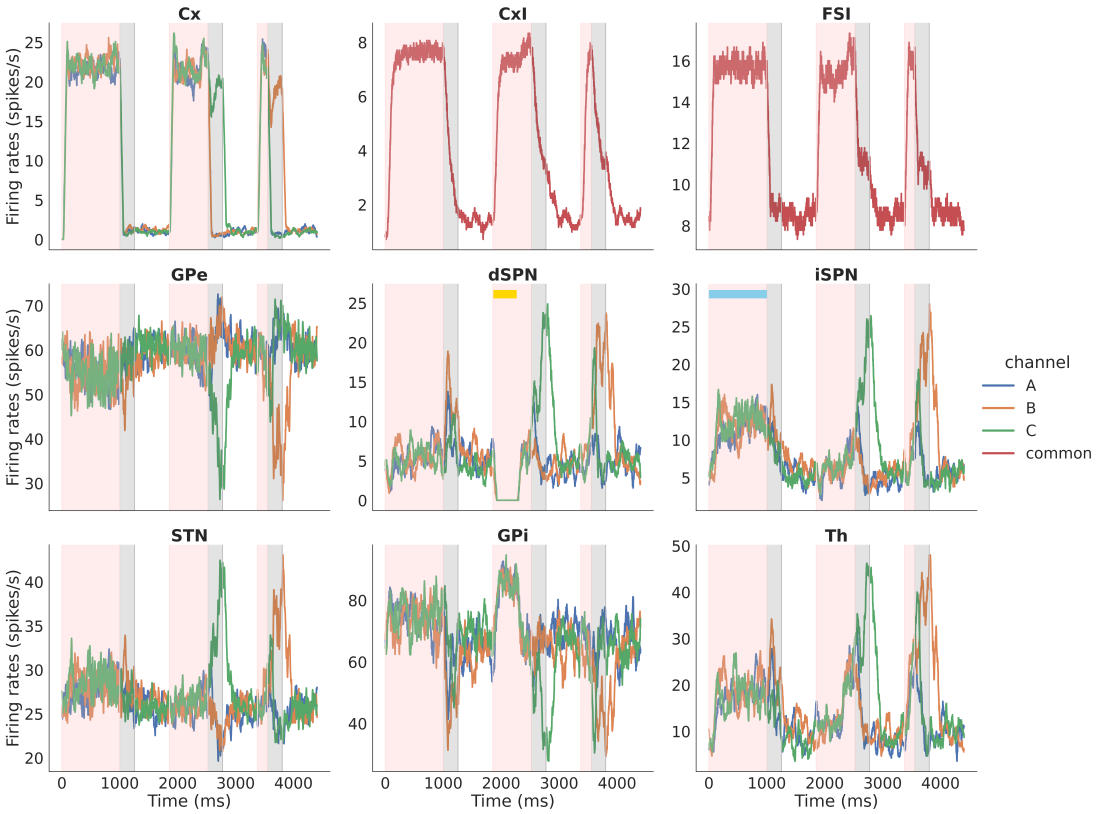

Figure 8: **Example figure showing optogenetic stimulation for the nuclei "iSPN" and "dSPN".** The configuration specified was: 1) amplitude: [0.1, -0.5] 2) duration: ["phase 0", 400.] 3) trial numbers: [[0], [1]] 4) channels: ["all", "all"]. The excitatory optogenetic stimulation given to iSPN is excitatory (shown as blue bar) and lasts all through the *phase 0*, whereas inhibitory stimulation to dSPN (shown as yellow bar) lasted for $400\,ms$. Both the stimulations were given to all the channels (namely A, B and C) of the nuclei.

