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 Ciències 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 on 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 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 dynamics 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

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 and rectified linear unit gating, among others) 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*. Although there are a wealth of biologically realistic simulations of cortical and non-cortical pathways in the literature, 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 model systems that emulate 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 reproduces the structural and functional properties of CBGT circuits in a variety of experimental environments. 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 incorporates biologically-based aspects of the underlying network pathways and dynamics, a dopamine-dependent plasticity rule [6], and the capacity to mimic 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 usage on a variety of standard cognitive tasks, before turning to a discussion in Section 4. Various appendices (S1.2, S2 Appendix, S4 Appendix, S5 Appendix) present additional details about the CBGT model and CBGTPy toolbox, including the implementation of synaptic plasticity and a guide for CBGTPy installation.

Recent findings have suggested that the simple concepts of rigidly parallel feedforward basal ganglia (BG) pathways may be outdated [7, 8][1], 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.

[1]We use traditional terminology of "direct" and "indirect" pathways and SPNs (e.g, Figure 1. While we recognize that the idea of a unified indirect pathway is outdated, it is useful to maintain a term to refer to the complement of the direct projection from dSPNs to GPi and the ascending pallidostriatal connections.

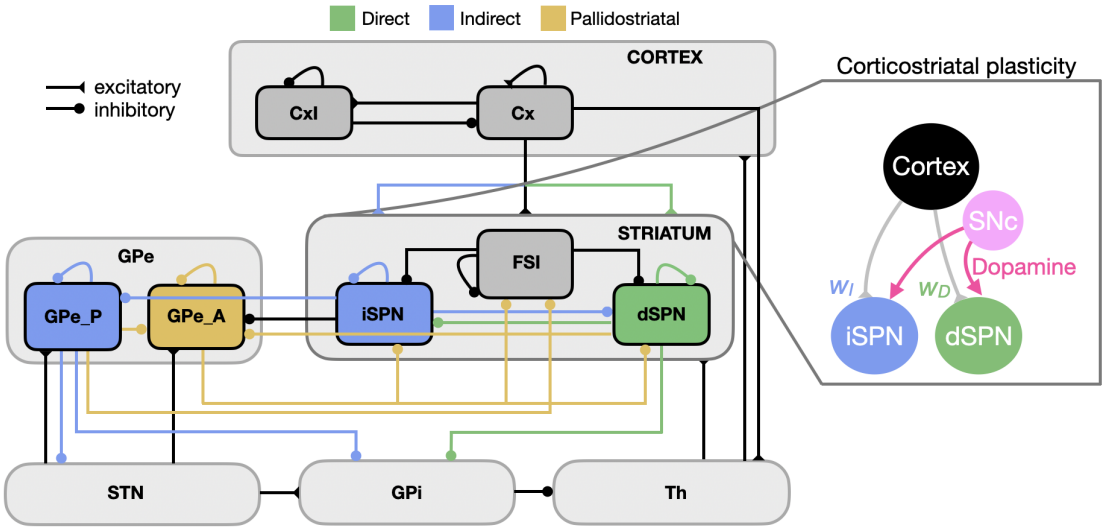


Figure 1: **Overview of the CBGT network.** The connectivity and cellular components of each CBGT channel in a CBGTPy agent are based on known biology. Direct pathway connections are shown in green, indirect pathway connections are shown in blue, and pallidostriatal connections are represented in gold, with arrows ending in circles marking the postsynaptic sites of inhibitory connections and those ending in triangles for excitatory connections. Dopaminergic feedback signals associated with rewards following actions induce plastic changes in corticostriatal synapses (pink arrows). A trial begins when the Cx population receives a stimulus and the model assumes a decision is made when the activity of the Th population reaches a certain threshold. In the current implementation of the network, the pallidostriatal pathways (gold colors) are only considered for the stop signal task.

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 [9–12]. 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 two subpopulations of spiny projection neuron (dSPNs involved in the so-called direct pathway, and iSPNs, involved in the indirect pathway) and also fast-spiking interneurons (FSI); external globus pallidus (GPe), which is divided into prototypical (GPeP) and arkypallidal (GPeA) subpopulations; subthalamic nucleus (STN); internal segment of globus pallidus (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 S1.2.

CBGTPy allows for the simulation of two general types of tasks that cover a variety of behavioral experiments used in neuroscience research. The first of these tasks is a discrete decision-making paradigm (*n-choice task*) in which the activity of the CBGT network results in the selection of one choice among a set of options (see Section 3.1). If plasticity is turned on during simulations, phasic dopamine, reflecting a reward prediction error, is released at the corticostriatal synapses and can 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 paradigm (*stop signal task*), where the network must control the execution or suppression of an action, following the onset of an imperative cue

(see Section 3.2). 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 across different values of parameters related to the stop signal.

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.

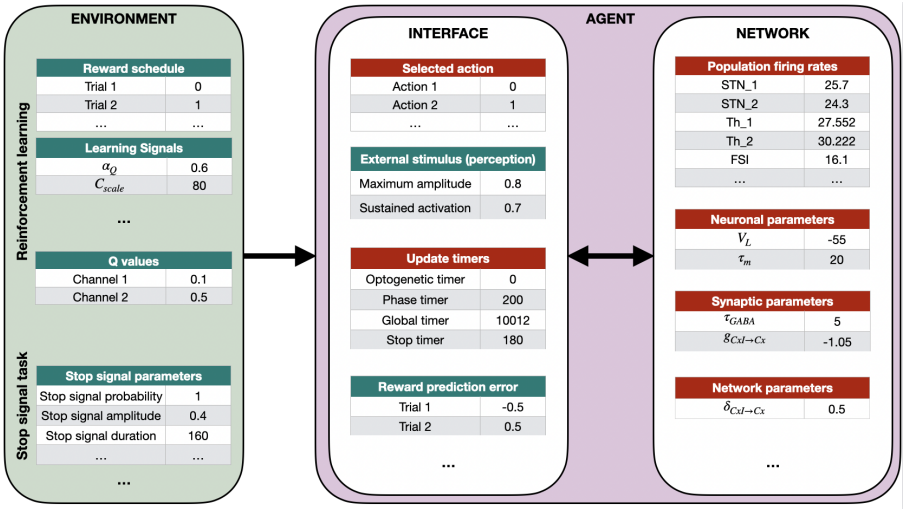


Figure 2: **Example of interactions between the environment and the agent (CBGT network).** The segregation of tasks between the environment and agent allows for independent modification of both. Tables with green title box depict some of the variables that can be easily modified by the user (see Section 2.3) while those in red are automatically updated or set internally. 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 is 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 evolves 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 [13]. 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 [14]. 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. The user, however, can choose to run the model without any multiprocessing library or an alternative multiprocessing library to Ray. These options are explained in detail in the Section 2.2.

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, specifically through use of 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), which can be found in the repository and include these steps and commands.

- Import all relevant functions.
- Create the main pipeline.
- Import the relevant `paramfile` for the selected experiment type.
- Create `configuration` dictionary with default values.
- Run the simulation, specifying which multiprocessing library to use.
- 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 if Ray multiprocessing is being used, 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.

Import relevant functions. All the relevant imports can be implemented 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`[2]. 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

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

```
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 or action channels the current instance of CBGTPy should create and run, using the variable `number_of_choices`:

```
number_of_choices = 2
```

While a common version of this 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)
```

Import the relevant `paramfile` for the selected experiment type. The `paramfile` contains dictionaries of default parameter values for the neural populations and plasticity model based on the choice of the experiment.

```
if experiment_choice == 'stop-signal':
    import stopsignal.paramfile_stopsignal as paramfile
elif experiment_choice == 'n-choice':
    import nchoice.paramfile_nchoice as paramfile
```

The imported attributes, which can be listed out using `dir(paramfile)`, can be modified according to the user's preferences. For example, setting the cellular capacitance value to 0.5 is accomplished with

```
paramfile.celldefaults['C'] = 0.5
```

Create `configuration` dictionary with default values. The `configuration` variable is a dictionary in which some parameters take internally set default values, whereas others 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 S7 Table, S9 Table, S10 Table, S11 Table, S12 Table, S13 Table and explained separately in Section 2.3.

```
configuration = {
    "experimentchoice": experiment_choice,
    "seed": 0,
    "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": paramfile.celldefaults,
    "pops": paramfile.popspecific,
    "receps": paramfile.receptordefaults,
    "base": paramfile.basestim,
    "dpmns": paramfile.dpmndefaults,
    "dSPN_params": paramfile.dSPNdefaults,
    "iSPN_params": paramfile.iSPNdefaults,
    "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": 0.7
}
```

Note that the parameter `corticostriatal_plasticity_present` does not have to be introduced in the configuration dictionary when running the stop signal task (see reference on the example notebook). Additionally, it is important to include the stop signal parameters within the `configuration` dictionary when executing the stop signal task.

```
configuration = {
    "stop_signal_present": [True, True],
    "stop_signal_probability": [1., 1.],
    "stop_signal_amplitude": [0.6, 0.6],
    "stop_signal_onset": [60.,60.],
    "stop_signal_duration": ["phase␣0",165.],
    "stop_signal_channel": ["all","left"],
    "stop_signal_population":["STN","GPeA"],
}
```

More details will be provided in the corresponding sections. For reference, you can find an example notebook on github.

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`). Moreover, the user can optionally specify one of the two supported multiprocessing libraries, Ray and Pathos, to use to run the simulation. Ray is a library providing a compute layer for parallel processing. To 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. To use Ray, the last step consists of setting the variable `use_library` in the notebook to `'ray'`. As an alternative to Ray, Pathos is a library that distributes processing across multiple nodes and provides other convenient improvements over Python's built-in tools. To use Pathos, no additional setup is required beyond setting the variable `use_library` to `'pathos'`. If the user does not want to use any of the above-mentioned libraries, this should be specified by setting the variable `use_library` to `'none'`. The simulation is performed by filling in values for these variables in the following command and executing it:

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

To ensure the simulation results are both reproducible and robust to minor changes in initial conditions, CBGTPy offers control over the pseudorandom number generator seed. The random seed controls the initial conditions of the network, including precisely which neurons connect together, under the constraint of the given connection probabilities and the baseline background activity levels 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 [15].

Extract relevant data frames Once the `results` object has been returned, specific variables and tables of interest can be extracted (see S2 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 S3 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 S4 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 the possible choices (see S5 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. The list of variables to be recorded should be specified in the configuration variable. In the example of the configuration variable used above, both `weight` and `optogenetic_input` are recorded.:

```
configuration = {
..
"record_variables": ["weight", "optogenetic_input"],
..
}
```

An example of plotting these data frames is included in the example python notebook in the GitHub repository. 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. Pickle files provide a method for saving complex Python data structures in a compact, binary format. 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 on every trial before indicating the result in the figure. The stop signal application is indicated as a bright horizontal red bar above firing traces of the the stimulated nuclei (e.g., Fig 7). The optogenetic stimulation is indicated as a blue bar for excitatory stimulation and yellow for inhibitory stimulation (e.g., Fig 8).

