A Hitchhiker's Guide to Bayesian Hierarchical Drift-Diffusion Modeling with dockerHDDM

Wanke Pan[1], Haiyang Geng[2], Lei Zhang[3, 4, 5], Alexander Fengler[6], Michael J. Frank[6], Ru-Yuan Zhang[7, 8], Hu Chuan-Peng[1]

[1] School of Psychology, Nanjing Normal University, Nanjing 210024, China
[2] Tianqiao and Chrissy Chen Institute for Translational Research, Shanghai, China
[3] Social, Cognitive and Affective Neuroscience Unit, Department of Cognition, Emotion, and Methods in Psychology, Faculty of Psychology, University of Vienna, Vienna, 1010, Austria
[4] Centre for Human Brain Health, School of Psychology, University of Birmingham, Birmingham B15 2TT, UK
[5] Institute for Mental Health, School of Psychology, University of Birmingham, Birmingham B15 2TT, UK
[6] Department of Cognitive, Linguistic and Psychological Sciences, Brown University, Providence, United States
[7] School of Psychology, Shanghai Jiao Tong University, Shanghai 200030, China.
[8] Shanghai Mental Health Center, School of Medicine, Shanghai Jiao Tong University, Shanghai 200030, China.

Author Note
Wanke Pan https://orcid.org/0000-0002-0896-6833
Hu Chuan-Peng https://orcid.org/0000-0002-7503-513
Haiyang Geng https://orcid.org/0000-0001-6115-807X
Lei Zhang https://orcid.org/0000-0002-9586-595X
Alexander Fengler https://orcid.org/0000-0002-0104-3905
Michael J. Frank https://orcid.org/0000-0001-8451-0523
Ru-Yuan Zhang https://orcid.org/0000-0002-0654-715X

Correspondence:
Hu Chuan-Peng,
School of Psychology
Nanjing Normal University (Suiyuan campus)
#122 Ninghai Road, Gulou District, 210024 Nanjing, Jiangsu Province, China
Email: hu.chuan-peng@nnu.edu.cn

Ru-Yuan Zhang
School of Psychology and Shanghai Mental Health Center
Shanghai Jiao Tong University
1954 HuaShan RD, Xuhui District, 200030 Shanghai, China
Email: ruyuanzhang@sjtu.edu.cn

Abstract

Drift diffusion models (DDMs) are pivotal in understand decision-making processes across psychology, behavioral economics, neuroscience, and psychiatry. Hierarchical drift diffusion models (HDDM), a Python library for hierarchical Bayesian estimation of DDMs, has been widely used among researchers, including those with limited coding proficiency, in fitting DDMs and other sequential sampling models to their data. However, issues of compatibility in installation and lack of support for more recently Bayesian modeling functionalities poses serious challenges for new users, limiting broader application of HDDM and reproducibility of research that used HDDM. To address these issues, we dockerize HDDM and add new functions into dockerHDDM, which brings three improvements: (1) easy-to-install once docker is installed, ensuring reproducibility and saving time for researchers; (2) compatible with machine with apple chips; (3) seamlessly integration with ArviZ, a state-of-the-art Bayesian modeling library. This tutorial serves as a practical, hands-on guide for researchers to leverage dockerHDDM's capabilities in conducting efficient Bayesian hierarchical analysis of DDMs. The notebook presented here and within the docker image will enable researchers with various programming levels to model their data with HDDM.

Keywords: HDDM, drift diffusion models, Bayesian hierarchical modeling, Reproducibility, Docker, Python

Box 1. Glossary of Terms Used in Bayesian Modeling

Prior, or prior distribution, often referred to as $p(\theta)$, is the initial belief that researchers have about the parameters θ in a model before observing data. It can be formed either from existing research or from pilot data.

Likelihood, or likelihood function, often referred to as $p(y|\theta)$, is the probability of the observed data y as a function of the specific parameters θ of a chosen statistical model. For example, the Bernoulli function is the likelihood function for statistically describing coin tossing.

Posterior, or posterior distribution, often referred to as $p(\theta|y)$, refers to the updated knowledge about the parameters θ after observing the data y, balancing prior knowledge with observed data according to the Bayes rule, i.e., $p(\theta|y) \propto p(y|\theta)p(\theta)$.

Markov chain Monte Carlo (MCMC), is a sampling method to infer the posterior distribution by simulation. The Markov chains (usually multiple MCMC chains are required) are algorithmically constructed so that their corresponding stationary distribution using MCMC samples approximates the posterior distribution of interest. The process of reaching this stationary distribution is called MCMC convergence. These sampled parameter values serve as the approximation to the posterior distribution and can then be used to obtain empirical estimates of the posterior distribution, and associated summary statistics of interest, using Monte Carlo integration. In the literature, a *chain* (or *trace*) is referred to as a collection of *samples* (or *draws*). Traces serve as a basis for diagnosing convergence and/or other potential problems with the procedure in a given application. MCMC is particularly useful for models with high complexity.

Effective sample size (ESS), is the number of independent samples with the same estimation power as the *N* autocorrelated samples from one MCMC chain. ESS is often used to determine whether the number of draws in MCMC chains is sufficient to guarantee reliable estimation of uncertainty. An ESS of 100 per MCMC chain is recommended by (Vehtari, et al., 2021).

Gelman-Rubin statistics ($\hat{R}$), the ratio of within-chain variability to between-chain variability. Values close to 1.0 for all parameters and quantities of interest suggest that the Markov chain Monte Carlo algorithm has sufficiently converged to stationary distributions. In practice, a maximum $\hat{R}$ of 1.05 is acceptable.

Posterior predictive samples, simulated new data conditional on the posterior distribution. The simulated data can then be used to check whether the model can be considered a good fit to the data-generating mechanism, by comparing the simulation with the observed data. This process is often called *posterior predictive checks* (PPCs).

Leave-one-out cross-validation (LOO-CV), a model evaluation approach that trains the model on all observations except observation y_i, and then predicts the hold-out observation y_i. This procedure is repeated for all *n* observations.

Log predictive density, $log\, p(\tilde{y}|\theta)$, an overall summary of a model's predictive abilities by estimating the log likelihood of new data $\tilde{y}$ given the true parameters $\hat{\theta}$. However, since both the new data $\tilde{y}$ and the true model parameters θ are typically unavailable in empirical data, the log predictive density is approximated using the observed data y and the posterior estimates of the parameters $\hat{\theta}$, hence $log\, p(\tilde{y}|\theta) \approx log\, p(y|\hat{\theta})$. This estimate, when multiplied by -2, gives the *deviance*, $-2\, log\, p(y|\hat{\theta})$. However, as $log\, p(y|\hat{\theta})$ is a biased estimate of $log\, p(\tilde{y}|\theta)$, an adjustment is required to correct the bias.

Log pointwise predictive density, likelihood of each observed data point conditional on the model parameters. In practice, this quantity is estimated using draws from the posterior in Bayesian analysis, i.e., the computed log pointwise predictive density: $\widehat{lpd} = \sum_{i=1}^{n} log\left(\frac{1}{S}\sum_{s=1}^{S} p(y_i\,|\theta^s)\right)$ (*lpd* in Vehtari et al., 2017, or *lppd* in Gelman et al., 2014).

Expected log pointwise predictive density (ELPD), a measure of predictive accuracy for *n* data points generated by the true data generating process. $\sum_{i=1}^{n} E_f\left(log\, p_{post}(\tilde{y}_i)\right)$, where *f* is the true model, *y* is the observed data, $\tilde{y}$ denotes future data or alternative datasets that could have been seen, E_f denotes expectation that averages over the distribution of data generating distribution, and p_{post} is the posterior distribution. This term is also called *mean log predictive density*.

61

Highest density interval (HDI): an estimate of a parameter's credible range in the context of Bayesian statistics. It encompasses an interval of the posterior distribution where each point within this interval has a higher density than points outside of it. For instance, a 95% HDI means that there is a 95% chance that the true parameter value falls within this range, making it a reliable indicator of parameter uncertainty. HDIs are commonly used for hypothesis testing regarding effect sizes, as well as comparisons across different conditions or groups.

A region of practical equivalence (ROPE) represents a predefined range of parameter values that are considered practically equivalent to zero, which could be based on existing literature or theoretical reasoning (Kruschke, 2018, 2021). To determine whether a parameter estimate is significantly different from zero, a ROPE might be set as a range around zero. If the 95% HDI of the parameter lies entirely outside this ROPE, the parameter is considered credibly different from zero. If the HDI is entirely within the ROPE, the parameter is effectively zero for practical purposes. Partial overlap suggests that the parameter's result should be interpreted with caution.

The drift-diffusion model (DDM) is one of the most widely used computational models (Ratcliff et al., 2016) to quantify decision-making processes in neuroscience (Cavanagh et al., 2011; Herz et al., 2016, 2017; Shadlen & Shohamy, 2016), psychology (Hu et al., 2020; Johnson et al., 2017; Kutlikova et al., 2023), behavioral economics (Desai & Krajbich, 2022; Sheng et al., 2020), and psychiatry (Ging-Jehli, Ratcliff, & Arnold, 2021; Pedersen et al., 2021). According to the DDM experimentally observed reaction-time choice pairs arise from a process of stochastic evidence accumulation to a decision boundary (e.g., Voss et al., 2013; Figure 1). This theoretical framework has been shown not only to correlate robustly with established neural substrates (Forstmann et al., 2016), but also to serve as a powerful measurement tool for examining individual differences across cognitive tasks, experimental manipulations, and participant populations (Evans & Wagenmakers, 2019). Despite its theoretical contributions, the DDM is difficult to apply to experimental data in practice, because the derivation of inference-relevant quantities (e.g., the likelihood function) requires a mathematical understanding of the complex stochastic process of evidence accumulation.

Several software packages have been developed to facilitate the application of DDM, proving particularly beneficial for researchers with limited computational expertise. Among them, *HDDM*, a Python library for hierarchical drift diffusion modeling, is by far the most cited toolbox in the community (Wiecki, Sofer, & Frank, 2013, with 908 citations in Google Scholar, retrieved on Mar. 20, 2024). Despite the success and popularity of HDDM, it suffers from several practical issues. First, the installation process of HDDM is cumbersome, exacerbated by its reliance on PyMC 2.3.8 for Markov Chain Monte Carlo (MCMC) sampling, a package that is no longer supported and may clash with latest computer modules. Second, and for the same reason, out of the box HDDM is not compatible with apple chips, which creates a significant barrier for Mac users. Third, although HDDM natively centers around Bayesian methods, it does not conveniently support all aspects of the evolved standards in Bayesian modeling workflows (Gelman et al., 2020; Kruschke, 2021; Zhang et al., 2020). Significant progress has recently been made in supporting the principled Bayesian modeling workflow in easy-to-use toolkits, such as the Python package ArviZ (Kumar et al., 2019). Bridging these new capabilities with HDDM facilitates a one-stop Bayesian modeling pipeline for experimentalists and computational modelers interested in applying the DDM to their experimental data.