4 Discussion

Here we introduced CBGTPy, an extensible generative modeling package that simulates responses in a variety of experimental testing environments with agent behavior driven by dynamics of the CBGT pathways of the mammalian brain. A primary strength of this package is the separation of the agent and environment components, such that modifications in the environmental paradigm can be made independent of the modifications in the CBGT network. This allows for creating predictions of both network function and behavior in a variety of experimental contexts that can be vetted against empirical observations. Moreover, various changes in the parameters of the network, as well as the experimental paradigm, can be made through the higher-level `configuration` variable that is sent as an argument in running the simulation, thereby avoiding a considerable coding effort on the part of the user. CBGTPy also returns behavioral outcomes (e.g., choices made and decision times) and "recordings" of neuronal outputs (instantaneous firing rates) for all the CBGT nuclei in the form of an easily usable and readable data frames. Taken together, CBGTPy allows for theorists and experimentalists alike to develop and test novel theories of the biological function of these critical pathways.

The individual components of CBGTPy are all designed to enable maximum flexibility. The basal ganglia model is constructed in an organized series of steps, beginning with high-level descriptions of the model and gradually achieving finer detail. Developing a modification to the network becomes a matter of inserting or modifying the appropriate steps, allowing high-level redesigns to be implemented as easily as more precise low-level modifications. CBGTPy's high degree of extensibility can, in large part, be attributed to its use of a data-flow programming paradigm. Neural pathways between major populations, for example, can be specified at a very high level, requiring only a single entry in the pathway table to describe how each subpopulation is connected. If the connectivity of a particular subpopulation, or even a particular neuron, needs adjustment, then the later steps in network construction can be responsible for those changes. CBGTPy was designed with this degree of flexibility to ensure that in the future, more complex biological models of the basal ganglia network can be developed and implemented in an efficient manner.

Of course, CBGTPy is not the only neural network model of these cortical-subcortical networks. Many other models exist that describe the circuit-level dynamics of CBGT pathways as both spiking [38, 49–53] and rate-based [21, 23, 54–60] systems. Indeed, for certain theoretical questions, CBGTPy may not be the optimal package for generating theoretical predictions. For example, rate-based network models (e.g., [54]) may be more computationally efficient at generating predictions of macroscale dynamics, such as those observed using fMRI or EEG, as well as simple behavioral predictions. Also, the properties of the cortical systems modeled in CBGTPy are simple and do not capture the nuanced connectivity and representational structure of real cortical systems. For these sorts of questions, there are many other modeling packages that would be better suited for generating hypotheses (e.g., [61]). Where CBGTPy thrives is in its a) biologically realistic description of subcortical pathways, b) scalability of adding in new pathways or network properties as they are discovered, c) flexibility at emulating a variety of typical neuroscientific testing environments, and d) ease of use for individuals with relatively limited programming experience. These benefits should make CBGTPy an ideal tool for many researchers interested in basal ganglia and thalamic pathways during behavior.

In summary, CBGTPy offers a simple way to start generating predictions of CBGT pathways in hypothesis-driven research. This tool enables researchers to run virtual experiments in parallel with *in vivo* experiments in both humans and non-human animals. The extensible nature of the tool makes it easy to be revised or expand in complexity as new observations come to light, making it a critical and evolving tool for understanding these pathways.

Acknowledgements

MC, JB, TV and JER are partly supported by NIH award R01DA053014 as part of the CRCNS program. CG and CV are supported by the PCI2020-112026 project, funded by MCIN/AEI/10.13039/501100011033 and by the European Union "NextGenerationEU"/PRTR as part of the CRCNS program.

Conflict of interest

The authors declare no conflict of interest.

References

[1] N. Kriegeskorte and P. K. Douglas, "Cognitive computational neuroscience," *Nature neuroscience*, vol. 21, no. 9, pp. 1148–1160, 2018.

[2] W. J. Ma and B. Peters, "A neural network walks into a lab: towards using deep nets as models for human behavior," *arXiv preprint arXiv:2005.02181*, 2020.

[3] O. Guest and A. E. Martin, "On logical inference over brains, behaviour, and artificial neural networks," *Computational Brain & Behavior*, pp. 1–15, 2023.

[4] J. S. Bowers, G. Malhotra, M. Dujmović, M. L. Montero, C. Tsvetkov, V. Biscione, G. Puebla, F. Adolfi, J. E. Hummel, R. F. Heaton *et al.*, "Deep problems with neural network models of human vision," *Behavioral and Brain Sciences*, pp. 1–74, 2022.