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 In case the Ray server was selected as the multiprocessing library to use to run the simulation, when the user is done working with the network, the Ray server can be shut down via the terminal 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 for modifications of several parameters that a user can easily perform. All 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

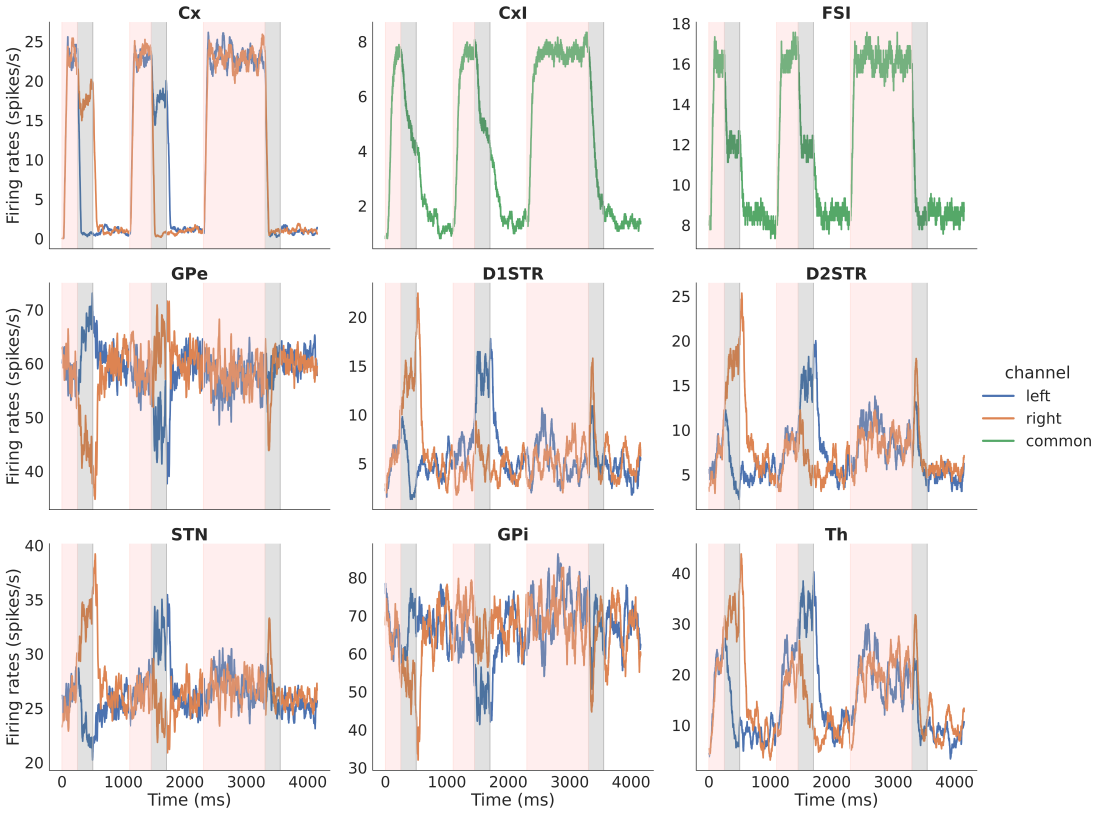


Figure 3: **Example figure showing firing rates for all the nuclei in the n-choice task**. Example figure showing firing rates for all nuclei for three consecutive trials of a 2-choice task, color-coded to distinguish times associated with decision making (pink, *decision phase*) and subsequent times 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. In this example, the model chooses right on the first trial and left on the second; in the third trial, where decision making times out, no sustained activation is applied to the cortical channel (top left subplot) during the *consolidation phase* (grey region).

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).

Through the adjustment of the appropriate parameters, the user can adapt the model to study a variety of important scientific questions. For example, one could introduce new neural pathways and vary their connectivity to study the effects on the system's dynamics and behavior (e.g., addition of a cortico-pallidal pathway, which may complement stopping mechanisms in the BG pathways [16]). Alternatively, the user could study the effects of specific neural parameters on decision-making and stop signal tasks. Additionally, by introducing optogenetic stimulation to different populations, such as dSPN and iSPN striatal populations, the user could model how the timing and intensity of this stimulation influences the plasticity and learning processes. For example, it has been shown that inhibition of dSPNs during learning impairs performance in a goal-directed learning task [17]. CBGTPy allows stimulation of multiple populations at different phases of a task, which enhances the options for exploring possible functional pathways and their roles in task performance. Lastly, CBGTPy allows many environmental parameters to be modified for an n-choice or stop-signal task while simulating the activity of the CBGT

network. For example, one could test the idea that the slowing down of decision times in healthy humans [18] when there is a high conflict or similarity between choices is related to increased activity in the STN [19]. Taken together, the large number of both network and environmental parameters available for the end user to control is a strength of the CBGTPy framework and greatly increases its ultimate scientific utility.

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 S7 Table. For example, the following dictionary entry can be modified to change 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`. These parameters will override the default values set by the `params` field. A complete list of editable parameters is given in S9 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 S10 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 equal to 1. Depending on the nature of the background input, the Gaussian process can be excitatory (AMPA and NMDA) 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 input parameters is given in S11 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 these editable parameters is given in S12 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 these editable parameters is given in S13 Table. In the following example, the learning rate (`dpmn_alphaw`) and the maximal value for the corticostriatal weights (`dpmn_wmax`) are specified:

```
"dSPN_params": pd.DataFrame([[39.5, 0.08]], columns=["dpmn_alphaw", "dpmn_wmax"]),
"iSPN_params": 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 of a cortico-pallidal pathway involving AMPA synapses with 50% connection probability, synaptic strength 0.01, and no plasticity is presented in the following dictionary entry:

```
"newpathways": pd.DataFrame([["Cx", "GPe", "AMPA", "syn", 0.5, 0.01, False]])
```

If the user wishes to change multiple pathways at once, then the variable can be given a list of data frames as input.

Q-learning process 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 process 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 dictionary entry:

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

The equations showing the roles 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 dictionary entry:

```
"Q_df_set": 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 consideration of possible choices. The maximum level of this input can be defined by the parameter `maxstim` and specified using the following dictionary 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 (for more details see S2 Appendix). 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 dictionary line:

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

Sustained activation to the action channel for the selected choice In order to resolve the temporal credit assignment problem [10], we rely on post-decision sustained activation to keep the selected channel active during the phasic dopaminergic activity [20, 21]. 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*); see Figure 3. 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 dictionary entry:

```
"sustainedfraction": 0.7,
```

Once the dopamine signal has been delivered at the end of the *consolidation phase*, 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 the *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 dictionary entry:

```
"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 70%) of the initial cortical input strength. As noted in the previous section, this phase of the simulation (*consolidation phase*) 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 $\mathcal{N}(250, 1.5)$. 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 changed 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 are as follows:

```
"n_trials": 2,

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

For more details about setting `volatility` parameter, please refer to the following paragraph.

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"],
```

Note that for a 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 $= 0$, as follows:

```
"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 `True`, with each boolean variable corresponding to each nucleus specified in the list of populations to be stimulated, as shown below:

```
"opt_signal_present": [True],
...
..
"opt_signal_population": ["dSPN"],
```

The above example shows the case, when only one population (i.e dSPNs) is stimulated. More than one population can be stimulated simultaneously as shown below:

```
"opt_signal_present": [True, True],
...
..
"opt_signal_population": ["dSPN", "iSPN"],
```

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. For example, the user wants all the trials in the simulation to be optogenetically stimulated, i.e `opt_signal_probability` = 1.0, it should be defined as shown below:

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

Alternatively, a specific list of trial numbers during which optogenetic stimulation should be applied can also be passed. An example is shown below:

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

In the above example, a list of trial numbers `[0, 1]` indicates that the optogentic stimulation is applied to trial numbers 0 and 1.

Please note that if more than one nucleus will be stimulated, the `opt_signal_probability` expects a list of floats (probabilities) or list of list (list of trial numbers). For example, as mentioned in an example above, say the user wants to stimulate two populations (dSPN and iSPN) at trial numbers [0,1] and [1,2] respectively:

```
"opt_signal_present": [True, True],
..
"opt_signal_population": ["dSPN", "iSPN"],
"opt_signal_probability": [[0, 1],[1, 2]],
```

Optogenetic signal amplitude The amplitude of the optogenetic signal can be passed as a list of floats 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],
```

If we want to send different amplitudes of optogenetic signals to different populations, for example, an excitatory (0.3) to dSPNs and inhibitory (-0.25) to iSPNs, then the amplitudes should be specified as :

```
"opt_signal_present": [True, True],
..
"opt_signal_population": ["dSPN", "iSPN"],
"opt_signal_amplitude": [0.3, -0.25],
```

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 the *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 numerical value in ms as well as 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 channel (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-shaped current.

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 first component of `record_variables` can be used to track the evolution of the weights from cortex to dSPNs or iSPNs during an n-choice experiment. The latter component records the optogenetic input applied to the target population and is especially useful 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. Note that, for the parameter `weight`, cortical weights to both dSPNs and iSPNs for all choice representations (channels) are recorded. 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"],
```

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.

```
"record_variables": ["stop_input"],
```

Stop signal If the experiment requires the stop signal to be applied, then this option should be selected in the `configuration` variable by setting `stop_signal_present` to `True`. Different stop signals can be applied to different target populations during the same execution. For these, the `stop_signal_present` variable is defined as a list whose length depends on the number of target populations. For example, if we apply stop signals to two different populations, we have to set this variable as follows:

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

Note that the user can apply the stop signals to as many nuclei as desired.

Stop signal populations The target populations for the stop signal 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 probability The stop signal probability can be specified using the parameter `stop_signal_probability`, which is a list whose entries can take a float or a list as input. If a float (between 0 and 1) is introduced, then this value represents the probability to which the stop signal is applied. These trials are picked randomly from the total number of trials. Alternatively, if a sublist is specified, then it must contain the numbers of those trials where the user wants the stop signal to be applied. Note that the entries in the main list refer to the populations specified in the same order as in the variable `stop_signal_populations`. For example, within the statement

```
"stop_signal_probability": [1.0, [2, 3, 6]],
```

the first float value represents a 100% probability of applying the "first" stop signal to the first specified nucleus (STN), while the subsequent list of values represents the numbers of the trials on which the "second" stop signal will be applied to the other target region (GPeA).

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. The onset time is measured with respect to the start of the *decision* phase. In the example proposed below, the stop signal stimulation at the STN starts $30\,ms$ after the *decision* phase begins, while that applied to the GPeA starts $60\,ms$ after the *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`. As in the previous cases, this is a list whose order must follow that of the target populations. In the following example, a stop signal lasting $100\,ms$ is applied to the STN while another with duration $160\,ms$ is applied to the GPeA:

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

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

```
"opt_signal_duration": ["phase␣0", "phase␣1"],
```