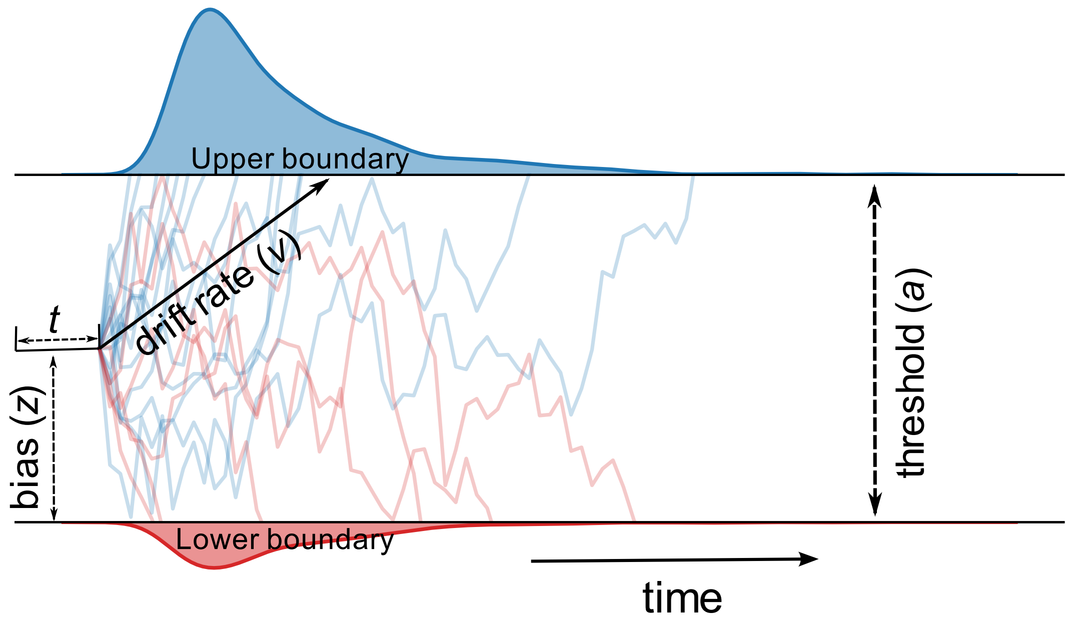


Figure 1. Illustration of the evidence accumulation process assumed by DDM. DDM has four basic parameters: drift rate (v), decision boundary (a), initial bias (z), and non-decision time (t). The drift rate (v) is the average speed of evidence accumulation toward a decision; the decision boundary (a) is the distance between two decision thresholds, and the evidence needed to make a decision increase as a increases; the initial bias (z) reflects the starting point of evidence accumulation. When z is closer to one of the boundaries, less evidence is required for that decision (conversely more evidence is required for the opposite decision); non-decision time (t) is the time not used for evidence accumulation, e.g., stimulus encoding or motor execution. A more complete version of DDM assumes that the values of drift rate, initial bias, and non-decision time vary across trials due to fluctuations in various psychological and physiological factors (e.g., attention lapse, arousal), so three additional parameters are included: trial-by-trial variation in drift rate (sv), variation in the initial bias (sz), and variation in non-decision time (st).

To address the above issues, we leveraged the *Docker* container technology to create *dockerHDDM*, a stable and complete virtualized Python computing environment that enables out-of-the-box implementations of Bayesian hierarchical drift-diffusion models. dockerHDDM has three major advantages (Table 1). First, it benefits from the easy-to-deploy nature of the Docker environment to avoid compatibility issues. Second, it is compatible with both Intel or Apple chips. Third, it augments HDDM with ArviZ, a Python module that enables a wide range of advanced Bayesian modeling analyses. We expect dockerHDDM to provide an easy-to-use environment to help researchers across various backgrounds efficiently use DDM in their research.

Table 1. Comparisons between dockerHDDM and the original HDDM package

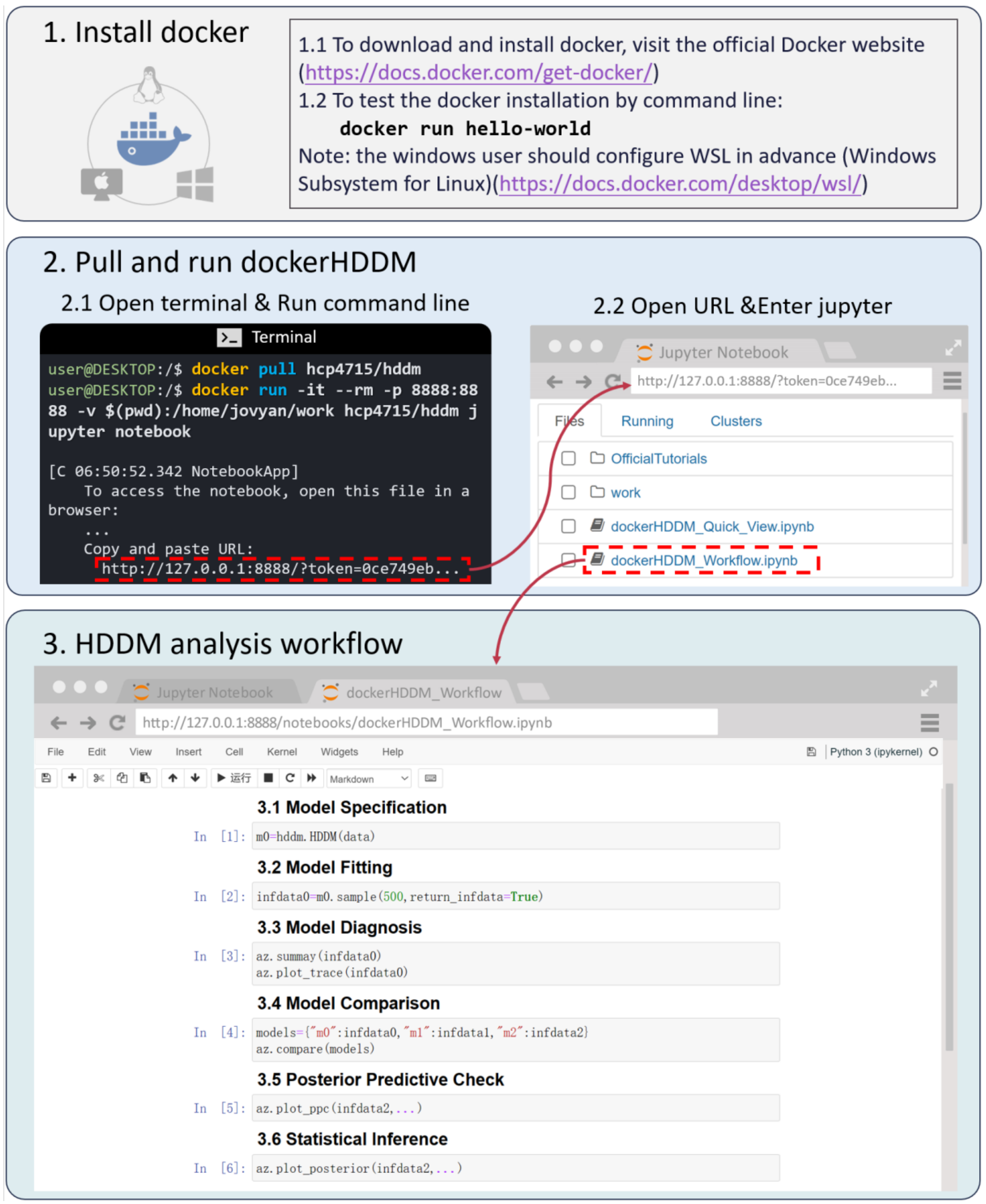
	HDDM	dockerHDDM
Support ArviZ *	No	Yes
Plotting (e.g., HDI,)	No	Yes
Diagnosis (e.g., ESS)	No	Yes
Model Comparison (LOO, WAIC)	No	Yes
Installation	Hard	Easy
Parallel processing	Hard	Easy
Compatibility with Apple chips	Hard	Easy

** Plotting, diagnosis, and model comparison are functions of ArviZ, including HDI, high-density interval; ESS, effective sample size, LOO, leave-one-out cross-validation; WAIC, widely applicable information criterion; PPC, posterior predictive checks.*

1. How to Follow This Tutorial

The primary goal of this paper is to present a practical guide to dockerHDDM for beginners with little modelling experience. The tutorial starts with step-by-step instructions on how to configure the dockerHDDM environment and how to use it in practical data analysis (Figure 2).

In the setup section (top panel in Figure 2, corresponding to Section 2.1 in this paper), we provide instructions on how to install Docker. After that, we demonstrate how to obtain the dockerHDDM image and how to use this image to access the Jupyter notebook interface (middle panel in Figure 2, corresponding to Sections 2.2 and 2.3). Finally, within a working Jupyter notebook we show how to analyze an example dataset with dockerHDDM in a principled Bayesian workflow (bottom panel in Figure 2, corresponding to Section 4).


1. Install docker

1.1 To download and install docker, visit the official Docker website (https://docs.docker.com/get-docker/)
1.2 To test the docker installation by command line:

```
docker run hello-world
```

Note: the windows user should configure WSL in advance (Windows Subsystem for Linux)(https://docs.docker.com/desktop/wsl/)

2. Pull and run dockerHDDM

2.1 Open terminal & Run command line

```
user@DESKTOP:/$ docker pull hcp4715/hddm
user@DESKTOP:/$ docker run -it --rm -p 8888:88
88 -v $(pwd):/home/jovyan/work hcp4715/hddm j
upyter notebook

[C 06:50:52.342 NotebookApp]
    To access the notebook, open this file in a
browser:
    ...
    Copy and paste URL:
     http://127.0.0.1:8888/?token=0ce749eb...
```

2.2 Open URL &Enter jupyter

```
http://127.0.0.1:8888/?token=0ce749eb...
```

Files | Running | Clusters

- OfficialTutorials
- work
- dockerHDDM_Quick_View.ipynb
- dockerHDDM_Workflow.ipynb

3. HDDM analysis workflow

```
http://127.0.0.1:8888/notebooks/dockerHDDM_Workflow.ipynb
```

3.1 Model Specification

```
In [1]: m0=hddm.HDDM(data)
```

3.2 Model Fitting

```
In [2]: infdata0=m0.sample(500,return_infdata=True)
```

3.3 Model Diagnosis

```
In [3]: az.summay(infdata0)
        az.plot_trace(infdata0)
```

3.4 Model Comparison

```
In [4]: models={"m0":infdata0,"m1":infdata1,"m2":infdata2}
        az.compare(models)
```

3.5 Posterior Predictive Check

```
In [5]: az.plot_ppc(infdata2,...)
```

3.6 Statistical Inference

```
In [6]: az.plot_posterior(infdata2,...)
```

Figure 2. Flowchart of how to use dockerHDDM. The top panel describes how to install Docker, corresponding to Section 2.1; the middle panel describes how to pull and run dockerHDDM, corresponding to Sections 2.2 and 2.3; and the bottom panel shows the workflow in dockerHDDM, corresponding to Section 4. In the bottom panel, the green circle represents the model defined based on the specified data; the purple ellipse represents the *InferenceData* obtained after model fitting; the dotted box shows the pseudo code. After model diagnosis, evaluation and comparison, the optimal model (Model 2 "m2" with "infdata2") is selected and used for inferential analysis.