[5] D. L. Yamins and J. J. DiCarlo, "Using goal-driven deep learning models to understand sensory cortex," *Nature neuroscience*, vol. 19, no. 3, pp. 356–365, 2016.

[6] S. Herculano-Houzel, "The human brain in numbers: a linearly scaled-up primate brain," *Frontiers in human neuroscience*, p. 31, 2009.

[7] C. Vich, K. Dunovan, T. Verstynen, and J. Rubin, "Corticostriatal synaptic weight evolution in a two-alternative forced choice task: a computational study," *Communications in Nonlinear Science and Numerical Simulation*, vol. 82, p. 105048, 2020.

[8] V. S. Chakravarthy, D. Joseph, and R. S. Bapi, "What do the basal ganglia do? a modeling perspective," *Biological cybernetics*, vol. 103, pp. 237–253, 2010.

[9] S. Bariselli, W. Fobbs, M. Creed, and A. Kravitz, "A competitive model for striatal action selection," *Brain research*, vol. 1713, pp. 70–79, 2019.

[10] K. Dunovan and T. Verstynen, "Believer-Skeptic meets actor-critic: Rethinking the role of basal ganglia pathways during decision-making and reinforcement learning," *Frontiers in Neuroscience*, vol. 10, no. MAR, pp. 1–15, 2016.

[11] G. E. Alexander, M. R. DeLong, and P. L. Strick, "Parallel organization of functionally segregated circuits linking basal ganglia and cortex," *Annual review of neuroscience*, vol. 9, no. 1, pp. 357–381, 1986.

[12] R. L. Albin, A. B. Young, and J. B. Penney, "The functional anatomy of basal ganglia disorders," *Trends in neurosciences*, vol. 12, no. 10, pp. 366–375, 1989.

[13] J. W. Mink, "The basal ganglia: focused selection and inhibition of competing motor programs," *Progress in neurobiology*, vol. 50, no. 4, pp. 381–425, 1996.

[14] A. Nambu, H. Tokuno, and M. Takada, "Functional significance of the cortico–subthalamo–pallidal 'hyperdirect'pathway," *Neuroscience research*, vol. 43, no. 2, pp. 111–117, 2002.

[15] A. V. Kravitz, L. D. Tye, and A. C. Kreitzer, "Distinct roles for direct and indirect pathway striatal neurons in reinforcement," *Nature neuroscience*, vol. 15, no. 6, pp. 816–818, 2012.

[16] N. Mallet, B. R. Micklem, P. Henny, M. T. Brown, C. Williams, J. P. Bolam, K. C. Nakamura, and P. J. Magill, "Dichotomous organization of the external globus pallidus," *Neuron*, vol. 74, no. 6, pp. 1075–1086, 2012.

[17] A. Aristieta, M. Barresi, S. A. Lindi, G. Barriere, G. Courtand, B. de la Crompe, L. Guilhemsang, S. Gauthier, S. Fioramonti, J. Baufreton *et al.*, "A disynaptic circuit in the globus pallidus controls locomotion inhibition," *Current Biology*, vol. 31, no. 4, pp. 707–721, 2021.

[18] W. Maass, "On the computational power of winner-take-all," *Neural computation*, vol. 12, no. 11, pp. 2519–2535, 2000.

[19] J. C. Hedreen and M. R. Delong, "Organization of striatopallidal, striatonigral, and nigrostriatal projections in the macaque," *Journal of Comparative Neurology*, vol. 304, no. 4, pp. 569–595, 1991.

[20] C. R. Gerfen, T. M. Engber, L. C. Mahan, Z. Susel, T. N. Chase, F. J. Monsma Jr, and D. R. Sibley, "D1 and d2 dopamine receptor-regulated gene expression of striatonigral and striatopallidal neurons," *Science*, vol. 250, no. 4986, pp. 1429–1432, 1990.