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

Stop signal channel The user can also control if the stop signal applied to a specific population is presented 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 parameter `stop_signal_channel`. In the following example, one stop signal is presented to the STN populations of all of the action channels, while the second stop signal is applied only to the GPeA population corresponding to the "left" action channel:

```
"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 An 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 probability associated with each choice can be abruptly changed as part of the experiment. Thus, there are two forms of environmental uncertainty 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. Higher conflict would represent a situation where the reward probabilities are more similar across choices, making detection of the optimal choice difficult, whereas a lower conflict represents highly disparate values of reward probabilities and easier detection of the optimal choice. Conflict is not specified directly in CBGTPy; 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 determines the number of trials before reward probabilities switch. The user can choose between whether the trials are switched after exactly λ trials or whether switches are determined probabilistically, in which case λ represents the rate parameter of a Poisson distribution that determines the number of trials before reward probabilities switch (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., an external stimulus, an internal process, or a combination of the two) 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.

To allow for selection between n different choices we instantiate n copies of all CBGT populations except FSI and CxI. This replication sets up action channels representing the available choices that can influence each other indirectly through the shared populations and otherwise remain separate over the whole CBGT loop. To distinguish between the firing ratse 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} for $n = 2$).

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

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

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

$$f_{ext,x}(t) = f_{ext,x}(t - dt) + I_{ramp}(t).$$

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 has duration sampled from a normal distribution $\mathcal{N}(\mu = 250ms,$

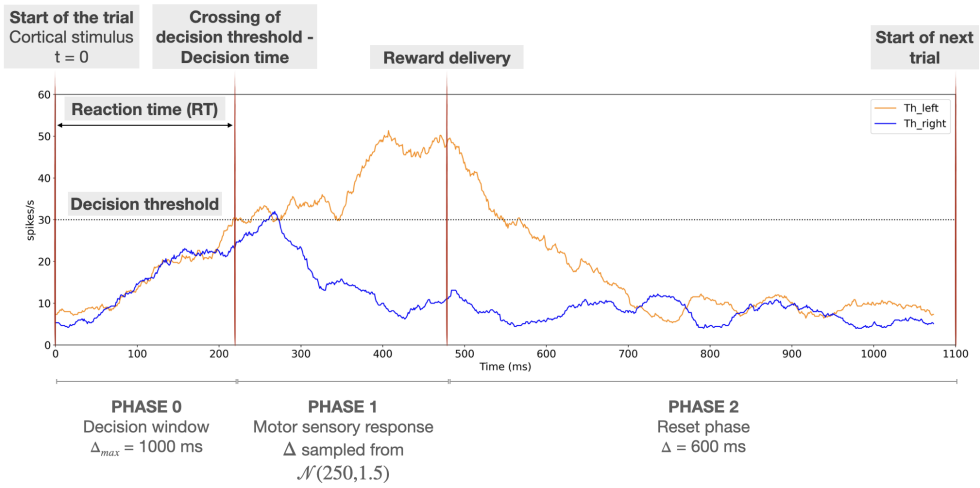


Figure 4: **Representation of the different phases of the simulated decision process.** This sketch represents the simulation of one trial of the 2-choice task in which a left choice is made. The first red vertical line, at time 0, represents the time of onset of a ramping stimulus to Cx_i for both i, 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 line depicts the end of the motor response period associated with the decision (end of *phase 1*), which is also the time when reward delivery occurs and hence dopamime level is updated. After this time, 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 $i \in$ {left, right}, respectively, and the horizontal black dotted line highlights the decision threshold.

$\sigma = 1.5ms$) 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 during *phase 1* [20] (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 $\Delta_{max}\, ms$, with a default value of 1000 ms), then no sustained activity is introduced in the cortex (see Figure 3, 3rd trial, 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* phase, 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 cortical input and striatal activity of both channels ramp 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 end of *phase 1* – that is, at the end of the motor sensory response – such that *phase 1* 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 70% of the maximum cortical stimulus applied during the ramping phase, although the user can change this percentage. This induces sustained activity that promotes dopamine- and 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 when dopamine rises above its baseline level. 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).

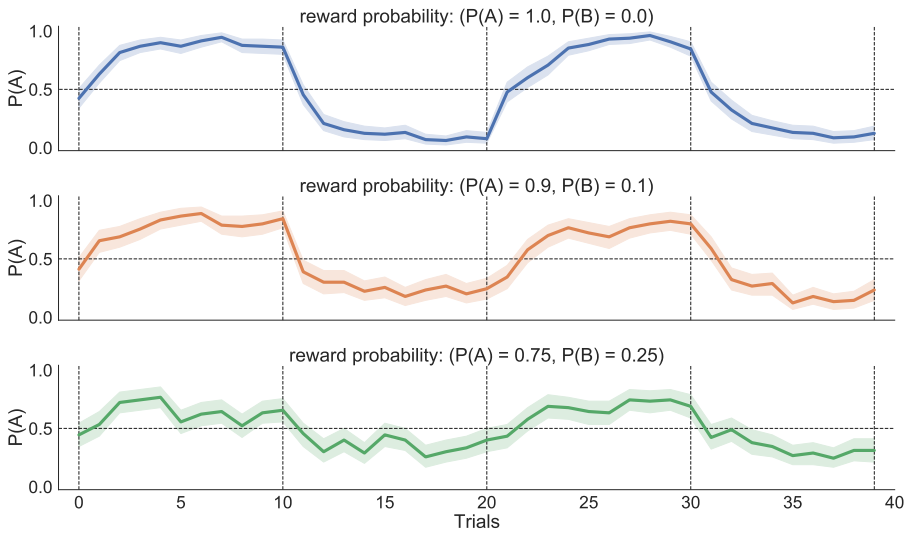


Figure 5: **Probability of choosing the more rewarded action (e.g., A or B) for different levels of conflict in a 2-choice task.** The reward contingencies flip between the two choices every 10 trials (marked by vertical dashed lines), at which point the probability of choosing the more rewarded option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

At the beginning of the simulation, with baseline network parameters, the selection probabilities are at chance level (i.e., 50% for a 2-choice task). If the network experiences rewards, however, 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 [22]. 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, rodents, and monkeys agree in reporting that STN neurons become activated in response to a stop signal [23], providing a fast, non-selective pause mechanism that contributes to action suppression through the activation of the

cortical hyperdirect pathway [24, 25]. 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 as needed to guarantee a complete cancellation of the execution of the motor response [26]. 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 [25–27]. According to this idea, a long-lasting action inhibition results from the activation of pallidostriatal GABAergic projections. For more details on how this mechanism takes place see [8].

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. This choice is based on the findings of Mallet et. al. (2016) [26]. 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.

(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 of these 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 kept active for a period equal to `stop_signal_duration`, with a magnitude equal to the `stop_signal_amplitude`. These values do not need to be the same for all of the stop signals used. At this stage, two possible outcomes follow: (a) despite the presentation of the stop signals, the network still manages to choose an action; or, (b) the network is not able to make an action after the presentation of the stop signals, and *phase 0* ends with no action triggered, which represents a `stop` outcome. The former option could arise for various reasons; the strength of the stop signal may not be sufficient to prevent the network from triggering an action or the evaluation process may still have enough time to recover, after the stop signal ends, to allowthe thalamic firing rates to reach the decision threshold (e.g., $30\,Hz$) within the permitted decision window.

In Figure 7 we show an example of stop signal stimulation applied to STN and GPeA populations, independently, in a 1-choice task. 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 both of the 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. These outcomes can be inferred from the activity of various populations at the end of the decision making windows: the thalamic firing rate reaches a higher level there and the firing traces in GPi decrease more for the failed stop trial than for correct stop, while cortical activity is sustained beyond this time specifically when stop fails.

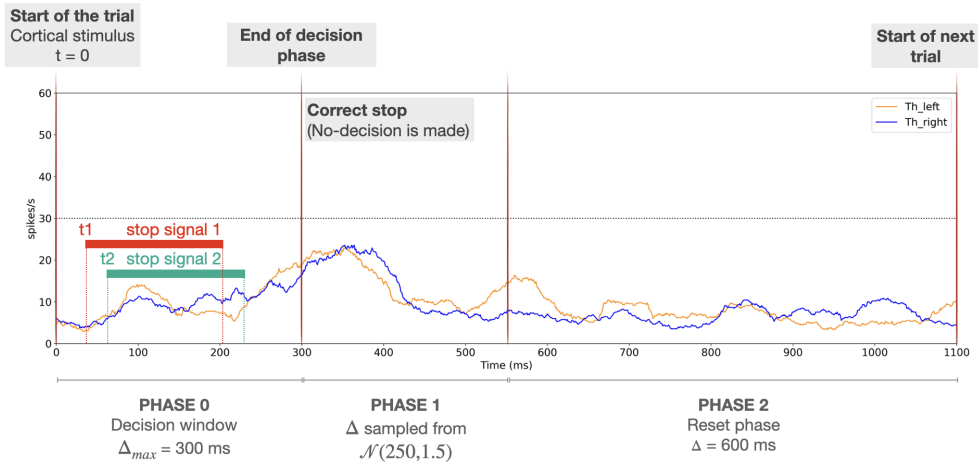


Figure 6: **Representation of the phases of the stop signal task with two action channels.** This sketch represents one trial of the simulation. The first red vertical line indicates the presentation of the cortical stimulus (causing ramping of cortical activity), which represents the start of the trial. Red and green horizontal bars 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 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 line depicts the end of the motor sensory response phase (end of *phase 1*), if an action is triggered (failed stop). Here, the stop was successful (no decision threshold crossing within the decision window), so no motor sensory response is visible. Finally, the *reset phase* (*phase 2*) occurs, 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 marks the decision threshold.

3.3 Optogenetic stimulation

CBGTPy also allows for simulation of optogenetic stimulation of CBGT nuclei while an agent performs the available tasks (stop signal or n-choice). The optogenetic stimulation is implemented as an excitatory or inhibitory current to all of 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;

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

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

(e) `probability`, indicating the fraction of trials or a list of trial numbers to include stimulation;

(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 a string specifying which *phase* should be stimulated. The numeric value stipulates that the list of selected populations will be stimulated from the specified onset time for the specified time duration. The string (e.g., "phase 0") stipulates that the stimulation should

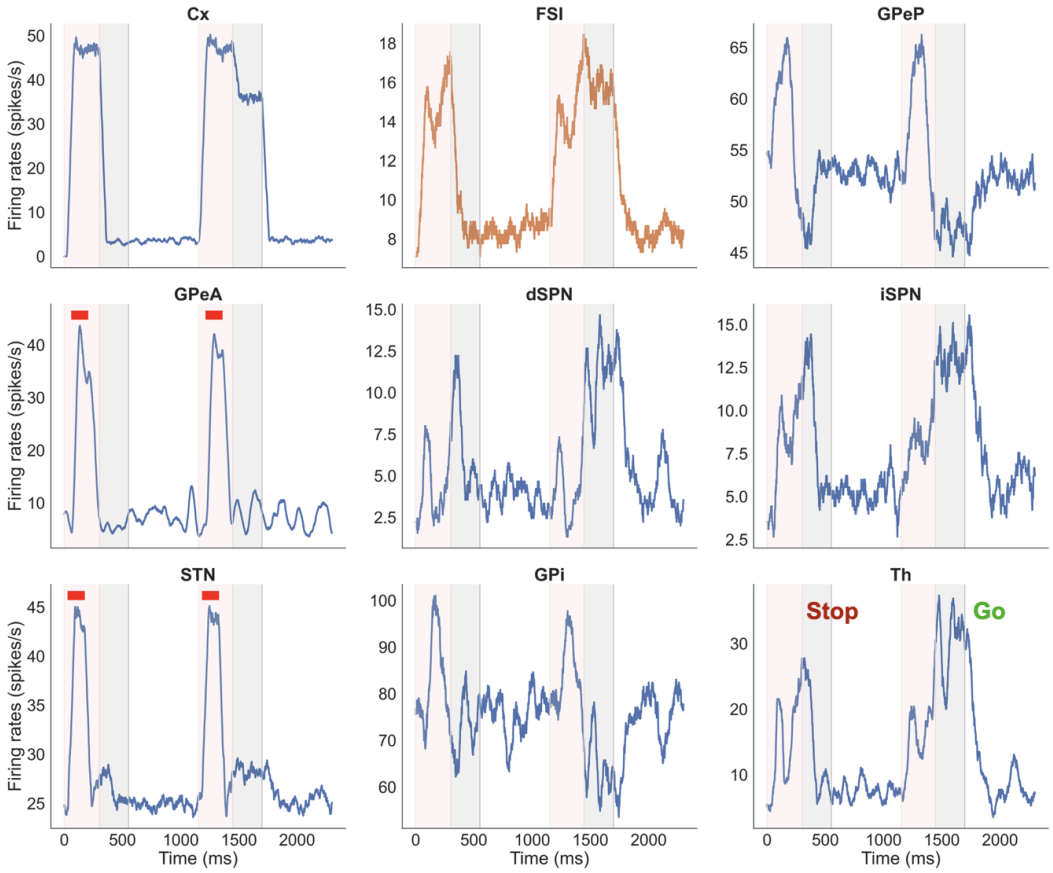


Figure 7: **Example figure showing firing rates for all nuclei for two consecutive stop trials.** Note that the simulation has been run in a 1-channel regime and two stop currents have been applied to STN and GPeA, respectively (see thick red bars). Segments of the simulation are color-coded to distinguish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

be applied throughout the specified phase, thereby allowing the user to specifically target the *decision*, *consolidation* or *inter-trial interval* phase. If an optogenetic configuration results in extending the duration of a phase (e.g., strongly inhibiting dSPN may extend *phase 0*), a time out is specified for every phase to prevent a failure to terminate the phase. 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 number that determines the fraction of trials (randomly sampled from the full collection of trials) on which the stimulation is to be delivered or else 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 of which randomly selects a channel for stimulation on each trial.

We show an example of optogenetic stimulation applied to a list of iSPN and dSPN populations in a 3-choice task (Figure 8). The excitatory stimulation (shown as thick blue bar), with an amplitude of 0.1, was applied to the iSPN populations of all the channels (namely A, B, and C) in the first trial, for the duration of *phase 0*. An increased activity in the iSPN population (activation of the indirect pathway) caused a choice time out on this trial. In the subsequent second trial, an inhibitory stimulation with an amplitude of -0.5 was applied to the dSPNs (shown as yellow

bar) for 400 ms. This resulted in brief but strong inhibition of dSPNs (direct pathway), thereby delaying the action selection. This can be observed by comparing the durations of the *decision phase between the second and third trials*, where no such manipulation was imposed.

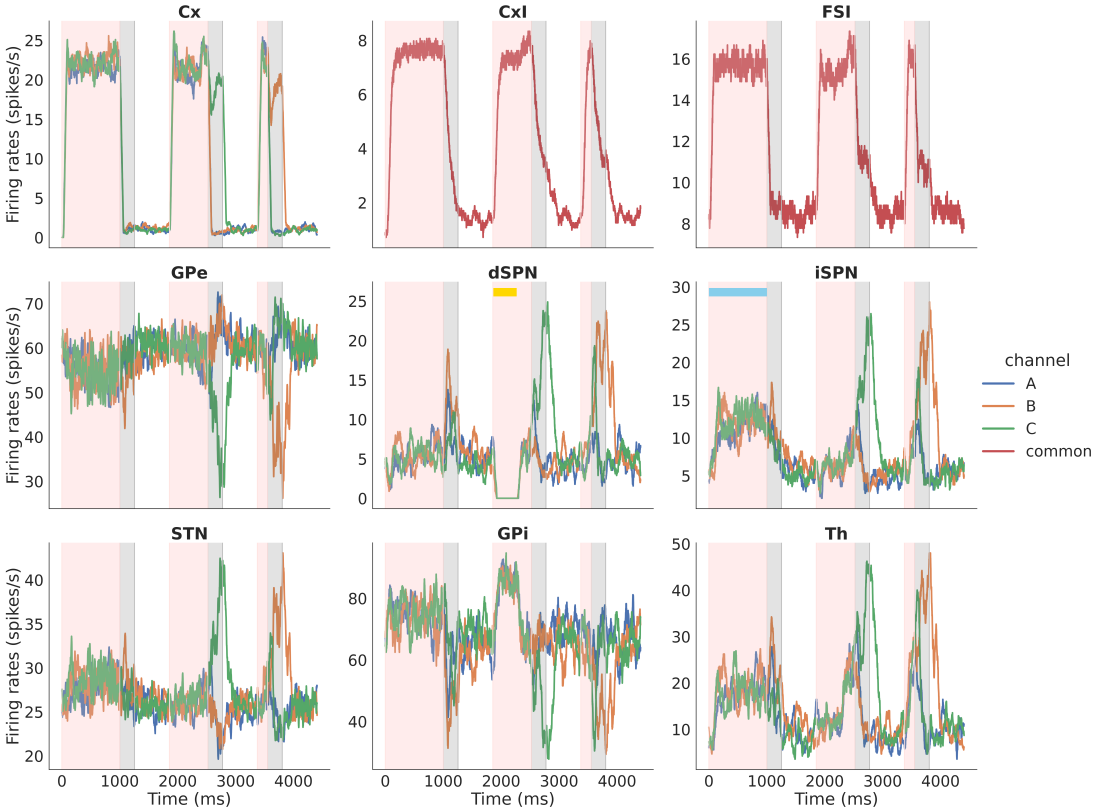


Figure 8: **Example figure showing optogenetic stimulation for the nuclei 'iSPN' and 'dSPN'.** The configuration specified was: `amplitude: [0.1, -0.5], duration: ['phase 0', 400], trial numbers: [[0], [1]], channels: ['all', 'all']`. The excitatory optogenetic stimulation given to iSPN (shown as blue bar) and lasts all through *phase 0*, whereas inhibitory stimulation to dSPN (shown as yellow bar) lasted for 400 ms. In both cases, stimulation were applied to all

channels (namely A, B and C) of the nuclei.

4 Discussion

Here we introduce 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 the user to derive predictions about network function and behavior in a variety of experimental contexts, which 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 of the CBGT nuclei in the form of easily usable and readable data frames. Overall, 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 providing more fine-grained details. Developing a modification to the network becomes a matter of inserting or modifying the appropriate components or 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 adjusted to implement those changes. CBGTPy was designed with this degree of flexibility to ensure that in the future, more complex biological models of the CBGT 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 either a spiking [16, 28–33] or a rate-based [34–42] system. CBGTPy has some limitations worth noting, such as not being as computationally efficient as rate-based models in generating macroscale dynamics, including those observed using fMRI or EEG, and associated predictions. Also, the properties of the cortical systems modeled in CBGTPy are quite 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., [43]). Where CBGTPy excels 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 about 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 introduce updates or expansions in complexity as new observations come to light, positioning it as a potentially important and highly useful tool for understanding these pathways.