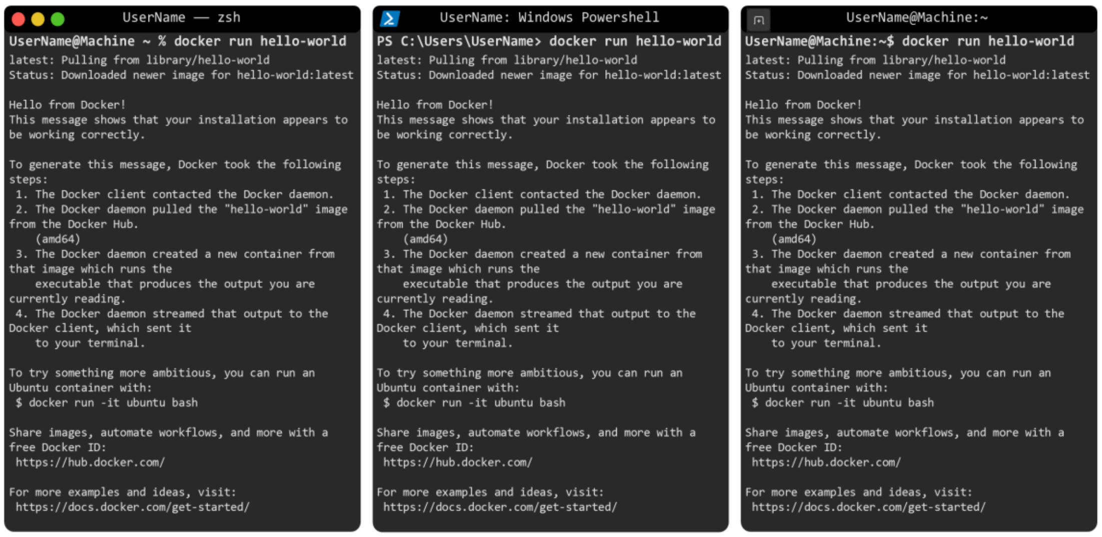
2. Install and Use dockerHDDM

2.1. Install Docker

Docker serves us to create an all-in-one, fast, cross-platform computing environment (e.g., Peikert & Brandmaier, 2021; Wiebels & Moreau, 2021). The Docker website provides easy-to-follow installation instructions (https://docs.docker.com/get-docker/) and supports Windows, MacOS, and Linux. Windows users should ensure their system version is 21H2 (build 19044) or higher and have either WSL or Hyper-V configured prior to installation (see https://docs.docker.com/desktop/install/windows-install/).

After installing Docker Desktop (or Docker Engine for Linux users), one can verify the installation by running the following command in a terminal[1] (Figure 3). If the container starts and runs successfully, it will display a confirmation message and then exit (Figure 3).

`docker run hello-world`



```
UserName@Machine ~ % docker run hello-world
latest: Pulling from library/hello-world
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to
be working correctly.

To generate this message, Docker took the following
steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image
from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from
that image which runs the
    executable that produces the output you are
currently reading.
 4. The Docker daemon streamed that output to the
Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an
Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a
free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

```
PS C:\Users\UserName> docker run hello-world
latest: Pulling from library/hello-world
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to
be working correctly.

To generate this message, Docker took the following
steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image
from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from
that image which runs the
    executable that produces the output you are
currently reading.
 4. The Docker daemon streamed that output to the
Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an
Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a
free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

```
UserName@Machine:~$ docker run hello-world
latest: Pulling from library/hello-world
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to
be working correctly.

To generate this message, Docker took the following
steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image
from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from
that image which runs the
    executable that produces the output you are
currently reading.
 4. The Docker daemon streamed that output to the
Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an
Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a
free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Figure 3. Command to check Docker installation in Terminal. After running the command `docker run hello-world` (highlighted at first line), the printout tells us that Docker has been successfully installed on the system. The schematic interfaces of the Terminal on different platforms: MacOS (left), Windows (middle), and Ubuntu (right).

[1] If you are unfamiliar with Terminal and the command line, don't panic! You can easily launch the Terminal application or the command line: MacOS users, search "Terminal" in Launchpad or Spotlight; Windows users, you can search for the terminal application "PowerShell"; Linux users, you can use the hotkey of "Ctrl, Alt and T" to start the Terminal. If you want to learn more about Termainal, we recommend https://www.freecodecamp.org/news/command-line-for-beginners/. Once the Terminal is active (see Figure 3), you can type `docker run hello-world` and then press "ENTER". For Windows and MacOS users, make sure the Docker desktop is running before typing `docker run hello-world`.

2.2. Pull dockerHDDM Image

After ensuring that Docker has been successfully installed and the Docker engine is running (Figure 3), you can pull the dockerHDDM image by simply running the command in the terminal (see the meaning of each argument in Figure 4A):

`docker pull hcp4715/hddm` or `docker pull hcp4715/hddm:latest`

This command will pull the latest default version of dockerHDDM, which corresponds to the image with the tag `1.0.1`. One can also select different tags for different versions of HDDM (see https://hub.docker.com/r/hcp4715/hddm/tags). Note that the tutorial in this paper works with the `latest` or `1.0.1` tags, it is compatible with 0.8.0, with minor grammar changes.

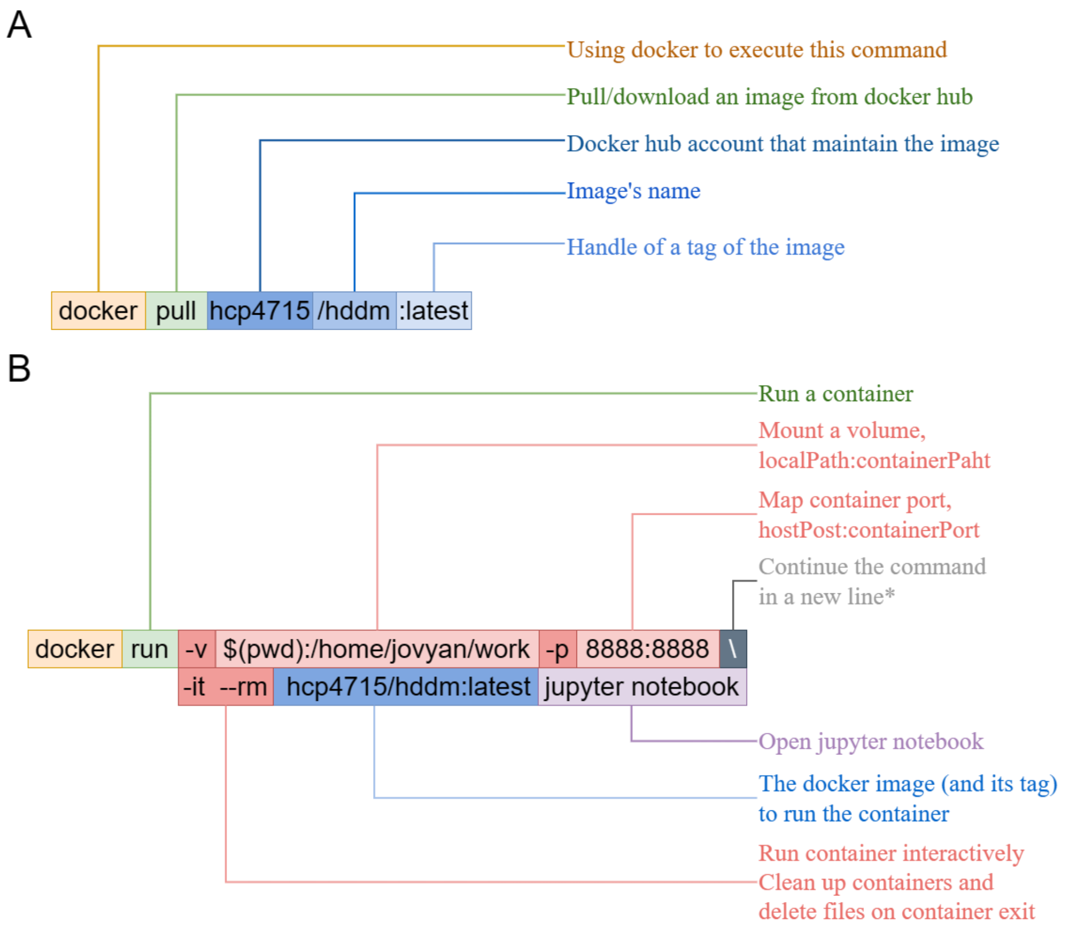

A
Using docker to execute this command
Pull/download an image from docker hub
Docker hub account that maintain the image
Image's name
Handle of a tag of the image
docker
pull
hcp4715
/hddm
:latest
B
Run a container
Mount a volume, localPath:containerPaht
Map container port, hostPost:containerPort
Continue the command in a new line*
docker
run
-v
$(pwd):/home/jovyan/work
-p
8888:8888
\
-it
--rm
hcp4715/hddm:latest
jupyter notebook
Open jupyter notebook
The docker image (and its tag) to run the container
Run container interactively
Clean up containers and delete files on container exit

Figure 4. Docker commands to download and run dockerHDDM. (A) Download/pull dockerHDDM from the Docker hub. The command by default downloads the latest version of `hcp4715/dockerHDDM` if the image tag is not specified. The CPU architecture (Apple or Intel chips, corresponding to ARM64 and AMD64 architectures, respectively) is automatically recognized when the image is downloaded. (B) Command to start a container. Note, "\" separates different lines of a command in Linux and MacOS Terminal but not in Windows.

2.3. Run dockerHDDM Container

After pulling the Docker image to a local machine, you can start a computing environment by running the dockerHDDM image with the command in the terminal (Figure 4B):