[21] K. N. Gurney, M. D. Humphries, and P. Redgrave, "A New Framework for Cortico-Striatal Plasticity: Behavioural Theory Meets In Vitro Data at the Reinforcement-Action Interface," *PLoS Biology*, vol. 13, no. 1, p. e1002034, jan 2015. [Online]. Available: http://dx.plos.org/10.1371/journal.pbio.1002034

[22] W. Schultz, "Predictive reward signal of dopamine neurons," *Journal of Neurophysiology*, vol. 80, no. 1, pp. 1–27, 1998.

[23] R. Bogacz and T. Larsen, "Integration of reinforcement learning and optimal decision-making theories of the basal ganglia," *Neural Computation*, vol. 23, no. 4, pp. 817–851, 2011.

[24] M. J. Frank, L. C. Seeberger, and R. C. O'reilly, "By carrot or by stick: cognitive reinforcement learning in parkinsonism." *Science (New York, N.Y.)*, vol. 306, no. 5703, pp. 1940–3, dec 2004. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/15528409

[25] K. Morita and A. Kato, "Striatal dopamine ramping may indicate flexible reinforcement learning with forgetting in the cortico-basal ganglia circuits." *Frontiers in neural circuits*, vol. 8, no. April, p. 36, jan 2014. [Online]. Available: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3988379&tool=pmcentrez&rendertype=abstract

[26] G. Cui, S. B. Jun, X. Jin, M. D. Pham, S. S. Vogel, D. M. Lovinger, and R. M. Costa, "Concurrent activation of striatal direct and indirect pathways during action initiation," *Nature*, vol. 494, no. 7436, p. 238, 2013.

[27] F. Tecuapetla, S. Matias, G. P. Dugue, Z. F. Mainen, and R. M. Costa, "Balanced activity in basal ganglia projection pathways is critical for contraversive movements," *Nature communications*, vol. 5, p. 4315, 2014.

[28] J. H. Shin, D. Kim, and M. W. Jung, "Differential coding of reward and movement information in the dorsomedial striatal direct and indirect pathways," *Nature communications*, vol. 9, no. 1, pp. 1–14, 2018.

[29] J. G. Parker, J. D. Marshall, B. Ahanonu, Y.-W. Wu, T. H. Kim, B. F. Grewe, Y. Zhang, J. Z. Li, J. B. Ding, M. D. Ehlers *et al.*, "Diametric neural ensemble dynamics in parkinsonian and dyskinetic states," *Nature*, vol. 557, no. 7704, pp. 177–182, 2018.

[30] J. T. Dudman and J. W. Krakauer, "The basal ganglia: from motor commands to the control of vigor," *Current opinion in neurobiology*, vol. 37, pp. 158–166, 2016.

[31] R. S. Turner and M. Desmurget, "Basal ganglia contributions to motor control: a vigorous tutor," *Current opinion in neurobiology*, vol. 20, no. 6, pp. 704–716, 2010.

[32] P. E. Rueda-Orozco and D. Robbe, "The striatum multiplexes contextual and kinematic information to constrain motor habits execution," *Nature neuroscience*, vol. 18, no. 3, pp. 453–460, 2015.

[33] D. Thura and P. Cisek, "The basal ganglia do not select reach targets but control the urgency of commitment," *Neuron*, vol. 95, no. 5, pp. 1160–1170, 2017.

[34] E. A. Yttri and J. T. Dudman, "Opponent and bidirectional control of movement velocity in the basal ganglia," *Nature*, vol. 533, no. 7603, pp. 402–406, 2016.

[35] T. Hashimoto, C. M. Elder, M. S. Okun, S. K. Patrick, and J. L. Vitek, "Stimulation of the subthalamic nucleus changes the firing pattern of pallidal neurons," *Journal of neuroscience*, vol. 23, no. 5, pp. 1916–1923, 2003.

[36] N. Mallet, R. Schmidt, D. Leventhal, F. Chen, N. Amer, T. Boraud, and J. D. Berke, "Arkypallidal cells send a stop signal to striatum," *Neuron*, vol. 89, no. 2, pp. 308–316, 2016.

[37] R. Schmidt and J. D. Berke, "A pause-then-cancel model of stopping: evidence from basal ganglia neurophysiology," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 372, no. 1718, p. 20160202, 2017.