Acknowledgements

We would like to thank all the member of exploratory intelligence group, especially Julia Badyna and Dr. Eric Yttri, for their helpful inputs. MC, JB, TV and JER are partly supported by NIH awards R01DA053014 and R01DA059993 as part of the CRCNS program. CG and CV are supported by the PCI2020-112026 project, and CV is also supported by the PCI2023-145982-2, both funded by MCIN/AEI/10.13039/501100011033 and by the European Union "NextGenerationEU"/PRTR as part of the CRCNS program. CG is also supported by the Conselleria de Fons Europeus, Universitat i Cultura del Govern de les Illes Balears under grant FPU2023-008-B.

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] 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.

[7] A. Nambu and S. Chiken, "External segment of the globus pallidus in health and disease: Its interactions with the striatum and subthalamic nucleus," *Neurobiology of Disease*, vol. 190, p. 106362, 2024.

[8] C. Giossi, J. Rubin, A. Gittis, T. Verstynen, and C. Vich, "Rethinking the external globus pallidus and information flow in cortico-basal ganglia-thalamic circuits," *Eur J Neurosci*, 2024.

[9] 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.

[10] 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.

[11] 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.

[12] 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

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

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

[15] 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.

[16] 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.

[17] P. Rothwell, S. Hayton, G. Sun, M. Fuccillo, B. Lim, and R. Malenka, "Input- and Output-Specific Regulation of Serial Order Performance by Corticostriatal Circuits," *Neuron*, vol. 88, no. 2, pp. 345–356, 2015. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0896627315008247

[18] M. J. Frank, J. Samanta, A. a. Moustafa, and S. J. Sherman, "Hold your horses: impulsivity, deep brain stimulation, and medication in parkinsonism." *Science (New York, N.Y.)*, vol. 318, no. 5854, pp. 1309–12, dec 2007. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/17962524

[19] K. a. Zaghloul, C. T. Weidemann, B. C. Lega, J. L. Jaggi, G. H. Baltuch, and M. J. Kahana, "Neuronal activity in the human subthalamic nucleus encodes decision conflict during action selection." *The Journal of neuroscience : the official journal of the Society for Neuroscience*, vol. 32, no. 7, pp. 2453–60, feb 2012. [Online]. Available: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3296967&tool=pmcentrez&rendertype=abstract

[20] 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

[21] S. Nonomura, K. Nishizawa, Y. Sakai, Y. Kawaguchi, S. Kato, M. Uchigashima, M. Watanabe, K. Yamanaka, K. Enomoto, S. Chiken *et al.*, "Monitoring and updating of action selection for goal-directed behavior through the striatal direct and indirect pathways," *Neuron*, vol. 99, no. 6, pp. 1302–1314, 2018.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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

[29] 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.

[30] S. Santaniello, M. M. McCarthy, E. B. Montgomery Jr, J. T. Gale, N. Kopell, and S. V. Sarma, "Therapeutic mechanisms of high-frequency stimulation in parkinson's disease and neural restoration via loop-based reinforcement," *Proceedings of the National Academy of Sciences*, vol. 112, no. 6, pp. E586–E595, 2015.

[31] 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

[32] 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.

[33] 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.

[34] 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

[35] 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.

[36] 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

[37] 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

[38] 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.

[39] 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.

[40] A. J. Nevado-Holgado, N. Mallet, P. J. Magill, and R. Bogacz, "Effective connectivity of the subthalamic nucleus - globus pallidus network during Parkinsonian oscillations." *The Journal of physiology*, pp. 1–12, mar 2014. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/24344162

[41] 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

[42] 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.

[43] 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.

[44] 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.

[45] 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.

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

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

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

[53] 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.

[54] 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.

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

[56] 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

[57] 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

[58] 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.

[59] 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.

[60] 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.

[61] 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.

[62] 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.

[63] 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.

[64] 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.

[65] 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.

[66] 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.

[67] 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.

[68] W. Wei, J. E. Rubin, and X.-J. Wang, "Role of the indirect pathway of the basal ganglia in perceptual decision making," *Journal of Neuroscience*, vol. 35, no. 9, pp. 4052–4064, 2015.

[69] G. D. Smith, C. L. Cox, S. M. Sherman, and J. Rinzel, "Fourier analysis of sinusoidally driven thalamocortical relay neurons and a minimal integrate-and-fire-or-burst model," *J. Neurophysiol.*, vol. 83, no. 1, pp. 588–610, 2000.

[70] M. Ketzef and G. Silberberg, "Differential synaptic input to external globus pallidus neuronal subpopulations in vivo," *Neuron*, vol. 109, no. 3, pp. 516–529, 2021.

[71] M. D. Bevan, P. A. Booth, S. A. Eaton, and J. P. Bolam, "Selective innervation of neostriatal interneurons by a subclass of neuron in the globus pallidus of the rat," *Journal of Neuroscience*, vol. 18, no. 22, pp. 9438–9452, 1998.

[72] A. Abdi, N. Mallet, F. Y. Mohamed, A. Sharott, P. D. Dodson, K. C. Nakamura, S. Suri, S. V. Avery, J. T. Larvin, F. N. Garas *et al.*, "Prototypic and arkypallidal neurons in the dopamine-intact external globus pallidus," *Journal of Neuroscience*, vol. 35, no. 17, pp. 6667–6688, 2015.

[73] A. Saunders, K. W. Huang, and B. L. Sabatini, "Globus pallidus externus neurons expressing parvalbumin interconnect the subthalamic nucleus and striatal interneurons," *PloS one*, vol. 11, no. 2, p. e0149798, 2016.

[74] V. M. Hernández, D. J. Hegeman, Q. Cui, D. A. Kelver, M. P. Fiske, K. E. Glajch, J. E. Pitt, T. Y. Huang, N. J. Justice, and C. S. Chan, "Parvalbumin+ neurons and npas1+ neurons are distinct neuron classes in the mouse external globus pallidus," *Journal of Neuroscience*, vol. 35, no. 34, pp. 11 830–11 847, 2015.

[75] P. D. Dodson, J. T. Larvin, J. M. Duffell, F. N. Garas, N. M. Doig, N. Kessaris, I. C. Duguid, R. Bogacz, S. J. Butt, and P. J. Magill, "Distinct developmental origins manifest in the specialized encoding of movement by adult neurons of the external globus pallidus," *Neuron*, vol. 86, no. 2, pp. 501–513, 2015.

[76] F. Fujiyama, T. Nakano, W. Matsuda, T. Furuta, J. Udagawa, and T. Kaneko, "A single-neuron tracing study of arkypallidal and prototypic neurons in healthy rats," *Brain Structure and Function*, vol. 221, pp. 4733–4740, 2016.

[77] K. J. Mastro, R. S. Bouchard, H. A. Holt, and A. H. Gittis, "Transgenic mouse lines subdivide external segment of the globus pallidus (gpe) neurons and reveal distinct gpe output pathways," *Journal of Neuroscience*, vol. 34, no. 6, pp. 2087–2099, 2014.

[78] K. E. Glajch, D. A. Kelver, D. J. Hegeman, Q. Cui, H. S. Xenias, E. C. Augustine, V. M. Hernández, N. Verma, T. Y. Huang, M. Luo *et al.*, "Npas1+ pallidal neurons target striatal projection neurons," *Journal of Neuroscience*, vol. 36, no. 20, pp. 5472–5488, 2016.

[79] V. L. Corbit, T. C. Whalen, K. T. Zitelli, S. Y. Crilly, J. E. Rubin, and A. H. Gittis, "Pallidostriatal projections promote β oscillations in a dopamine-depleted biophysical network model," *Journal of Neuroscience*, vol. 36, no. 20, pp. 5556–5571, 2016.

[80] A. Pamukcu, Q. Cui, H. S. Xenias, B. L. Berceau, E. C. Augustine, I. Fan, S. Chalasani, A. W. Hantman, T. N. Lerner, S. M. Boca *et al.*, "Parvalbumin+ and npas1+ pallidal neurons have distinct circuit topology and function," *Journal of Neuroscience*, vol. 40, no. 41, pp. 7855–7876, 2020.

S1 Appendix CBGT network

S1.1 Overview of 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 stable [**?**, 44–46]. Much of the theoretical work on this topic relates to a profoundly influential framework for representing the passage of top-down signals through the BG, 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 [23, 47–50] (Fig 1 in the manuscript). 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 traditional *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, rely both on prototypical (GPeP) and arkypallidal (GPeA) GPe cells [51] and are not considered part of the indirect pathway [27]. 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 [49]. Thus, the framework conceptualizes the basal ganglia as a collection of independent action channels [49], 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 [52]). 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 [23].

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 [53], where the dSPNs and iSPNs predominantly express D1 and D2 dopamine receptors, respectively [54]. Phasic dopamine signals differentially modulate D1 and D2 pathways in response to post-action feedback [41], 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) [55]. 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 [6, 38, 41, 56]; such a model has even been used to suggest the computational underpinning for the temporal profile of the dopamine signal [57].

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 [58–61]. 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 [62–66]. Continuous stimulation of the STN with deep brain stimulation in individuals with parkinsonism has been shown to fundamentally change BG output [67], impacting impulsiveness without compromising the selection of learned actions. Finally, the idea that hyperdirect pathway activation of the STN acts as a brake that can prevent 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 outputs and thalamic projections to the striatum), have been recognized as playing critical roles in stopping [16, 25–27].

S1.2 CBGT model details

The total number of neurons per population is provided in S1 Table.

Population	CxI	Cx	$dSPN$	$iSPN$	FSI	GPe_A	GPe_P	GPi	STN	Th
Number of neurons	186	204	75	75	75	190	560	75	750	75

S1 Table: Number of neurons considered in each population. When no distinction between GPe_A and GPe_P is considered, the total number of neurons at GPe is the sum of arkypallidals and prototypicals (750).

As in previous works [9, 11, 68], the activity of each neuron is described by the integrate-and-fire-or-burst model [69], with equations given by

$$\begin{aligned} C\frac{dV}{dt} &= -g_L(V(t)-V_L) - g_T h(t) H(V(t)-V_h)(V(t)-V_T) - I_{syn}(t) + I_{ext}(t), \\ \frac{dh}{dt} &= \begin{cases} -h(t)/\tau_h^- & \text{when } V \geq V_h \\ -(1-h(t))/\tau_h^+ & \text{when } V < V_h \end{cases} \end{aligned} \tag{1}$$