```
`docker run -v $(pwd):/home/jovyan/work -p 8888:8888
            -it --rm hcp4715/hddm jupyter notebook`
```

This command creates a Docker container, which is a specialized environment encapsulated within the Docker platform. The `` `-v` `` option is used to mount a local folder into the container's filesystem, enabling file exchange from the host machine. The example code `` `$(pwd):/home/jovyan/work` `` specifies two paths separated by a colon. The path on the left, denoted by `` `$(pwd)` ``, represents the current working directory on the host machine, and the path on the right, `` `/home/jovyan/work` ``, is the location inside the container where the folder will be mounted (Figure 4B). `` `$(pwd)` `` can be replaced with a valid folder path on your local machine, such as "D:\docker" on Windows, which is an absolute path to a folder named 'docker' on drive D. The other arguments in the command are explained in Figure 4B.

After running the `` `docker run …` `` command, a URL will be displayed at the end of the terminal output (middle panel in Figure 2). You can copy and paste this URL into any web browser (such as Firefox or Chrome) to launch a Jupyter interface based on the dockerHDDM container. You can then open or initialize a Jupyter notebook[2] to code, run and view the output directly. It is worth noting that the `` `--rm` `` flag included in the command means that the dockerHDDM container, along with any data or newly installed Python modules, will be deleted when the container stops. However, any files or data mounted to the container from the `` `$(pwd)` `` path will remain unaffected. This ensures the reproducibility of the computing environment. If you wish to modify the computing environment, for example by installing additional Python modules, we recommend that you first read the Docker API before removing `` `--rm` `` directly.

In the Jupyter interface, you will find two files and two folders (middle panel in Figure 2). The notebook *dockerHDDM_workflow.ipynb* offers a detailed reproduction of the analyses

[2] For beginners unfamiliar with Jupyter Notebook, don't panic! It is just an interface where you can write code and immediately check results. You may visit the official website at https://jupyter.org/try-jupyter/retro/notebooks/?path=notebooks/Intro.ipynb to try out a web-based platform online. The Jupyter website also provides extensive documentation for users who want to learn more about Jupyter Notebook and Python programming (see https://docs.jupyter.org/).

presented in this article, which we will discuss further in Section 3. In contrast, the notebook *dockerHDDM_Quick_View.ipynb* provides a brief overview of the dockerHDDM image's new features and an introduction to basic modeling processes. One folder is "work", which mounts the local path into the docker environment. The other folder, "OfficialTutorials" contains notebooks that reproduce the official tutorials available at https://hddm.readthedocs.io/en/latest/tutorials.html. Beginners can follow *HDDM_Basic_Tutorial.ipynb* to get a basic understanding of HDDM, as discussed in Wiecki et al. (2013); *HDDM_Regression_Stimcoding.ipynb* covers more advanced models with regression, where parameters can vary based on experimental conditions and other covariates; *Posterior_Predictive_Checks.ipynb* provide an introduction to posterior predictive checks for HDDM, showing how to generate predicted data from fitted parameter posteriors and how to analyze those predicted data; *LAN_Tutorial.ipynb* provides advanced use of LAN functions that address the problematic likelihood of more complicated models based on neural network methods (see Fengler, Govindarajan, Chen, & Frank, 2021).

3. New Features of dockerHDDM

The *dockerHDDM_Quick_View.ipynb* illustrates two new features in dockerHDDM (compared to HDDM installed directly without Docker): parallel computing for MCMC chains and creating *InferenceData* data for Arivz analyses (as shown in the <Code Block 1>).

<Code Block 1>

~~~
```Python
# define a simple model with preloaded data
model = hddm.HDDM(data)

# origin model fitting code
# model.sample(500, burn = 100)

# dockerHDDM new model fitting code
model.sample(
        500, burn = 100,
        chains = 4,                  # parallel computing for MCMC chains
        return_infdata = True,       # return InferenceData for Arivz analysis
        loglike = True, ppc = True,
        save_name = 'example'
)
```
~~~

For all hddm models defined by methods such as `hddm.HDDM()` or `hddm.HDDMRegressor()`, we can employ the `.sample()` method to run the MCMC algorithm for model fitting. The original HDDM provided two main parameters to set the MCMC algorithm, the first parameter was the number of samples (`500`) and the second was the number of burn-ins (`burn=100`)[3].

In dockerHDDM, we included five extra arguments in `.sample()` method to provide parallel computing for MCMC chains and create *InferenceData*.

To preserve compatibility and consistent output with origin HDDM, the arguments are configured with the following defaults: `return_infdata=False`, `loglike=False `, and `ppc=False `, `save_name=None`, and `chains=1`.

The `chains` argument determines the number of MCMC chains. Using more than two chains triggers multi-threaded parallel computation, which can significantly reduce time when multi-chains are need for calculating model diagnosis index $\hat{R}$ (see Section 4.4).

The `return_infdata` argument converts HDDM results into the InferenceData structure[4], accessible via `model.infdata`, by default set to `False` to maintain compatibility with original HDDM output. Additionally, we have included `loglike` for computing and saving log-likelihood values (see Section 4.5) and `ppc` for posterior predictive checks (see Section 4.6). When setting `ppc` as `True`, it defaults to generating 500 predictions for each observed data, but users can adjust this by add argument `n_ppc`.

Finally, the `save_name` argument specify the path and filename for saving the model and InferenceData, which is convenient for reusing results.

4. Example of Workflow

In this section (bottom panel of Figure 2), we demonstrate how to use dockerHDDM (i.e., HDDM and Arviz) to perform key steps of Bayesian modeling (Gelman et al., 2020; Martin et al., 2021): model specification and fitting, model diagnosis, model comparison, posterior predictive check,

[3] To run the example notebooks faster, we only use 500 samples here. For a more in-depth understanding of the MCMC settings, we recommend reading (van de Schoot et al., 2021; Wiecki et al., 2013). The burn-in samples serve to calibrate the fitting, so the final samples need to exclude burn-in samples, yielding a total of $500 - 100 = 400$ samples. Generally, a larger number of samples improves the estimation accuracy of a model.

[4] InferenceData is a more modern data construct that contains prior, posterior, a posterior predictive samples and observed data, facilitating the visualization and analysis of multiple joint datasets (Hoyer & Hamman, 2017).

and statistical inference. The code reproduced in this section can be found in *dockerHDDM_Workflow.ipynb* in dockerHDDM environment.

4.1. Example data

For convenience, we use the data from Cavanagh et al. (2011), which is built within HDDM, as an example to demonstrate how to implement the modeling workflow. This dataset contains reaction time and choice data from 14 Parkinson’s patients (see Table 2). In the experiment, participants were asked to choose between two options associated with either high or low reward values (i.e., reward probabilities in typical reinforcement learning tasks). The relative value differences between the two options define two levels conflict: high conflict for low-low and high-high trials (“HC” in variable “conf”), and low conflict for low-high trials (“LC” in variable “conf”).

Table 2 Example dataset from Cavanagh et al. (2011).

Subj_idx	rt	response	conf
0	1.21	1.0	HC
0	1.63	1.0	LC
0	1.03	1.0	HC
0	2.77	1.0	LC
0	1.14	0.0	HC

Note: The data structure required for HDDM is long-format data, where each row represents one trial. “subj_idx is” the subject index; “rt” is the response time (in seconds), and “response” in this case represents the accuracy, where is correct and 0 is incorrect. These three columns of data are mandatory when using HDDM and must be kept consistent with the column names, as well as the units (rt, seconds). “conf” is an optional variable, corresponding to the conflict level, and can be varied according to the experimental design.

Note that, HDDM requires the inclusion of three columns of variables, “subj_idx”, “rt” and “response”, to construct the hierarchical model. This means that when analyzing your own data, these three columns of variables must appear in the dataset with identical column names. In addition, the unit of “rt” must be seconds, and “response” is coded as 1 for the upper boundary of the corresponding choice and 0 for the lower boundary (see https://hddm.readthedocs.io/en/latest/howto.html for more details).

4.2. Model Specification

As a demonstration of model specification, we will test an example question: is there an effect of conflict levels on drift rate (see Wiecki et al., 2013). To answer the question, we constructed three computational models (see Table 3).

Table 3. Models used in this tutorial.

Models	HDDM functions for defining a model (`df` is the data from Cavanagh et al., 2011)	# params
Model 0	`hddm.HDDM(df, include=['a', 'v', 't', 'z', 'sv', 'sz', 'st'])`	67
Model 1	`hddm.HDDM(df, include=['a', 'v', 't','z', 'sv', 'st', 'sz'], depends_on={'v': 'conf'})`	82
Model 2	`hddm.HDDMRegressor(df, "v ~ 1 + C(conf, Treatment('LC'))", group_only_regressors=False, keep_regressor_trace=True, include=['a', 'v', 't', 'z', 'sv', 'st', 'sz'])`	83

Note: `hddm.HDDM()` is the default function for constructing a hierarchical drift diffusion model. The `include` argument allows the addition of free parameters, which are fixed by default. The `depends_on` argument specifies a parameter (e.g., v) that depends on a categorical independent variable (e.g., 'conf'). The `hddm.HDDMRegressor()` is a HDDM function that includes effects of conditions in a linear regression fashion. The `keep_regressor_trace` argument allows a trace of the regressor to be kept, which is needed for posterior predictive checks. By default, the hierarchical regression allows only the intercept to vary across participants, while the slope is fixed at the population level. The `group_only_regressors = FALSE` argument additionally estimates the slopes at the individual level in the regression model.

Model 0 served as the baseline without considering the effect of conflict level on the model parameters. The model contains the seven parameters, referred to as the full DDM, including the decision boundary (a), drift rate (v), non-decision time (t), and decision bias (z), as well as sv, st, and sz that indicates the trial-by-trial variations of v, t, and z (Boehm et al., 2018; Ratcliff & Rouder, 1998; Ratcliff & Tuerlinckx, 2002). By default, HDDM considers the hierarchical modeling approach that includes parameters at both the individual- and the group-level (see Box 2). Model 0 has 11 population-level parameters, including the mean and the standard deviation for the four basic parameters ($a/v/t/z$) and three parameters ($sv/st/sz$) for the inter-trial variations. At the individual level, each subject also has a full set of four basic parameters, yielding a total of $56 = 14 * 4$ parameters. Thus, Model 0 has $11 + 56 = 67$ free parameters.

296

Box 2 Parameters in hierarchical drift-diffusion models

HDDM employs hierarchical Bayesian modelling by default, where each participant's free parameters are sampled from population-level distributions (Wiecki et al., 2013). Taking full DDM (Model 0) as an example, non-decision time t_p is assumed to be drawn from a normal distribution: $t_p \sim N(u_t, \sigma_t)$, where u_t and σ_t are the mean and standard deviation of the population-level normal distribution of non-decision time *t*. Similarly, $u_z/u_a/u_v$ and $\sigma_z/\sigma_a/\sigma_v$ are the means and standard deviations for the other three parameters, respectively. In addition, three free parameters $st/sv/sa$ indicate the trial-by-trial variability of non-decision time (t), drift rate (v), and initial bias (a), which are estimated only at the population level. Consequently, there are a total of 11 population-level parameters.

At the subject level, each subject has her own estimate of the parameter of *a*, *v*, *t*, *z*, leading to a total of $4 * p$ subject -level parameters. Thus, in the full DDM, the number of parameters is 11 plus $4 * p$.

Full DDM: hddm.`HDDM(data, include=['z', 'sv', 'sz', 'st'])`

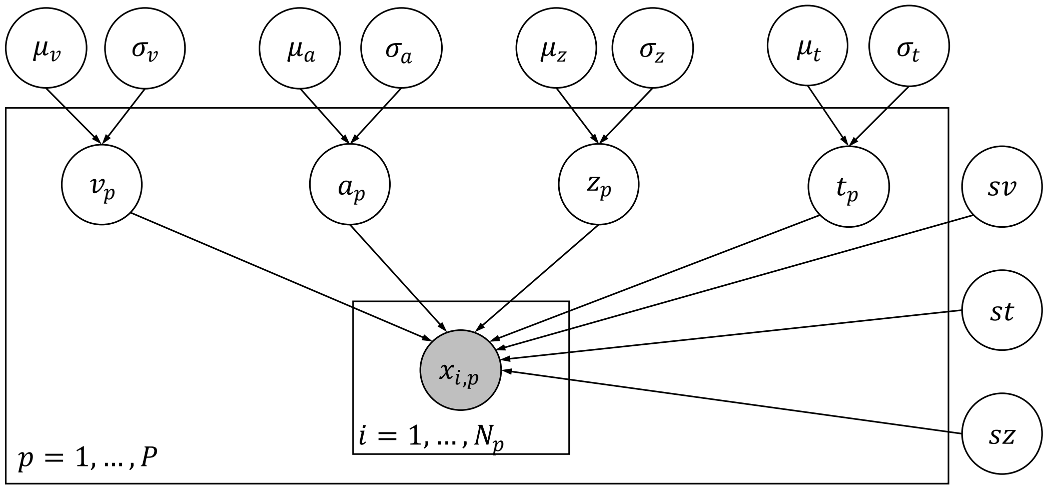


Figure | . The hierarchical structure of the full DDM in HDDM. The parameters inside and outside the rectangle are subject and population level parameters, respectively. p/i are the indices of participants ($p = 1, 2, \dots, P$) and trials ($i = 1, 2, \dots . N$), where $x_{i,p}$ is the data (choice/reaction time) of the i-th trial in the *p*-th subject.

HDDM also allows parameters to vary with variables by integrating hierarchical linear regression models (also called linear mixed models or multi-level models). Specifically, the `` `hddm.HDDMRegressor()` `` function allows any or all of the four parameters of DDM (*a*, *v*, *t*, *z*) to be modelled as a function of experimental conditions or other variables (e.g., EEG signal). In HDDM, the regression models are defined using the Python package patsy (see https://patsy.readthedocs.io/en/latest/quickstart.html), which uses the same syntax for defining regression functions as in other commonly used statistical packages. For example, in Model 2 in the main text, we used the expression `` `v ~ 1 + C(conf, Treatment('LC'))` ``, where the term to the left of "~" is the dependent variable and the term to the right of "~" is the regression equation. The term '1' refers to the intercept, which corresponds to the variable $v\_Intercept$ in the output. The term 'C(conf, Treatment('LC'))' indicates the slope coefficient, which corresponds to the variable $v\_C(conf, Treatment('LC'))[T.HC]$. As in other hierarchical regression models, both the intercept and the slope can be estimated at the population level and the subject level (referred to as "fixed effects" and "random effects" or "varying effects" respectively, Johnson et al., 2017; Pedersen & Frank, 2020; Wiecki et al., 2013), depending on how the model is specified. In `hddm.HDDMRegressor()`, the default is hierarchical model with random intercept but no random slope. We need to set ` group_only_regressors=False `to include the random slope (as we did int Model 2).

Model 1 allows the drift rate to vary as a function of the conflict levels (i.e., `` `depends_on={'v': 'conf'}` `` in HDDM). Specifically, Model 1 sets two drift rate variables each for low and high conflict levels at the both population- and individual-level, respectively. Thus, Model 1 has 12 population-level parameters: the mean and standard deviation for a, t, and z; two mean ("v_(LC)" and "v_(HC)") and one standard deviation for v; and three inter-trial variability parameters ($sv/st/sz$). Similarly, at the individual level, there are 5 ($v_{LC}/v_{HC}/t/z/a$) x 14 (subjects) = 70 individual-level parameters. Thus, Model 1 has a total of 82 free parameters.

Note that Model 1 assumes complete independence between high and low conflict levels within subjects. This assumption may be inappropriate because it is likely that a person who responded relatively fast in the "LC" condition will also be responded relatively fast in the "HC" condition and vice versa.

Model 2 was constructed to include correlations between drift rate across conflicting levels. In Model 2, we use a hierarchical regression model with `` `hddm.HDDMRegressor()` `` by using the formula `` `v ~ 1 + C(conf, Treatment('LC'))` `` (see Box 2 and Box 4). This formulation automatically assigns two free parameters, the intercept and slope, to each subject. Thus, there are $5 * 14 = 70$ individual-level parameters in Model 2. Accordingly, Model 2 has four parameters for *v*: "v_Intercept" and "v_Intercept_std" are the mean and standard deviation of the intercept; "v_C(conf)[T.HC]" and "v_C(conf)[T.HC]_std" are the mean and standard deviation of the slope. Therefore, Model 2 has 13 population-level parameters: the mean and standard deviation for a, t, and z; the mean and standard deviation of the slope and the intercept of the regression for v; and three inter-trial variability parameters ($sv/st/sz$). Taken together, Model 2 has a total of 13 + 70 = 83 free parameters.

4.3. Model Fitting

The defined HDDM model allows the MCMC algorithm to be run using the `.sample()` method for model fitting and parameter estimation. The definition and fitting of Model 2 are used here as an example (see <Code Block 2>):

<Code Block 2>