[38] L. Goenner, O. Maith, I. Koulouri, J. Baladron, and F. H. Hamker, "A spiking model of basal ganglia dynamics in stopping behavior supported by arkypallidal neurons," *European Journal of Neuroscience*, vol. 53, no. 7, pp. 2296–2321, 2021.

[39] K. Dunovan, C. Vich, M. Clapp, T. Verstynen, and J. Rubin, "Reward-driven changes in striatal pathway competition shape evidence evaluation in decision-making," *PLoS computational biology*, vol. 15, no. 5, p. e1006998, 2019.

[40] J. E. Rubin, C. Vich, M. Clapp, K. Noneman, and T. Verstynen, "The credit assignment problem in cortico-basal ganglia-thalamic networks: A review, a problem and a possible solution," *European Journal of Neuroscience*, vol. 53, no. 7, pp. 2234–2253, 2021.

[41] C. Vich, M. Clapp, J. E. Rubin, and T. Verstynen, "Identifying control ensembles for information processing within the cortico-basal ganglia-thalamic circuit," *PLOS Computational Biology*, vol. 18, no. 6, p. e1010255, 2022.

[42] K. Bond, J. Rasero, R. Madan, J. Bahuguna, J. Rubin, and T. Verstynen, "Competing neural representations of choice shape evidence accumulation in humans," *bioRxiv*, 2022. [Online]. Available: https://www.biorxiv.org/content/early/2022/11/23/2022.10.03.510668

[43] T. B. Sousa, "Dataflow programming concept, languages and applications," in *Doctoral Symposium on Informatics Engineering*, vol. 130, 2012.

[44] The Ray Team, "Ray 1.x architecture," Sept 2020. [Online]. Available: https://docs.ray.io/

[45] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman, Eds., 2010, pp. 56 – 61.

[46] P. Cisek and J. F. Kalaska, "Neural correlates of reaching decisions in dorsal premotor cortex: Specification of multiple direction choices and final selection of action," *Neuron*, vol. 45, no. 5, pp. 801–814, Mar 2005. [Online]. Available: https://doi.org/10.1016/j.neuron.2005.01.027

[47] F. Verbruggen, A. R. Aron, G. P. Band, C. Beste, P. G. Bissett, A. T. Brockett, J. W. Brown, S. R. Chamberlain, C. D. Chambers, H. Colonius *et al.*, "A consensus guide to capturing the ability to inhibit actions and impulsive behaviors in the stop-signal task," *elife*, vol. 8, p. e46323, 2019.

[48] R. Schmidt, D. K. Leventhal, N. Mallet, F. Chen, and J. D. Berke, "Canceling actions involves a race between basal ganglia pathways," *Nature neuroscience*, vol. 16, no. 8, pp. 1118–1124, 2013.

[49] M. Lindahl and J. Hellgren Kotaleski, "Untangling Basal Ganglia Network Dynamics and Function: Role of Dopamine Depletion and Inhibition Investigated in a Spiking Network Model," *eNeuro*, vol. 3, no. 6, 2016. [Online]. Available: http://www.eneuro.org/content/3/6/ENEURO.0156-16.2016

[50] M. D. Humphries, R. D. Stewart, and K. N. Gurney, "A physiologically plausible model of action selection and oscillatory activity in the basal ganglia." *The Journal of neuroscience : the official journal of the Society for Neuroscience*, vol. 26, no. 50, pp. 12 921–42, dec 2006. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/17167083

[51] O. Maith, F. Villagrasa Escudero, H. Ü. Dinkelbach, J. Baladron, A. Horn, F. Irmen, A. A. Kühn, and F. H. Hamker, "A computational model-based analysis of basal ganglia pathway changes in Parkinson's disease inferred from resting-state fMRI," *European Journal of Neuroscience*, vol. 53, no. 7, pp. 2278–2295, 2021.

[52] K. Chakravarty, S. Roy, A. Sinha, A. Nambu, S. Chiken, J. H. Kotaleski, and A. Kumar, "Transient Response of Basal Ganglia Network in Healthy and Low-Dopamine State," *eNeuro*, vol. 9, no. 2, 2022.