where $V(t)$ denotes the activity of the membrane potential at time t. The equation describing the evolution of the membrane potential (dV/dt) contains the leak current, with constant conductance g_L and reversal potential V_L; the low-threshold Ca^{2+} current, with constant conductance g_T, gating variable $h(t)$, and reversal potential V_T; the synaptic current $I_{syn(t)}$; and, finally, the external current $I_{ext}(t)$. Parameter C stands for the capacitance of the membrane potential. The evolution (dh/dt) of the gating variable $h(t)$ changes according to a the relation of V to a constant voltage threshold for burst activation, V_h, where τ_h^+ and τ_h^- represent, respectively, the decay time constant when the membrane potential is below or above V_h . Finally, $H(\cdot)$ represents the Heaviside step function. As with all integrate-and-fire models, a reset condition is added to the model to control the spike generation such that if $V(t)$ crosses a certain threshold value V_{th}, then the membrane potential is reset to ta hyperpolarized membrane potential V_{re}, simulating the spike onset time. That is, if $V(t^-) > V_{th}$, then $V(t^+) = V_{re}$ and a spike has been made by the specific neuron. Parameters of the neuronal model are provided in S2 Table.

All neurons in the network communicate through the simulated release of neurotransmitters across synapses. When action potentials arrive at postsynaptic neurons, the activity of each neuron's AMPA, GABA, and NMDA receptors increases according to the synaptic weight and neurotransmitter type. The activation of these receptors induces cellular currents which, in turn, drive future action potentials. The synaptic current $I_{syn}(t)$ is therefore modeled as

$$\begin{aligned} I_{syn}(t) = \quad & g_{AMPA} s_{AMPA}(t)(V(t)-V_E) + \frac{g_{NMDA} s_{NMDA}(t)}{1+e^{-0.062V(t)/3.57}}(V(t)-V_E) \\ & + g_{GABA} s_{GABA}(t)(V(t)-V_I) \end{aligned}$$

where g_x, for $x \in \{AMPA, NMDA, GABA\}$, stands for the maximal conductance in each channel; V_E and V_I are the excitatory and inhibitory reversal potentials, respectively; and finally, the variable $s_x(t)$, for $x \in \{AMPA, NMDA, GABA\}$, corresponds to the fraction of open

Population	$C\,(cm^2)$	$g_L\,(mS/cm^2)$	$g_T\,(mS/cm^2)$	$V_L\,(mV)$	$V_h\,(mV)$	$V_T\,(mV)$	$V_{th}\,(mV)$	$V_{re}\,(mV)$	$\tau_h^-\,(ms)$	$\tau_h^+\,(ms)$
CxI	1	1/10	0	−55	−60	120	−50	−55		
Cx	1	1/20	0	−55	−60	120	−50	−55	20	100
$dSPN$	1	1/20	0	−55	−60	120	−50	−55	20	100
$iSPN$	1	1/20	0	−55	−60	120	−50	−55	20	100
FSI	1	1/10	0	−55	−60	120	−50	−55	20	100
$^*GPe$	1	1/20	0.06	−55	−60	120	−50	−55	20	100
GPe_A	1	1/20	0.06	−55	−60	120	−50	−55	20	100
GPe_P	1	1/20	0.06	−55	−60	120	−50	−55	20	100
GPi	1	1/20	0	−55	−60	120	−50	−55	20	100
STN	1	1/20	0.06	−55	−60	120	−50	−55	20	100
Th	1	1/27.78	0	−55	−60	120	−50	−55	20	100

S2 Table: **Neuronal parameters.** Parameters used in the integrate-and-fire-or-burst model (see equation (1)) where C is the membrane capacitance and coincide with the inverse of the membrane time constant, g_L is the leak conductance, g_T is the low threshold Ca^{2+} maximal conductance, V_L is the leak reversal potential, V_h is the threshold potential for the burst activation, V_T is the low threshold Ca^{2+} reversal potential, τ_h^- is the burst duration, and τ_h^+ is the hyperpolarization duration. * Values in this row are the ones used when no intrinsic separation of neurons is considered.

channels of each type. The latter variables evolve according to the differential equations

$$\frac{ds_{AMPA}}{dt} = \sum_j \delta(t-t_j) - \frac{s_{AMPA}}{\tau_{AMPA}},$$
$$\frac{ds_{NMDA}}{dt} = \alpha(1-s_{NMDA})\sum_j \delta(t-t_j) - \frac{s_{NMDA}}{\tau_{NMDA}},$$
$$\frac{ds_{GABA}}{dt} = \sum_j \delta(t-t_j) - \frac{s_{GABA}}{\tau_{GABA}}$$

where t_j stands for the $j-$th spike onset time; α is a constant rate; τ_x, for $x \in \{AMPA,\, NMDA,\, GABA\}$, is the decay time constant of the corresponding s_x. The term $(1-s_{NMDA})$ has been designed in order to prevent s_{NMDA} from exceeding the value of 1. Finally, $\delta(\cdot)$ stands for the Dirac delta function.

Individual neurons within the same population are connected to each other with a population-specific probability (p_x) and connection strength (weights, w_x), such that the maximal conductance for a specific receptor x, for $x \in \{AMPA,\, NMDA,\, GABA\}$, is given by $g_x = p_x w_x$. Connections between populations similarly are characterized by probabilities and strengths. These connections are depicted with arrows in Fig 1 in the manuscript. The direct (green connections), indirect (blue connections), and pallidostriatal (yellow connections) pathways are present. Depending on the type of receptors, these connections can be inhibitory (arrows ending in a circle) or excitatory (arrows ending in a triangle). Parameters used for the connectivity are provided in S3 Table.

All populations have an external current I_{ext} to tune their baseline firing rate, given by

$$I_{ext}(t) = S_{ext,AMPA}(V(t)-V_E) + S_{ext,GABA}(V(t)-V_I)$$

where $S_{ext,x}$ for $x \in \{AMPA,\, GABA\}$ is a mean-reverting random walk derived from the stochastic differential equation

$$dS_{ext,x} = \frac{(\mu_{ext,x} - S_{ext,x})}{\tau_x}dt + \sigma_{ext,x}\sqrt{\frac{2}{\tau_x}}dW_t.$$

Here, W_t is a Wiener process, τ_x is the time decay of the external current, and $\mu_{ext,x}$ and $\sigma_{ext,x}$ are computed as

$$\mu_{ext,x} = 0.001 E_{ext,x} f_{ext,x} N_{ext,x} \tau_x,$$
$$\sigma_{ext,x} = E_{ext,x}\sqrt{0.0005 f_{ext,x} N_{ext,x} \tau_x}.$$

The parameter $f_{ext,x}$ is the external input frequency, $E_{ext,x}$ is the mean efficacy of the external connections, $N_{ext,x}$ is the number of connections, and τ_x the time decay constant. Values of all of these parameters are specified in S3 Table and S4 Table.

all pathways network				direct/indirect pathways network			
Connected populations	Receptor type	Connection Probability	Connection strength	Connected populations	Receptor type	Connection Probability	Connection strength
$CxI - CxI$	GABA	1	1.075	$CxI - CxI$	GABA	1.0	1.075
$CxI - Cx$	GABA	0.5	1.05	$CxI - Cx$	GABA	0.5	1.05
$Cx - Cx$	AMPA	0.13	0.0127	$Cx - Cx$	AMPA	0.13	0.0127
	NMDA	0.13	0.1		NMDA	0.13	0.08
$Cx - CxI$	AMPA	0.0725	0.113	$Cx - CxI$	AMPA	0.0725	0.113
	NMDA	0.0725	0.525		NMDA	0.0725	0.525
$Cx - dSPN$	AMPA	1	0.022	$Cx - dSPN$	AMPA	1.0	0.015
	NMDA	1	0.03,		NMDA	1.0	0.02
$Cx - iSPN$	AMPA	1	0.022	$Cx - iSPN$	AMPA	1.0	0.015
	NMDA	1	0.028		NMDA	1.0	0.02
$Cx - FSI$	AMPA	1	0.085	$Cx - FSI$	AMPA	1.0	0.19
$Cx - Th$	AMPA	1	0.025	$Cx - Th$	AMPA	1.0	0.025
	NMDA	1	0.029		NMDA	1.0	0.029
$dSPN - dSPN$	GABA	0.45	0.28	$dSPN - dSPN$	GABA	0.45	0.28
$dSPN - iSPN$	GABA	0.45	0.28	$dSPN - iSPN$	GABA	0.45	0.28
$dSPN - GPi$	GABA	1	1.8	$dSPN - GPi$	GABA	1.0	2.09
$dSPN - GPe_A$ [70]	GABA	0.4	0.054				
$iSPN - iSPN$	GABA	0.45	0.28	$iSPN - iSPN$	GABA	0.45	0.28
$iSPN - dSPN$	GABA	0.5	0.28	$iSPN - dSPN$	GABA	0.5	0.28
$iSPN - GPe_A$ [27, 70]	GABA	0.4	0.61	$iSPN - GPe$	GABA	1.0	4.07
$iSPN - GPe_P$ [27, 70, 71]	GABA	1	4.07				
$FSI - FSI$	GABA	1	2.7	$FSI - FSI$	GABA	1.0	3.25833
$FSI - dSPN$	GABA	1	1.25	$FSI - dSPN$	GABA	1.0	1.2
$FSI - iSPN$	GABA	1	1.15	$FSI - iSPN$	GABA	1.0	1.1
$GPe_A - GPe_A$	GABA	0.4	0.15				
$GPe_A - iSPN$ [51, 70, 72–74]	GABA	0.4	0.12				
$GPe_A - dSPN$ [51, 70, 72–74]	GABA	0.4	0.32				
$GPe_A - FSI$ [51, 70, 72–74]	GABA	0.4	0.01				
$GPe_P - GPe_P$	GABA	0.4	0.45	$GPe - GPe$	GABA	0.0667	1.75
$GPe_P - GPe_A$ [27, 40, 70, 75, 76]	GABA	0.5	0.3				
$GPe_P - STN$ [71–73, 77, 78]	GABA	0.1	0.37	$GPe - STN$	GABA	0.0667	0.35
$GPe_P - GPi$ [71–73, 77, 78]	GABA	1	0.058	$GPe - GPi$	GABA	1.0	0.058
$GPe_P - FSI$ [51, 72, 73, 78, 79]	GABA	0.4	0.1				
$STN - GPe_P$ [27, 70, 80]	AMPA	0.161666	0.10	$STN - GPe$	AMPA	0.161666	0.07
	NMDA	0.161666	1.51		NMDA	0.161666	1.51
$STN - GPe_A$ [27, 70, 80]	AMPA	0.161666	0.026				
	NMDA	0.161666	0.075				
$STN - GPi$	AMPA	1	0.0325	$STN - GPi$	AMPA	1.0	0.038
$GPi - Th$	GABA	1	0.3315	$GPi - Th$	GABA	1.0	0.3315
$Th - dSPN$	AMPA	1	0.3825	$Th - dSPN$	AMPA	1.0	0.3825
$Th - iSPN$	AMPA	1	0.3825	$Th - iSPN$	AMPA	1.0	0.3825
$Th - FSI$	AMPA	0.8334	0.113	$Th - FSI$	AMPA	0.8334	0.1
$Th - Cx$	AMPA	0.8334	0.03	$Th - Cx$	AMPA	0.8334	0.03
$Th - CxI$	AMPA	0.8334	0.015	$Th - CxI$	AMPA	0.8334	0.015

S3 Table: **CBGT connectivity parameters.** Two blocks of 4 columns each are depicted. The first block contains information regarding the network when the 4 different pathways are simulated (see Fig 1 in the manuscript), while the second block contains information regarding the network when only direct/indirect pathways are simulated. Columns in each block describe the parameters used to compute, in each population (1st columns), the maximal conductances g_x, for $x \in \{AMPA, NMDA, GABA\}$ (2nd column), which is the product of the probability of connectivity (3rd column) times the strength of connection (4th column). The rest of parameters are common such that $\tau_{AMPA} = 2\,ms$, $\tau_{NMDA} = 100\,ms$, $\tau_{GABA} = 5\,ms$, $V_E = 0\,mV$, $V_I = -70\,mV$, and $\alpha = 0.6332$.

S2 Appendix Dopamine-dependent plasticity of corticostriatal weights

Synaptic plasticity in CBGTPy is implemented using a dopamine-dependent plasticity rule, in which the synaptic updates are governed solely by local factors, without requiring individual neurons to access information about the global system state. This rule is an adaptation of the plasticity mechanism presented in [6].

At each corticostriatal AMPA synapse, the model tracks three key values: eligibility $E(t)$, weight $w(t)$, and conductance $g_x(t)$. The conductance is associated with the synaptic current. How much the conductance grows with each pre-synaptic spike is determined by the weight. The weight is the plastic element in the system, which changes over time depending on the time courses of eligibility and dopamine release.

all pathways network					direct/indirect pathways network				
Population	Receptor	External input frequency	External connection efficacy	External connections number	Population	Receptor	External input frequency	External connection efficacy	External connections number
CxI	AMPA	3.7	1.2	640	CxI	AMPA	3.7	1.2	640
Cx	AMPA	2.5	2.0	800	Cx	AMPA	2.3	2.0	800
$dSPN$	AMPA	1.3	4.0	800	$dSPN$	AMPA	1.3	4.0	800
$iSPN$	AMPA	1.3	4.0	800	$iSPN$	AMPA	1.3	4.0	800
FSI	AMPA	4.8	1.55	800	FSI	AMPA	3.6	1.55	800
GPe_A	GABA	2.0	2.0	2000					
	AMPA	2.5	2.0	800					
GPe_P	GABA	2.0	2.0	2000	$^*GPe$	GABA	2.0	2.0	2000
	AMPA	4.0	2.0	800		AMPA	4.0	2.0	800
GPi	AMPA	0.84	5.9	800	GPi	AMPA	0.8	5.9	800
STN	AMPA	4.45	1.65	800	STN	AMPA	4.45	1.65	800
Th	AMPA	2.2	2.5	800	Th	AMPA	2.2	2.5	800

S4 Table: **External current parameters.** Parameters used to describe the external current (I_{ext}) arriving at the different populations of the CBGT network. From the third column to the last, we specify the different parameters used to describe the external current impinging in each population specified in column 1 and for the specific type of receptors. A non described receptor type means that the parameters are considered to be zero. The time decay constant τ is the same for all populations and only depends on the type of receptor being $\tau = 2\,ms$ if the receptor type is AMPA and $\tau = 5\,ms$ if it is GABA. * Values in this row are the ones used when no intrinsic separation of neurons is considered.

At a computational level, $E(t)$, which represents a synapse's eligibility to undergo weight modification, depends on the relative spike times of the pre- and post-synaptic neurons involved in the synapse. To compute this quantity, we first define the variables $A_{PRE}(t)$ and $A_{POST}(t)$, which serve as instantaneous estimates of the recent levels of pre- and post-synaptic spiking, respectively. Each time a spike occurs in the pre- or post-synaptic cell, these values are increased by a fixed amount (Δ_{PRE} and Δ_{POST}, respectively), and between spikes, they decay exponentially with a time decay constant τ_{PRE} and τ_{POST}, respectively. That is,

$$\frac{dA_{PRE}}{dt} = \frac{1}{\tau_{PRE}} \left(\Delta_{PRE} X_{PRE}\left(t\right) - A_{PRE}(t)\right),$$
$$\frac{dA_{POST}}{dt} = \frac{1}{\tau_{POST}} \left(\Delta_{POST} X_{POST}\left(t\right) - A_{POST}(t)\right)$$

where $X_{PRE}(t)$ and $X_{POST}(t)$ are sums of Dirac delta functions representing the spike trains of the two neurons. That is,

$$X_{PRE} = \sum_{t_s \in \mathcal{X}_{Cx}} \delta(t - t_s), \qquad X_{POST} = \sum_{t_s \in \mathcal{X}_{SPN}} \delta(t - t_s),$$

where t_s is the spike onset, $\mathcal{X}_{Cx}$ is the set of all cortical neurons projecting to the postsynaptic neuron of interest, and $\mathcal{X}_{SPN}$ refers to the identity of that postsynaptic neuron within the striatum.

Eligibility ($E(t)$) changes over time according to

$$\frac{dE}{dt} = \frac{1}{\tau_E} \left(X_{POST}\left(t\right) A_{PRE}(t) - X_{PRE}\left(t\right) A_{POST}(t) - E\right) \tag{2}$$

where τ_E is a time constant. Note that based on equation (2), $E(t)$ tends toward a level that is boosted whenever a post-synaptic spike occurs soon enough after a pre-synaptic spike and is reduced whenever a pre-synaptic spike occurs soon enough after a post-synaptic spike.

The corticostriatal synaptic conductance g_x takes the value of the synaptic weight, $w(t)$, at each pre-synaptic spike time and decays exponentially in-between these spikes:

$$\frac{dg_x}{dt} = \sum_j w(t_j)\delta(t - t_j) - \frac{g_x}{\tau_{AMPA}},$$

where x stands for the specific connection, t_j denotes the time of the $j-$th spike in the cortical presynaptic neuron, $\delta(t)$ is the Dirac delta function, τ_{AMPA} is the decay time constant associated with AMPA synapses, and w itself changes over time based on dopamine release and the post-synaptic neuron's eligibility. The evolution of w is given by

$$\frac{dw}{dt} = [\alpha_w^j E(t) f(K_{DA})(w_{max}^j - w)]^+ + [\alpha_w^j E(t) f(K_{DA})(w - w_{min}^j)]^-, \tag{3}$$

where the nomenclature $[\cdot]^+$ ($[\cdot]^-$) represents a function whose output is the value inside the brackets if it is positive (negative) and 0 otherwise. The learning rate is denoted in equation (3) by α_w^j, for $j \in \{dSPN, iSPN\}$, depending on to which of the two populations the post-synaptic neuron belongs. This rate has a positive sign for dSPN neurons and a negative one for iSPN neurons to reproduce the observation that positive feedback signals lead to a strengthening of the eligible direct pathway connections and a weakening of the eligible indirect pathway connections. Furthermore, w_{max}^j and w_{min}^j are upper and lower bounds for the weight w, respectively, for $j \in \{dPSN, iSPN\}$.

In equation (3), the variable K_{DA} represents the level of available dopamine in the network, which is computed from the amount of dopamine released through the effect of the differential equation

$$\frac{dK_{DA}}{dt} = C_{scale} \sum_i (DA_{inc}(t_i) - K_{DA})\delta(t_i) - \frac{K_{DA}}{\tau_{DA}},$$

where $DA_{inc}(t_j)$ the increment of dopamine, relative to a baseline level, that is delivered at time t_j. That is, after a specific decision i is made at time t_j, a reward value $r_i(t_j)$ associated to action i is received, which induces a dopamine increment based on the reward prediction error

$$DA_{inc}(t_j) = r_i(t_j) - Q_i(t_j),$$

where $Q_i(t_j)$ is the expected reward for action i at time t_j. This expected reward obeys the update rule

$$Q(t_{j+1}) = Q_i(t_j) + \alpha_Q(r_i(t_j) - Q_i(t_j)),$$

where $\alpha_Q \in [0, 1]$ is the value learning rate. More precisely, note that to account for the motor sensory response, the reward is delivered to the network at the end of *phase 1*, $300\,ms$ after the decision is made (see Fig 3 in the manuscript); Q and DA_{inc} are updated together at this reward delivery time, and the update of DA_{inc} in turn impacts the evolution of K_{DA}. Finally, the function $f(K_{DA})$ in equation (3) represents the impact that the available dopamine K_{DA} has on plasticity, such that, if the target neuron lies in the dSPN population, then

$$f(K_{DA}) = \begin{cases} -\gamma, & \text{if } K_{DA} < -\mu, \\ \frac{\gamma}{\mu} K_{DA}, & \text{if } K_{DA} \geq -\mu, \end{cases}$$

while if the target neuron lies in the iSPN population, then

$$f(K_{DA}) = \begin{cases} \varepsilon \frac{\gamma}{\mu} K_{DA}, & \text{if } K_{DA} < \mu, \\ \varepsilon\gamma, & \text{if } K_{DA} \geq \mu. \end{cases}$$

for fixed, positive scaling parameters γ, μ. Parameters values used for the plasticity implementation can be found in S14 Table

To achieve effective learning, it is critical to address the credit assignment problem of ensuring that the pathways promoting the choice of selected action are the ones that are reinforced by the reward following that action. To achieve this alignment, we introduce a sustained activation signal to the action channel associated with the selected action throughout *phase 1*, based on the patterns of sustained activity that have been observed in motor planning tasks [20]. Specifically, during this phase, the internal gain of cortical stimulation is altered so that the cortical population corresponding to the selected action maintains elevated activity (at 70% of its firing rate from the end of *phase 0*), while cortical populations corresponding to other actions return to baseline firing now that those actions are no longer under consideration. The localized,

Parameter	Value
δ_{PRE}	0.8
δ_{POST}	0.04
τ_{PRE}	$15\,ms$
τ_{POST}	$6\,ms$
τ_E	$100\,ms$
α_w^{dSPN}	39.5
α_w^{iSPN}	-38.2
w_{max}^{dSPN}	0.055
w_{max}^{iSPN}	0.035
w_{min}^{dSPN}	0.001
w_{min}^{iSPN}	0.001
ε	0.3
γ	3.0
μ	0.5
C_{scale}	85
τ_{DA}	$2.0\,ms$
α_Q	0.6

S5 Table: **Parameters used for the plasticity implementation.**

sustained cortical activation ensures that the downstream striatal neurons in the appropriate action channel have high eligibility [10].

Taken together, the alteration of the direct-indirect pathway balance increases the tendency of the network to select the rewarded action, giving rise to learning. By using a realistic plasticity rule to produce learning, CBGTPy will enable users to investigate the interplay between the dopaminergic system and basal ganglia dynamics in a way that would be impossible with a less physiologically-accurate learning rule.

S3 Appendix CGBTpy installation and dependencies

The CBGTPy codebase is written in Python 3.8. If the user is using Python version < 3, e.g. 2.7, some of the dependent libraries may not work. Further details about the installation procedure can be found on our github repository.

S4 Appendix List of files

Here we provide the list of the files that are found on our Github repository and that make up the network, including a short reference to what is implemented in each of them. We distinguish between different sets of files: some are common and used regardless of the type of experiment performed. The remainder have separate versions specific to each experiment type, either the n-choice experiment or the stop-signal task, enabling easier swapping between alternative configurations.

The common files are:

- agentmatrixinit.py: builds the CBGT network.
- backend.py: functions for handling pipeline modules, also connects to the Ray server.
- frontendhelpers.py: deals with the environment variable passed.
- generateepochs.py: where rewards and changepoints are defined; rewards are probabilistic and delivered according to which action has been chosen.
- pipeline_creation.py: creates all modules constituting the pipeline.

- plotting_functions.py: implementation of functions useful for data visualization.
- plotting_helper_functions.py: implementation of functions useful for extracting relevant data.
- postprocessing_helpers.py: contains code to extract the data frames for recorded variables.
- qvalues.py: sets up and updates the parameters for the Q-learning algorithm on every trial.
- setup.py: cythonizes the corresponding core simulator code in agent_timestep.pyx.
- tracetype.py: defines wrapper classes that can pair numeric values with metadata.
- generate_opt_dataframe.py: reads in all optogenetic signal-related parameters and generates a data frame.

The files that are used for the simulation of the plasticity experiments are:

- agent_timestep_nchoice.pyx: contains code for simulating the timesteps of the spiking network.
- init_params_nchoice.py: sets neurons' parameters, receptors' parameters, populations' parameters, dopamine-related parameters for dSPNs and iSPNs, and action channels' parameters with either the defaults or values passed as arguments from the notebook.
- interface_nchoice.py: main simulation controller loop, interacts between environment and the CBGT network.
- popconstruct_nchoice.py: sets up connections between populations and corresponding parameters such as the probability of connection, the mean synaptic efficacy, and the parameters associated with synaptic plasticity (S2 Appendix).

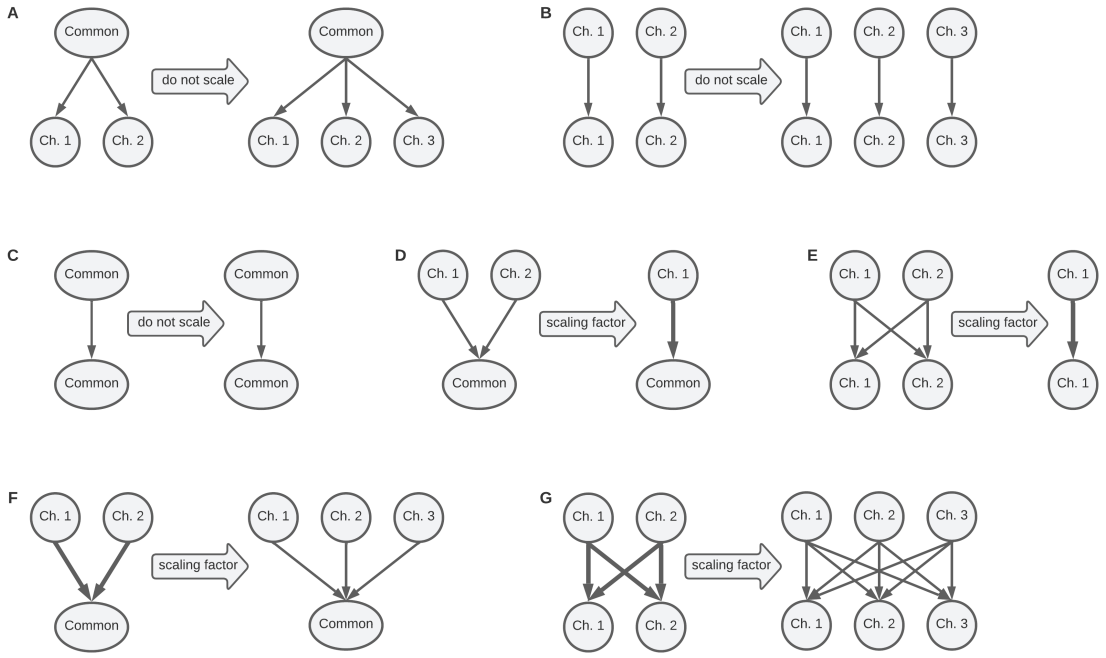
The files that belong to the stop-signal task experiment are:

- agent_timestep_stopsignal.pyx: contains code for simulating the timesteps of the spiking network.
- generate_stop_dataframe.py: reads in all stop signal-related parameters and generates a data frame.
- init_params_stopsignal.py: sets neurons' parameters, receptors' parameters, populations' parameters, dopamine-related parameters for dSPNs and iSPNs, and action channels' parameters with either the defaults or values passed as arguments from the notebook; this version differs from the one used to perform the plasticity experiment since different populations are considered for the simulation of the two experiments.
- interface_stopsignal.py: main simulation controller loop, interacts between environment and the CBGT network.
- popconstruct_stopsignal.py: sets connections between populations and corresponding parameters such as the probability of connection, the mean synaptic efficacy, and the parameters associated with synaptic plasticity.

S5 Appendix Network scaling

The CBGTPy model allows for the simulation of networks with an arbitrary number of action channels, with a default setting of 2 channels. As the number of channels is varied, certain pathways are automatically adjusted to ensure that the overall quantity of synaptic input to each subpopulation remains constant, allowing the neurons to maintain their proper baseline firing rates. To determine which pathways require scaling, the connectivity pattern of each pathway is compared to a set of cases, which are outlined in S1 Fig. If, for a given pathway, each target subpopulation only receives input from a single channel or from a shared source, no scaling

factor is applied. If, however, each target subpopulation receives input from all action channels, then that pathway requires a scaling factor. This factor is calculated as $2/n$, where n is the new number of action channels. When $n > 2$, the scaling factor is used to reduce the connection probability so that the expected number of afferent synapses per neuron remains constant. As a special case, when $n = 1$ and the scaling factor is 2, the weights of the synapses are increased rather than the connection probability, to avoid potentially setting the pathway's connection probability over 100%. For a detailed listing of connections to which a scaling factor is applied, see S6 Table.



S1 Fig: **Overview of scaling rules.** Pathways featuring solely divergent (**A**) or parallel (**B**,**C**) connectivity never have a scaling factor applied. As the number of incoming connections to each target subpopulation remains constant, no scaling of the pathway parameters is needed. When the number of action channels is reduced from 2 to 1, pathways defined by convergent (**D**) or all-to-all (**E**) connectivity have their synaptic weights scaled up by a factor of 2. When the number of action channels is increased above 2, convergent (**F**) and all-to-all (**G**) pathways have their synaptic connection probabilities decreased via the scaling factor.

S6 Appendix Supporting tables

S7 Appendix Supporting Figures

all pathways network			direct/indirect pathways network		
Connected populations	Receptor type	Scaling rule applied?	Connected populations	Receptor type	Scaling rule applied?
$CxI - CxI$	GABA	no	$CxI - CxI$	GABA	no
$CxI - Cx$	GABA	no	$CxI - Cx$	GABA	no
$Cx - Cx$	AMPA	no	$Cx - Cx$	AMPA	no
	NMDA	no		NMDA	no
$Cx - CxI$	AMPA	yes	$Cx - CxI$	AMPA	yes
	NMDA	yes		NMDA	yes
$Cx - dSPN$	AMPA	no	$Cx - dSPN$	AMPA	no
	NMDA	no		NMDA	no
$Cx - iSPN$	AMPA	no	$Cx - iSPN$	AMPA	no
	NMDA	no		NMDA	no
$Cx - FSI$	AMPA	yes	$Cx - FSI$	AMPA	yes
$Cx - Th$	AMPA	no	$Cx - Th$	AMPA	no
	NMDA	no		NMDA	no
$dSPN - dSPN$	GABA	no	$dSPN - dSPN$	GABA	no
$dSPN - iSPN$	GABA	no	$dSPN - iSPN$	GABA	no
$dSPN - GPi$	GABA	no	$dSPN - GPi$	GABA	no
$dSPN - GPe_A$	GABA	no			
$iSPN - iSPN$	GABA	no	$iSPN - iSPN$	GABA	no
$iSPN - dSPN$	GABA	no	$iSPN - dSPN$	GABA	no
$iSPN - GPe_A$	GABA	no	$iSPN - GPe$	GABA	no
$iSPN - GPe_P$	GABA	no			
$FSI - FSI$	GABA	no	$FSI - FSI$	GABA	no
$FSI - dSPN$	GABA	no	$FSI - dSPN$	GABA	no
$FSI - iSPN$	GABA	no	$FSI - iSPN$	GABA	no
$GPe_A - GPe_A$	GABA	yes			
$GPe_A - iSPN$	GABA	no			
$GPe_A - dSPN$	GABA	no			
$GPe_A - FSI$	GABA	yes			
$GPe_P - GPe_P$	GABA	yes	$GPe - GPe$	GABA	yes
$GPe_P - GPe_A$	GABA	no			
$GPe_P - FSI$	GABA	yes			
$GPe_P - STN$	GABA	no	$GPe - STN$	GABA	no
$GPe_P - GPi$	GABA	no	$GPe - GPi$	GABA	no
$GPe_P - FSI$	GABA	no			
$STN - GPe_P$	AMPA	no	$STN - GPe$	AMPA	no
	NMDA	no		NMDA	no
$STN - GPe_A$	AMPA	no			
	NMDA	no			
$STN - GPi$	AMPA	yes	$STN - GPi$	AMPA	yes
$GPi - Th$	GABA	no	$GPi - Th$	GABA	no
$Th - dSPN$	AMPA	no	$Th - dSPN$	AMPA	no
$Th - iSPN$	AMPA	no	$Th - iSPN$	AMPA	no
$Th - FSI$	AMPA	yes	$Th - FSI$	AMPA	yes
$Th - Cx$	AMPA	yes	$Th - Cx$	AMPA	yes
$Th - CxI$	AMPA	yes	$Th - CxI$	AMPA	yes

S6 Table: **Scaling rule application per connection.** Two blocks of 2 columns each are depicted. The first block applies to the full network containing all pathways, while the second block applies to the reduced network containing only the direct/indirect pathways.