~~~
```Python
# define a model by hddm.HDDMRegressor
m2 = hddm.HDDMRegressor(
        df, 'v ~ C(conf, Treatment('LC'))',
~~~

```
        group_only_regressors = False,
        keep_regressor_trace = True,
        include=['a', 'v', 't', 'z', 'sv', 'st', 'sz'])
# fitting model and return InferenceData
m2_infdata = m2.sample(
        10000, chains = 4, save_name = 'm2',
        loglike = True, ppc = True, return_infdata = True)
```

To accurately estimate parameters and ensure convergence in hierarchical modeling, we set up four MCMC chains of 10,000 samples with 5,000 burn-ins (i.e., a total of 20,000 samples for each parameter). Please refer to Section 3 for the more detailed settings and arguments description.

With the new functionality introduced by dockerHDDM, we can calculate the log-likelihood of the model and generate posterior predictions after model fitting. Furthermore, the output of the model fitting can be converted into InferenceData, `m2_infdata`, for subsequent analyses as described in Section 3.

4.4. Model Diagnosis

In Bayesian inference, it is crucial to ensure the convergence of MCMC chains. With ArivZ, dockerHDDM supports both visual inspection and quantitative convergence checks (see Section 2.4 in Martin et al., 2021).

`az.plot_trace()` can be used to visualize the posterior distributions of parameters (i.e., trace plots of the MCMC, Figure 5A).

The Gelman-Rubin statistics ($\hat{\boldsymbol{R}}$), and effective sample size (ESS) provide quantitative measures (see Box 1).

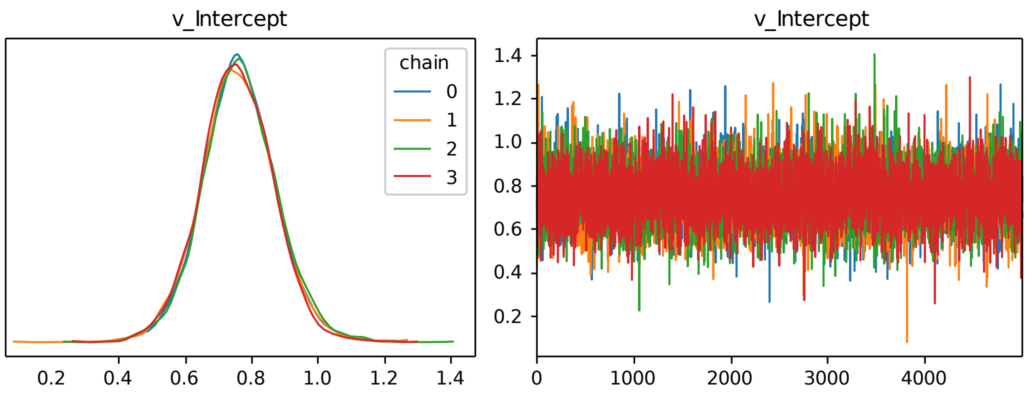
`az.rhat()`computes $\hat{\boldsymbol{R}}$, which should be close to 1 for good convergence; values below 1.01 are typically recommended (Gelman & Rubin, 1992).

`az.ess()` calculates ESS, a measure of the precision of posterior estimates. If the ESS-bulk is over 400, the distribution's center is well-resolved, and we should ensure high ESS across all regions of the parameter space (Martin et al., 2021; Vehtari et al., 2021).

The latter two methods are covered by ArviZ's `az.summary()` (Figure 5B).

A

```
axes = az.plot_trace(
           m2_infdata, var_names=["v_Intercept"],
           compact = False, legend=True, figsize = [10, 3])
```



B

```
summary_tmp = az.summary(
           m2_infdata, kind = "diagnostics", round_to=4)
summary_tmp.sort_values('r_hat', ascending=False).head(10)
```

	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
z_subj.10	0.0007	0.0005	559.6614	1508.8663	1.0137
z_std	0.0017	0.0012	231.4752	162.3132	1.0118
z_subj.9	0.0005	0.0004	720.1698	1515.1079	1.0088
z_subj.5	0.0005	0.0004	690.6138	1399.5685	1.0082
z_subj.6	0.0005	0.0004	800.6655	2042.2894	1.0081
z	0.0004	0.0003	507.1598	1668.9164	1.0050
z_subj.8	0.0004	0.0003	931.8502	3861.8206	1.0050
z_subj.12	0.0004	0.0003	973.4637	4366.5704	1.0050
z_subj.1	0.0005	0.0003	936.9064	4326.4901	1.0043
v_Intercept_subj.10	0.0019	0.0014	2738.3745	6198.5123	1.0042

Figure 5. Model diagnosis. (A) Visualization of the traces of all chains using `az.plot_trace()`, with the argument `var_names` set to focus on the parameter "V_Intercept" as an example. `compact=False` and `legend=True` ensured that the individual traces of each chain would be visible. The MCMC chains are valid and reliable when they fluctuate around a value and different chains are indistinguishable from each other, a scenario often referred to as a "caterpillar" shape. (B) Output of `az.summary()`, which includes the mean and standard deviation of the Monte Carlo standard error (MCSE), the effective sample sizes (bulk-ESS and tail-ESS), and $\hat{R}$. Note that the summary data frame has been sorted by $\hat{R}$ so that we can easily compare the minimum and maximum values of $\hat{R}$.

4.5. Model Comparison

Upon verifying chain convergence, we proceed with model comparison to identify the best-fitting model. The evaluation metric provided in the original HDDM is deviance information criterion

(DIC, Spiegelhalter, Best, Carlin, & Linde, 2002). We include two more methods in dockerHDDM: widely applicable information criterion (WAIC, Watanabe, 2010) and Pareto-smoothed importance sampling leave-one-out cross-validation (PSIS-LOO-CV, Vehtari, Gelman, & Gabry, 2017). These methods comprehensively integrate posterior samples for model comparison and evaluation (see Box 3).