[53] A. Mandali, M. Rengaswamy, V. Srinivasa Chakravarthy, and A. A. Moustafa, "A spiking Basal Ganglia model of synchrony, exploration and decision making," *Frontiers in Neuroscience*, vol. 9, no. MAY, pp. 1–21, 2015.

[54] K. Gurney, T. J. Prescott, and P. Redgrave, "A computational model of action selection in the basal ganglia. I. A new functional anatomy," *Biological Cybernetics*, vol. 84, no. 6, pp. 401–410, 2001. [Online]. Available: http://link.springer.com/10.1007/PL00007984

[55] A. Leblois, T. Boraud, W. Meissner, H. Bergman, and D. Hansel, "Competition between feedback loops underlies normal and pathological dynamics in the basal ganglia." *Journal of Neuroscience*, vol. 26, no. 13, pp. 3567–3583, 2006. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/16571765

[56] S. J. van Albada and P. a. Robinson, "Mean-field modeling of the basal ganglia-thalamocortical system. I Firing rates in healthy and parkinsonian states." *Journal of theoretical biology*, vol. 257, no. 4, pp. 642–63, apr 2009. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/19168074

[57] J. Bahuguna, T. Tetzlaff, A. Kumar, J. H. Kotaleski, and A. Morrison, "Homologous basal ganglia network models in physiological and Parkinsonian conditions," *Frontiers in Computational Neuroscience*, vol. 11, no. August, pp. 1–21, 2017.

[58] B. Girard, J. Lienard, C. E. Gutierrez, B. Delord, and K. Doya, "A biologically constrained spiking neural network model of the primate basal ganglia with overlapping pathways exhibits action selection," *European Journal of Neuroscience*, vol. 53, no. 7, pp. 2254–2277, 2021.

[59] M. Guthrie, A. Leblois, A. Garenne, and T. Boraud, "Interaction between cognitive and motor cortico-basal ganglia loops during decision making: A computational study," *Journal of Neurophysiology*, vol. 109, no. 12, pp. 3025–3040, 2013.

[60] M. J. Frank, "Hold your horses: A dynamic computational role for the subthalamic nucleus in decision making," *Neural Networks*, vol. 19, no. 8, pp. 1120–1136, 2006.

[61] T. Bekolay, J. Bergstra, E. Hunsberger, T. DeWolf, T. C. Stewart, D. Rasmussen, X. Choo, A. R. Voelker, and C. Eliasmith, "Nengo: a python tool for building large-scale functional brain models," *Frontiers in neuroinformatics*, vol. 7, p. 48, 2014.

S1 Appendix CBGT model details

S2 Appendix Dopamine-dependent plasticity of corticostriatal weights

S3 Appendix CGBTpy installation and dependencies

S4 Appendix List of files

S5 Appendix Network scaling

S6 Appendix Supporting tables

S1 Table: **Neuronal parameters editable by the user.** These parameters can be modified through the data frame `params`.

S2 Table: **Population-specific neuron parameters changeable by the user.** These parameters can be modifiable through the dictionary textttpops, addressing the population of interest.

S3 Table: **Synaptic parameters changeable by the user.** These parameters can be modifiable through the data frame `receps`.

S4 Table: **Population-specific baseline parameters modifiable by the user.** These parameters can be modified through the dictionary `base`, addressing the population of interest.

S5 Table: **Dopamine-related parameters editable by the user.** These parameters can be modified through the data frame `dpmns`.

S6 Table: **Dopamine-related parameters for corticostriatal projections to D1- and D2-SPN neurons.** These parameters are modifiable through the data frames `d1` and `d2`, respectively.

S7 Table: **Parameters used for plasticity implementation.**

S7 Appendix Supporting Figures

S1 Fig: **List of parameters and data frames that are returned from the simulation.**

S2 Fig: **Example of `results['popfreqs']` data frame.**

S3 Fig: **Example of `datatables[0]` data frame.**

S4 Fig: **Example of `Q_df` data frame.**