```
In [12]: # List all the agent variables accessible
         results[0].keys()

Out[12]: dict_keys(['experimentchoice', 'params', 'pops', 'receps', 'base', 'dpmns', 'd1', 'd2', 'channels', 'newpathways',
         'Q_support_params', 'Q_df_set', 'n_trials', 'volatility', 'conflict', 'reward_mu', 'reward_std', 'maxstim', 'recor
         d_variables', 'opt_signal_present', 'opt_signal_probability', 'opt_signal_amplitude', 'opt_signal_onset', 'opt_sig
         nal_duration', 'opt_signal_channel', 'opt_signal_population', 'sustainedfraction', 'actionchannels', 'volatile_pat
         tern', 'cp_idx', 'cp_indicator', 'noisy_pattern', 't_epochs', 'block', 'trial_num', 'chosen_action', 'celldefault
         s', 'popspecific', 'receptordefaults', 'basestim', 'dpmndefaults', 'd1defaults', 'd2defaults', 'popdata', 'pathway
         s', 'opt_df', 'opt_channels_df', 'opt_amplitude_df', 'opt_onset_df', 'opt_populations_df', 'opt_list_trials', 'con
         nectivity_AMPA', 'meaneff_AMPA', 'plastic_AMPA', 'connectivity_GABA', 'meaneff_GABA', 'plastic_GABA', 'connectivit
         y_NMDA', 'meaneff_NMDA', 'plastic_NMDA', 'Q_df', 'AMPA_con', 'AMPA_eff', 'GABA_con', 'GABA_eff', 'NMDA_con', 'NMDA
         _eff', 'agent', 'datatables', 'reward_val', 'popfreqs'])
```

S2 Fig: **List of parameters and data frames that are returned from the simulation.**

Parameter	Definition
N	Number of receptors of the neuron
C	Capacitance in nF
$Taum$	Membrane time constant in ms
$RestPot$	Neuron resting potential in mV
$ResetPot$	Neuron reset potential in mV
$Threshold$	Neuron reset potential in mV
$RestPot_ca$	Resting potential for calcium ions
$Alpha_ca$	Amount of increment of [Ca] with each spike discharge
Tau_ca	Time constant of Ca-related conductance
Eff_ca	Calcium efficacy
$tauhm$	Duration of the burst in ms
$tauhp$	Duration of hyperpolarization necessary to recruit a maximal post-inhibitory rebound response in ms
$V_$	Threshold for bursts activation in mV
V_T	Low-threshold of Ca reversal potential in mV
g_T	Low-threshold of Ca maximal conductance in mS/cm^2
g_adr_max	Maximum value of the conductance
$Vadr_h$	Potential for g_adr_max
$Vadr_s$	Slop of g_adr at $Vadr_h$, defining how sharp the shape of g_ard is
$ADRRevPot$	Reverse potential for ADR
g_k_max	Maximum outward rectifying current
Vk_h	potential for g_k_max
Vk_s	Defines how sharp the shape of g_k is
tau_k_max	Maximum time constant for outward rectifying K current
n_k	Gating variable for outward rectifying K channel
h	Gating variable for the low-threshold Ca current

S7 Table: **Neuronal parameters editable by the user.** These parameters can be modified through the data frame `params`.

Parameter	Definition
N	Population-specific number of neurons in the nuclei
C	Capacitance in nF
$Taum$	Membrane time constant in ms
g_T	Ca low-threshold maximal conductance in mS/cm^2

S8 Table: **Population-specific neuron parameters changeable by the user.** These parameters can be modifiable through the dictionary `pops`, addressing the population of interest.

S9 Table: **Population-specific neuron parameters changeable by the user.** These parameters can be modified through the dictionary `pops`, addressing the population of interest.

Parameter	Definition
Tau_AMPA	AMPA time constant in ms
$RevPot_AMPA$	AMPA reversal potential in mV
Tau_GABA	GABA time constant in ms
$RevPot_GABA$	GABA reversal potential in mV
Tau_NMDA	NMDA time constant in ms
$RevPot_NMDA$	NMDA reversal potential in mV

S10 Table: **Synaptic parameters changeable by the user.** These parameters can be modified through the data frame `receps`.

Parameter	Definition
$FreqExt_AMPA$	Baseline input firing rate to AMPA receptors
$MeanExtEff_AMPA$	AMPA conductance
$MeanExtCon_AMPA$	average of AMPA connections
$FreqExt_GABA$	input firing to GABA receptors
$TMeanExtEff_GABA$	GABA conductance
$MeanExtCon_GABA$	average of GABA connections

S11 Table: **Population-specific baseline parameters modifiable by the user.** These parameters can be modified through the dictionary `base`, addressing the population of interest.

Parameter	Definition
$dpmn_DOP$	Time constant of the dopamine trace
$dpmn_DAt$	Tonic dopamine
$dpmn_dPRE$	Fixed increment for pre-synaptic spiking (Apre)
$dpmn_dPOST$	fixed increment for post-synaptic spiking (Apost)
$dpmn_tauE$	Eligibility trace decay time constant
$dpmn_tauPRE$	Decay time constant for the pre-synaptic spiking trace (Apre)
$dpmn_tauPOST$	Decay time constant for the post-synaptic spiking trace (Apost)
$dpmn_m$	Motivation, that modulates the strength of the dopamine level
$dpmn_E$	Eligibility trace
$dpmn_DAp$	Phasic dopamine
$dpmn_APRE$	Pre-synaptic spiking trace
$dpmn_APOST$	Post-synaptic spiking trace
$dpmn_XPRE$	Pre-synaptic spike time indicators
$dpmn_XPOST$	Post-synaptic spike time indicators
$dpmn_fDA_D1$	f(DA) value for D1-SPNs
$dpmn_fDA_D2$	f(DA) value for D2-SPNs
$dpmn_x_FDA$	threshold for f(DA) function
$dpmn_y_FDA$	threshold for f(DA) function
$dpmn_d2_DA_eps$	Scaling factor for dopamine levels of D2-SPNs as compared to D1-SPNs

S12 Table: **Dopamine-related parameters editable by the user.** These parameters can be modified through the data frame `dpmns`.

Parameter	Value
$dpmn_type$	1 for dopamine-related variables of dSPNs or 2 for iSPN neurons
$dpmn_alphaw$	weight increment proportional to the dopamine discharge
$dpmn_wmax$	upper bound for W

S13 Table: **Dopamine-related parameters for corticostriatal projections to dSPN and iSPN neurons.** Each of the striatal SPN population, dSPN and iSPN, maintain a copy of this data structure which can be independently modified through the data frames `dSPN_params` and `iSPN_params` defined in the `configuration` dictionary in the notebooks.

Parameter	Value
δ_{PRE}	0.8
δ_{POST}	0.04
τ_{PRE}	$15\,ms$
τ_{POST}	$6\,ms$
τ_E	$100\,ms$
α_w^{dSPN}	39.5
α_w^{iSPN}	-38.2
w_{max}^{dSPN}	0.055
w_{max}^{iSPN}	0.035
w_{min}^{dSPN}	0.001
w_{min}^{iSPN}	0.001
ε	0.3
δ	3.0
μ	0.5
C_{scale}	85
τ_{DA}	$2.0\,ms$
α_Q	0.6

S14 Table: **Parameters used for plasticity implementation.** For more details about the plasticity parameters, please refer to S2 Appendix. The parameters without a subscript can be modified using data frame `dpmns`, whereas the parameters with a subscript `dSPN` or `iSPN` can be modified through the data frames `dSPN_params` and `iSPN_params` respectively.