Box 3. Linking DIC, WAIC, and PSIS-LOO-CV to AIC

The Deviance Information Criterion (DIC), Widely Applicable Information Criterion (WAIC), and Pareto-Smoothed Importance Sampling Leave-One-Out Cross-Validation (PSIS-LOO-CV) are criteria founded on the concept of out-of-sample predictive accuracy, i.e., the accuracy of using the fitted model to predict new data generated by the assumed data-generating process. Predictive accuracy is often encapsulated by the *log predictive density* (Box 1). However, the log predictive density approximated using the observed data and the posterior estimates of parameters is a biased, an adjustment is required to correct the bias. Thus, the key difference between DIC, WAIC and PSIS-LOO-CV lies in the difference between the two terms of log predicted density and corrected bias (see the table below).

DIC uses the Bayesian posterior means for estimating log predictive density and includes an adjustment based on the effective number of parameters (P_{DIC}). It is particularly suited for hierarchical models, offering an improved estimate of predictive density (Spiegelhalter, Best, Carlin, & Van Der Linde, 2002).

WAIC further refines DIC, evaluating the log predictive density across the entire posterior and correcting bias via the variability of log predictive density ($\hat{p}_{WAIC}$). This adjustment is crucial for measuring model robustness and guarding against overfitting (Watanabe, 2010).

PSIS-LOO-CV estimates the predictive density by simulating the leave-one-out cross-validation, which by definition is the out-of-sample predictive accuracy, so bias correction is no longer needed for PSIS-LOO-CV. Please see Gelman, Hwang, & Vehtari (2014) and Vehtari, Gelman, & Gabry (2017) for more details on these three indices.

	Predictive accuracy	Adjustment	Formula
AIC	$log\, p(y \mid \hat{\theta}_{mle})$	k	$-2\,(log\, p(y \mid \hat{\theta}_{mle}) - k)$
DIC	$log\, p(y \mid \widehat{\theta_{Bayes}})$	P_{DIC}	$-2\,\left(log\, p(y \mid \widehat{\theta_{Bayes}}) - P_{DIC}\right)$
WAIC	$\widehat{lpd}$	$\hat{p}_{WAIC}$	$-2\,(\widehat{lpd} - \hat{p}_{WAIC})$
PSIS-LOO-CV	$\widehat{elpd}_{psis-loo}$	N.A.	$-2\; elpd_{psis-loo}$

Note: $\widehat{lpd}$, computed log pointwise predictive density, see Glossary for details; $\widehat{elpd}_{psis-loo}$ expected log pointwise predictive density for a new dataset based on PSIS-LOO method. k represents the count of model parameters. P_{DIC} is the DIC's adjustment for the effective number of parameters (Spiegelhalter, Best, Carlin, & Van Der Linde, 2002). $\hat{p}_{WAIC}$ is the WAIC's approach to adjusting the effective number of parameters (Watanabe, 2010).

For the demonstration, we compared three models across all three evaluation metrics (lower value is better)[5]. As shown in Table 4, Model 2 exhibits the lowest values on all three

[5] DIC can be extracted directly from the model rather than InferenceData, e.g. `m0.dic`.

metrics, indicating it is the best model. The results of model comparison revealed that Models 1 and 2 are much better than the baseline Model 0, suggesting that experimental conflict conditions have a substantial effect on drift rates. Moreover, Models 2 is slightly better than Model 1, suggesting that regression model may suit the data better. Nevertheless, the similarities between Model 1 and Model 2 suggests that both models fit the data adequately in this case.

Table 4. Model comparison with different criteria.

Rank*	DIC	PSIS-LOO-CV	WAIC
1	m2 (10654.89)	m2 (10646.25)	m2 (10646.20)
2	m1 (10655.24)	m1 (10647.21)	m1 (10647.15)
3	m0 (10835.24)	m0 (10824.93)	m0 (10824.89)

* Rank is from the best model to the worst. Models 0 to 2 are referred to as m0 to m2.

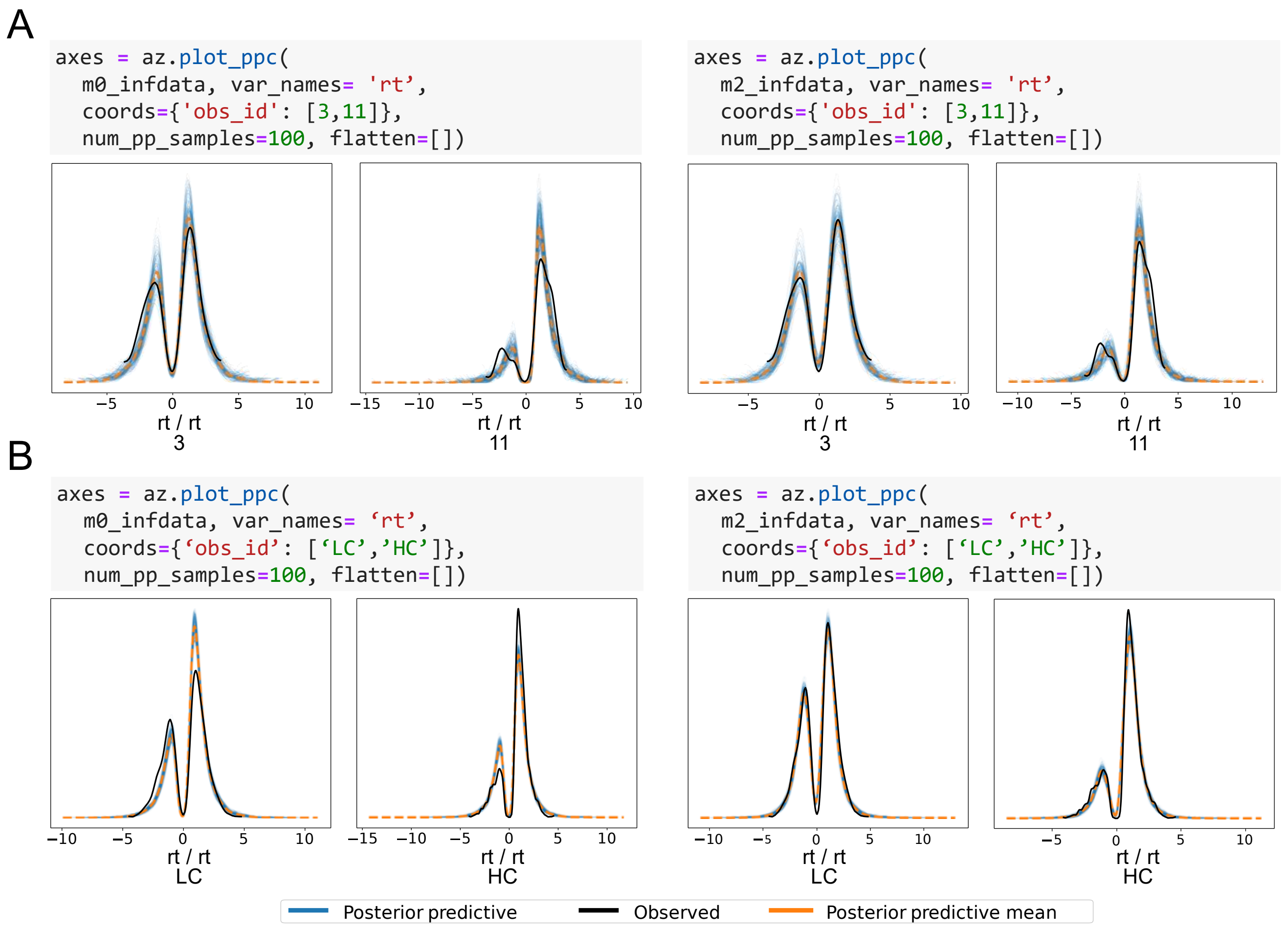
Note that WAIC and PSIS-LOO-CV require the pointwise log-likelihood of each data point given a posterior sample of parameters, which must be computed using the likelihood function and posterior trace (see Box 3). This variable is not directly provided in the HDDM object and must be customized to be computed via the likelihood function and the posterior trace.

In dockerHDDM, the *pointwise log-likelihood* can be computed at sampling and fitting stage, via `m.sample(... , retutn_infdata = True, loglike = True)` (see <Code Block 2>), or after the model has been sampled and fitted, by `m.to_infdata(loglike = True)`. Both ways return InferenceData, allowing users to immediately compute WAIC and PSIS-LOO-CV. After that, the evaluation metrics for each model's InferenceData are available using ArviZ's `compare` method (see <Code Block 3>), which returns the results of WAIC for the argument `ic="waic"` or PSIS-LOO-CV for `ic="loo"`.

<Code Block 3>

```Python
compare_dict = {
        'm0': m0_infdata,
        'm1': m1_infdata,
        'm2': m2_infdata
}
az.compare(compare_dict, ic = 'loo')
```

Finally, it's important to note that the model comparison metrics only allow us a *relative* ranking of alternatives. To assess the *absolute* goodness-of-fit of the model, we recommend performing the posterior predictive check (PPC), as discussed in the next section, alongside the diagnostic information provided by LOO and WAIC (see Martin et al., 2021).



A

```
axes = az.plot_ppc(
  m0_infdata, var_names= 'rt',
  coords={'obs_id': [3,11]},
  num_pp_samples=100, flatten=[])
```

```
axes = az.plot_ppc(
  m2_infdata, var_names= 'rt',
  coords={'obs_id': [3,11]},
  num_pp_samples=100, flatten=[])
```

B

```
axes = az.plot_ppc(
  m0_infdata, var_names= 'rt',
  coords={'obs_id': ['LC','HC']},
  num_pp_samples=100, flatten=[])
```

```
axes = az.plot_ppc(
  m2_infdata, var_names= 'rt',
  coords={'obs_id': ['LC','HC']},
  num_pp_samples=100, flatten=[])
```

Figure 6. Posterior predictive check plot `az.plot_ppc()` for Model 0 "m0" and Model 2 "m2". Solid black lines are the density plot of the observed RT data; blue lines are the posterior predictive samples, each line represents the predicted RT distribution based on one posterior predictive sample; yellow dashed lines represent the mean of all predicted RT distributions across all posterior predictive samples. (A) shows the results of the comparison between the two models (m0 vs. m2) at the individual level (subjects 3 and 11 as an example); (B) shows the results of the comparison at the condition level (i.e., "LC" represents lower conflict and "HC" represents higher conflict). All plots in the left column are for m0 and all plots in the right column are for m2. Note that the argument `coords` specifies the PPC level (individual or group level) that should be preprocessed before plotting. `num_pp_samples` is used to set the number of predictive data required for plotting.

4.6. Posterior Predictive Check

In addition to model comparison, which assesses relative performance, the posterior predictive check (PPC) evaluates how well synthetic data generated from posterior samples of parameters align with the actual data. PPC is crucial because model comparison only evaluates the "least worst" model, not necessarily selects the one that can account for the data very well (see Pedersen, Frank, & Biele, 2017; Steingroever, Wetzels, & Wagenmakers, 2014).