```
In [20]: results[0]['popfreqs']
Out[20]:
```

	GPi_left	GPi_right	STNE_left	STNE_right	GPeP_left	GPeP_right	D1STR_left	D1STR_right	D2STR_left	D2STR_right	Cx_left	Cx_right	Th_lef
0	64.000000	69.555556	25.511111	25.644444	61.755556	58.711111	5.333333	4.222222	4.666667	5.555556	0.000000	0.000000	7.777778
1	63.777778	68.444444	25.377778	25.711111	61.733333	58.911111	5.111111	4.444444	4.666667	5.333333	0.000000	0.000000	7.777778
2	64.000000	68.444444	25.511111	25.422222	61.488889	58.866667	5.111111	4.444444	4.666667	5.777778	0.000000	0.000000	7.777778
3	64.666667	67.555556	25.333333	25.488889	61.622222	58.266667	5.111111	4.444444	4.666667	5.777778	0.000000	0.000000	7.333333
4	65.111111	67.555556	25.200000	25.355556	61.800000	58.088889	5.111111	4.444444	4.666667	5.777778	0.000000	0.000000	7.333333
...	...	...	...	...	...	...	...	...	...	...	...	...	..
4026	64.222222	66.444444	27.133333	24.555556	56.711111	65.000000	5.555556	4.000000	5.555556	3.333333	0.735294	1.062092	10.000000
4027	63.777778	65.777778	27.400000	24.177778	56.555556	65.355556	5.777778	4.000000	5.333333	3.333333	0.735294	1.143791	9.777778
4028	64.222222	65.777778	27.355556	24.155556	56.977778	65.266667	5.777778	3.777778	4.888889	3.333333	0.735294	1.143791	9.111111
4029	65.111111	65.777778	27.377778	23.911111	57.422222	64.977778	5.777778	3.777778	4.888889	3.555556	0.735294	1.143791	9.111111
4030	64.444444	65.777778	27.600000	23.888889	57.977778	64.311111	5.777778	3.777778	4.888889	4.000000	0.735294	1.143791	9.333333

4031 rows × 17 columns

S3 Fig: **Example of `results['popfreqs']` data frame.**

```
In [28]: datatables[0]
Out[28]:
```

	decision	stimulusstarttime	decisiontime	decisionduration	decisiondurationplusdelay	rewardtime	correctdecision	reward
0	right	0	247	247	501	501	left	0.0
1	left	1102	1323	221	473	1575	right	0.0
2	none	2176	3177	1001	1253	3429	left	0.0

S4 Fig: **Example of `datatables[0]` data frame.**

```
In [29]: # Check the Q-values data frame
         results[0]['Q_df']
```

Out[29]:

	left	right
0	0.50	0.50
0	0.50	0.45
0	0.45	0.45
0	0.45	0.45

S5 Fig: **Example of `Q_df` data frame.**