ArviZ offers convenient visualization tools for inspecting PPC (see section 2.3 in Martin et al., 2021). The function `` `az.plot_ppc()` `` is helpful to visualize PPC at the individual or condition level (Figure 6). In the demonstration, the synthetic data from Model 2 match more closely the actual data compared to the baseline Model 0, and this difference becomes apparent when examining PPC at the individual- (Figure 6A) and condition-level (Figure 6B).

4.7. Statistical Inference

A final step in Bayesian modeling is to draw statistical inferences from the posterior parameter distributions in the best-fitting model. In our example, we will test the hypothesis whether drift rates significantly differ between high and low conflict conditions based on Model2 ("m2" in the Notebook). This hypothesis will be tested using the posterior samples of the regression coefficient in "m2", which has a variable name "v_C(conf, Treatment('LC'))[T.HC]".

Note that there are several acceptable methods for Bayesian hypothesis testing, such as probability of direction, Bayes factor, and *Maximum a posteriori* (*MAP*) based *p*-value (Makowski et al., 2019). Here we adopted the approach combining *Highest Density Interval* (HDI) and the *Region of Practical Equivalence* (ROPE, Kruschke, 2018) (see Box 1).

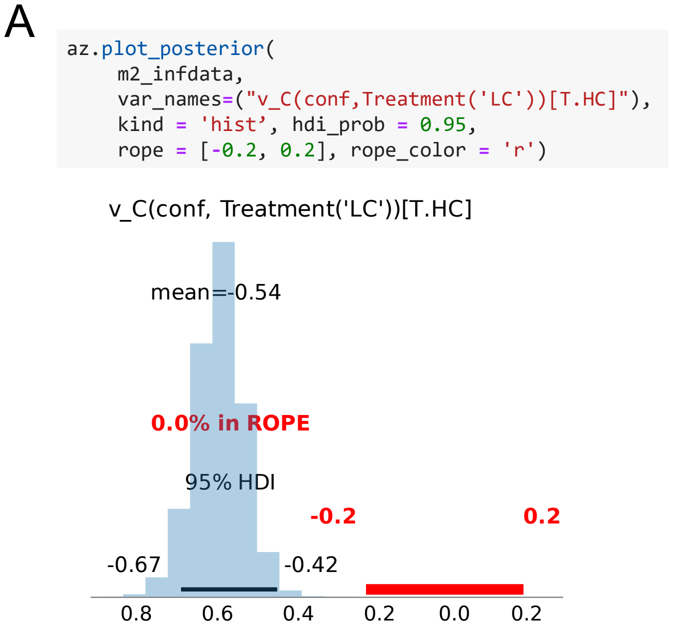
We define a ROPE of [-0.2, 0.2] to represent values practically equivalent to zero[6] and use `` `plot_posterior()` `` function from ArviZ to implement ROPE test. By comparing the 95% HDI of the regression coefficient to this ROPE, we find that the HDI falls completely outside the

[6] The ROPE should be tailored to the specific paradigm and research question (Dienes, 2021) and reflect the range of possible values for each parameter (e.g., Tran et al., 2021). For example, a recent systematic parameter review of DDM found that the absolute value of a drift rate ranged from 0.01 to 18.51, with a median of 2.25 (Tran et al., 2021); another simulation and meta-analysis of conflict tasks showed that a drift rate between 0.05 and 0.35 captured the conflict effect (Hedge et al., 2018). Accordingly, we choose ROPE [-0.2 0.2] for illustrative purposes, implying that effects on drift rates smaller than 0.2 are not of interest.

ROPE (Figure 7A), suggesting that the drift rate is higher in the low conflict condition than the high conflict condition (Figure 7B).

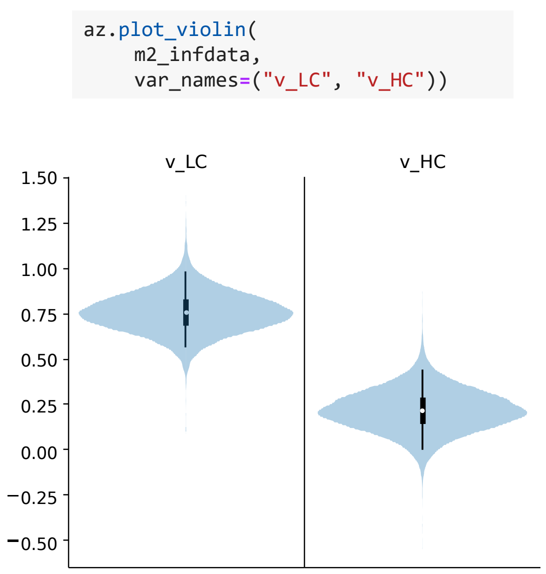
Therefore, considering the results from various aspects (model comparison, ppc, and posterior inference), we conclude that the model which takes into account the influence of conflict level on drift rate performs the best. Moreover, high conflict affects the cognitive process of decision-making by impeding the speed of evidence accumulation.

A



```
az.plot_posterior(
    m2_infdata,
    var_names=("v_C(conf,Treatment('LC'))[T.HC]"),
    kind = 'hist’, hdi_prob = 0.95,
    rope = [-0.2, 0.2], rope_color = 'r')
```

B



```
az.plot_violin(
    m2_infdata,
    var_names=("v_LC", "v_HC"))
```

Figure 7. (A) Statistical inference of parameters. The high-density interval (HDI, black line and texts) is compared with the region of practical equivalence (ROPE, red line and text). `var_names` argument can be used to select both group-level and individual-level parameters for analysis. `hdi_prob ` argument specifies the probability of the HDI, typically set at 0.95 to correspond to a 95% confidence interval. `rope` defines the limitations of ROPE, which is a range considered to be equivalent to the null hypothesis or a reference value for the parameter. The results show no overlap between the 95% HDI and the ROPE, indicating that the parameter is credibly different from zero. (B) Violin plot of parameter posteriors at two conflict levels. The black line is the 95% HDI and the white dot is the mean. The drift rate is lower in high conflict (HC) than in low conflict (LC) conditions.

Box 4. Recommendation for Further Reading

A full understanding of how Bayesian hierarchical drift-diffusion modeling works requires not only basic knowledge of DDM, but also knowledge of Python programming, Bayesian statistics, and hierarchical regression models. This background knowledge is generally not part of the coursework in psychology or neuroscience education, although the situation is changing in recent years (e.g., Hart et al., 2022). We recommend the following resources to quickly catch up and avoid misuse or abuse of HDDM.

Background knowledge/skills	Resource
Bayesian statistics	Etz & Vandekerckhove, 2018; Kruschke & Liddell, 2018; Lambert, 2018; Martin, Kumar, & Lao, 2021; van de Schoot et al., 2021.
(Bayesian) Hierarchical (regression) models	https://twiecki.io/blog/2014/03/17/bayesian-glms-3/; https://github.com/lei-zhang/BayesCog_Wien Capretto et al., 2020
Computational modeling	Blohm, Kording, & Schrater, 2020; Wilson & Collins, 2019; Zhang, Lengersdorff, Mikus, Gläscher, & Lamm, 2020.
Drift Diffusion Models	Ratcliff & McKoon, 2008; Voss, Nagler, & Lerche, 2013.
Sequential sampling models beyond DDMs	Fengler, Bera, Pedersen, & Frank, 2022; Ratcliff et al., 2016.

Summary

In this article, we introduce dockerHDDM, a user-friendly, out-of-the-box, and one-stop Docker image for implementing HDDM analysis within a modern Bayesian hierarchical workflow. Our dockerHDDM has three major advantageous: (1) it leverages Docker to solve compatibility issues and simplify the installation process; (2) it ensures broad support across different machines equipped with either Intel or Apple chips; and (3) it integrates state-of-the-art Bayesian modeling practices with ArviZ, facilitating a more principled Bayesian workflow. We also provide a step-by-step tutorial on how to implement HDDM using dockerHDDM. As HDDM continues to advance, with recent developments including reinforcement learning DDM (Pedersen & Frank, 2020; Pedersen, Frank, & Biele, 2017) and likelihood approximation networks (LANs, Fengler et al., 2022, 2021), dockerHDDM will serve as a critical tutorial for computational reproducibility for published studies. Given the extensive knowledge required for principled computational modelling, we recommend readers go through materials in Box 4 for a deeper understanding of the DDM family, cognitive modeling, hierarchical models, and Bayesian modeling. We expect that dockerHDDM and this detailed tutorial will reduce the technical burden and promote the computational reproducibility of drift-diffusion modeling for users of all levels of computational expertise.

Code Availability

The software, data, and scripts (Jupyter notebooks) used to generate the models and results described in this article are available at https://hub.docker.com/r/hcp4715/hddm. Readers can pull the entire image from Docker hub after successfully installing Docker desktop (for MacOS and Windows) or Docker engine (for Linux) and using the following code in a terminal (Linux or MacOS) or Windows (power) shell:

```
`docker pull hcp4715/hddm`
```

Alternatively, readers can find our online notebook here: https://github.com/hcp4715/dockerHDDM/. Readers can also find the code that created our dockerHDDM images at https://github.com/hcp4715/dockerHDDM/dockerfiles/. Any questions about this tutorial or related dockerHDDM images can be posted and discussed here: https://github.com/hcp4715/dockerHDDM/issues.

Conflict of Interest

The authors declare no competing financial interests.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (32100901) and Natural Science Foundation of Shanghai (21ZR1434700) to R-Y. Z.

Author Contributions

H. C-P., H.G., L.Z., and R-Y.Z. conceived and designed the study. W.P. & H. C-P. implemented and maintain the dockerHDDM Docker image. H. C-P., H.G., and R-Y.Z. made the first draft of the manuscript. W.P., H. C-P., and R-Y.Z. re-organized the draft since version 7 of the preprint. All authors edited the manuscript.

References

Boehm, U., Annis, J., Frank, M. J., Hawkins, G. E., Heathcote, A., Kellen, D., Krypotos, A.-M., Lerche, V., Logan, G. D., Palmeri, T. J., van Ravenzwaaij, D., Servant, M., Singmann, H., Starns, J. J., Voss, A., Wiecki, T. V., Matzke, D., & Wagenmakers, E.-J. (2018). Estimating across-trial variability parameters of the Diffusion Decision Model: Expert advice and recommendations. *Journal of Mathematical Psychology*, *87*, 46–75. https://doi.org/10.1016/j.jmp.2018.09.004

Cavanagh, J. F., Wiecki, T. V., Cohen, M. X., Figueroa, C. M., Samanta, J., Sherman, S. J., & Frank, M. J. (2011). Subthalamic nucleus stimulation reverses mediofrontal influence over decision threshold. *Nature Neuroscience*, *14*(11), 1462–1467. https://doi.org/10.1038/nn.2925

Desai, N., & Krajbich, I. (2022). Decomposing preferences into predispositions and evaluations. *Journal of Experimental Psychology: General*, *151*(8), 1883–1903. https://doi.org/10.1037/xge0001162

Dienes, Z. (2021). Obtaining Evidence for No Effect. *Collabra: Psychology*, *7*(1), 28202. https://doi.org/10.1525/collabra.28202

Evans, N. J., & Wagenmakers, E.-J. (2019). Theoretically meaningful models can answer clinically relevant questions. *Brain*, *142*(5), 1172–1175. https://doi.org/10.1093/brain/awz073

Fengler, A., Bera, K., Pedersen, M. L., & Frank, M. J. (2022). Beyond Drift Diffusion Models: Fitting a Broad Class of Decision and Reinforcement Learning Models with HDDM. *Journal of Cognitive Neuroscience*, *34*(10), 1780–1805. https://doi.org/10.1162/jocn_a_01902

Fengler, A., Govindarajan, L. N., Chen, T., & Frank, M. J. (2021). Likelihood approximation networks (LANs) for fast inference of simulation models in cognitive neuroscience. *eLife*, *10*, e65074. https://doi.org/10.7554/eLife.65074

Forstmann, B. U., Ratcliff, R., & Wagenmakers, E.-J. (2016). Sequential Sampling Models in Cognitive Neuroscience: Advantages, Applications, and Extensions. *Annual Review of Psychology*, *67*(1), 641–666. https://doi.org/10.1146/annurev-psych-122414-033645

Gelman, A., & Rubin, D. B. (1992). Inference from Iterative Simulation Using Multiple Sequences. *Statistical Science*, *7*(4). https://doi.org/10.1214/ss/1177011136

Gelman, A., Vehtari, A., Simpson, D., Margossian, C. C., Carpenter, B., Yao, Y., Kennedy, L., Gabry, J., Bürkner, P.-C., & Modrák, M. (2020). Bayesian Workflow. *arXiv:2011.01808 [Stat]*. http://arxiv.org/abs/2011.01808

Ging-Jehli, N. R., Ratcliff, R., & Arnold, L. E. (2021). Improving neurocognitive testing using computational psychiatry—A systematic review for ADHD. *Psychological Bulletin*, *147*(2), 169–231. https://doi.org/10.1037/bul0000319

Hedge, C., Powell, G., Bompas, A., Vivian-Griffiths, S., & Sumner, P. (2018). Low and variable correlation between reaction time costs and accuracy costs explained by accumulation models: Meta-analysis and simulations. *Psychological Bulletin*, *144*(11), 1200–1227. https://doi.org/10.1037/bul0000164

Herz, D. M., Tan, H., Brittain, J.-S., Fischer, P., Cheeran, B., Green, A. L., FitzGerald, J., Aziz, T. Z., Ashkan, K., Little, S., Foltynie, T., Limousin, P., Zrinzo, L., Bogacz, R., & Brown, P. (2017). Distinct mechanisms mediate speed-accuracy adjustments in cortico-subthalamic networks. *eLife*, *6*, e21481. https://doi.org/10.7554/eLife.21481

Herz, D. M., Zavala, B. A., Bogacz, R., & Brown, P. (2016). Neural Correlates of Decision Thresholds in the Human Subthalamic Nucleus. *Current Biology*, *26*(7), 916–920. https://doi.org/10.1016/j.cub.2016.01.051

Hoyer, S., & Hamman, J. (2017). xarray: N-D labeled Arrays and Datasets in Python. *Journal of Open Research Software*, *5*(1), Article 1. https://doi.org/10.5334/jors.148

Hu, C.-P., Lan, Y., Macrae, C. N., & Sui, J. (2020). Good Me Bad Me: Does Valence Influence Self-Prioritization During Perceptual Decision-Making? In *Collabra-Psychology* (Vol. 6, Issue 1, p. 20). https://doi.org/10.1525/collabra.301

Johnson, D. J., Hopwood, C. J., Cesario, J., & Pleskac, T. J. (2017). Advancing Research on Cognitive Processes in Social and Personality Psychology: A Hierarchical Drift Diffusion Model Primer. *Social Psychological and Personality Science*, *8*(4), 413–423. https://doi.org/10.1177/1948550617703174

Kruschke, J. K. (2018). Rejecting or Accepting Parameter Values in Bayesian Estimation. *Advances in Methods and Practices in Psychological Science*, *1*(2), 270–280. https://doi.org/10.1177/2515245918771304

Kruschke, J. K. (2021). Bayesian Analysis Reporting Guidelines. *Nature Human Behaviour*, *5*, 1282–1291. https://doi.org/10.1038/s41562-021-01177-7

Kumar, R., Carroll, C., Hartikainen, A., & Martin, O. (2019). ArviZ a unified library for exploratory analysis of Bayesian models in Python. *Journal of Open Source Software*, *4*(33), 1143. https://doi.org/10.21105/joss.01143

Kutlikova, H. H., Zhang, L., Eisenegger, C., van Honk, J., & Lamm, C. (2023). Testosterone eliminates strategic prosocial behavior through impacting choice consistency in healthy males. *Neuropsychopharmacology*, *48*(10), Article 10. https://doi.org/10.1038/s41386-023-01570-y

Makowski, D., Ben-Shachar, M. S., Chen, S. H. A., & Lüdecke, D. (2019). Indices of effect existence and significance in the bayesian framework. *Frontiers in Psychology*, *10*, 2767. https://doi.org/10.3389/fpsyg.2019.02767

Martin, O. A., Kumar, R., & Lao, J. (2021). *Bayesian Modeling and Computation in Python*. Chapman and Hall/CRC. https://doi.org/10.1201/9781003019169

Pedersen, M. L., & Frank, M. J. (2020). Simultaneous Hierarchical Bayesian Parameter Estimation for Reinforcement Learning and Drift Diffusion Models: A Tutorial and Links to Neural Data. *Computational Brain & Behavior*, *3*, 458–471. https://doi.org/10.1007/s42113-020-00084-w

Pedersen, M. L., Frank, M. J., & Biele, G. (2017). The drift diffusion model as the choice rule in reinforcement learning. *Psychonomic Bulletin & Review*, *24*(4), 1234–1251. https://doi.org/10.3758/s13423-016-1199-y

Pedersen, M. L., Ironside, M., Amemori, K., McGrath, C. L., Kang, M. S., Graybiel, A. M., Pizzagalli, D. A., & Frank, M. J. (2021). Computational phenotyping of brain-behavior dynamics underlying approach-avoidance conflict in major depressive disorder. *PLOS Computational Biology*, *17*(5), e1008955. https://doi.org/10.1371/journal.pcbi.1008955

Peikert, A., & Brandmaier, A. M. (2021). A Reproducible Data Analysis Workflow With R Markdown, Git, Make, and Docker. *Quantitative and Computational Methods in Behavioral Sciences*, *1*, e3763. https://doi.org/10.5964/qcmb.3763

Ratcliff, R., & Rouder, J. N. (1998). Modeling Response Times for Two-Choice Decisions. *Psychological Science*, *9*(5), 347–356. https://doi.org/10.1111/1467-9280.00067

Ratcliff, R., Smith, P. L., Brown, S. D., & McKoon, G. (2016). Diffusion Decision Model: Current Issues and History. *Trends in Cognitive Sciences*, *20*(4), 260–281. https://doi.org/10.1016/j.tics.2016.01.007

Ratcliff, R., & Tuerlinckx, F. (2002). Estimating parameters of the diffusion model: Approaches to dealing with contaminant reaction times and parameter variability. *Psychonomic Bulletin & Review*, *9*(3), 438–481. https://doi.org/10.3758/bf03196302

Shadlen, M. N., & Shohamy, D. (2016). Decision Making and Sequential Sampling from Memory. *Neuron*, *90*(5), 927–939. https://doi.org/10.1016/j.neuron.2016.04.036

Sheng, F., Ramakrishnan, A., Seok, D., Zhao, W. J., Thelaus, S., Cen, P., & Platt, M. L. (2020). Decomposing loss aversion from gaze allocation and pupil dilation. *Proceedings of the National Academy of Sciences*, *117*(21), 11356–11363. https://doi.org/10.1073/pnas.1919670117

Spiegelhalter, D. J., Best, N. G., Carlin, B. P., & Linde, A. V. D. (2002). Bayesian measures of model complexity and fit. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *64*(4), 583–639. https://doi.org/10.1111/1467-9868.00353

Steingroever H., Wetzels R., & Wagenmakers E.-J. (2014). Absolute performance of reinforcement-learning models for the iowa gambling task. *Decision*, *1*(3), 161–183. https://doi.org/10.1037/dec0000005

Tran, N.-H., Van Maanen, L., Heathcote, A., & Matzke, D. (2021). Systematic Parameter Reviews in Cognitive Modeling: Towards a Robust and Cumulative Characterization of Psychological Processes in the Diffusion Decision Model. *Frontiers in Psychology*, *11*, 608287. https://doi.org/10.3389/fpsyg.2020.608287

van de Schoot, R., Depaoli, S., King, R., Kramer, B., Märtens, K., Tadesse, M. G., Vannucci, M., Gelman, A., Veen, D., Willemsen, J., & Yau, C. (2021). Bayesian statistics and modelling. *Nature Reviews Methods Primers*, *1*(1), Article 1. https://doi.org/10.1038/s43586-020-00001-2

Vehtari, A., Gelman, A., & Gabry, J. (2017). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*, *27*(5), 1413–1432. https://doi.org/10.1007/s11222-016-9696-4

Vehtari, A., Gelman, A., Simpson, D., Carpenter, B., & Bürkner, P.-C. (2021). Rank-Normalization, Folding, and Localization: An Improved Rˆ for Assessing Convergence of MCMC (with Discussion). *Bayesian Analysis*, *16*(2), 667–718. https://doi.org/10.1214/20-BA1221

Voss, A., Nagler, M., & Lerche, V. (2013). Diffusion models in experimental psychology: A practical introduction. *Experimental Psychology*, *60*(6), 385–402. https://doi.org/10.1027/1618-3169/a000218

Watanabe, S. (2010). Asymptotic Equivalence of Bayes Cross Validation and Widely Applicable Information Criterion in Singular Learning Theory. *The Journal of Machine Learning Research*, *11*, 3571–3594.

Wiebels, K., & Moreau, D. (2021). Leveraging Containers for Reproducible Psychological Research. *Advances in Methods and Practices in Psychological Science*, *4*(2). https://doi.org/10.1177/25152459211017853

Wiecki, T. V., Sofer, I., & Frank, M. J. (2013). HDDM: Hierarchical Bayesian estimation of the Drift-Diffusion Model in Python. *Frontiers in Neuroinformatics*, *7*. https://doi.org/10.3389/fninf.2013.00014

Zhang, L., Lengersdorff, L., Mikus, N., Gläscher, J., & Lamm, C. (2020). Using reinforcement learning models in social neuroscience: Frameworks, pitfalls, and suggestions of best practices. *Social Cognitive and Affective Neuroscience*, *15*(6), 695–707. https://doi.org/10.1093/scan/nsaa089