Behavior Research Methods
https://doi.org/10.3758/s13428-023-02162-w

PyBEAM: A Bayesian approach to parameter inference for a wide class of binary evidence accumulation models

Matthew Murrow[1] · William R. Holmes[2]

Accepted: 3 June 2023

Abstract
Many decision-making theories are encoded in a class of processes known as evidence accumulation models (EAM). These assume that noisy evidence stochastically accumulates until a set threshold is reached, triggering a decision. One of the most successful and widely used of this class is the Diffusion Decision Model (DDM). The DDM however is limited in scope and does not account for processes such as evidence leakage, changes of evidence, or time varying caution. More complex EAMs can encode a wider array of hypotheses, but are currently limited by computational challenges. In this work, we develop the Python package PyBEAM (Bayesian Evidence Accumulation Models) to fill this gap. Toward this end, we develop a general probabilistic framework for predicting the choice and response time distributions for a general class of binary decision models. In addition, we have heavily computationally optimized this modeling process and integrated it with PyMC, a widely used Python package for Bayesian parameter estimation. This 1) substantially expands the class of EAM models to which Bayesian methods can be applied, 2) reduces the computational time to do so, and 3) lowers the entry fee for working with these models. Here we demonstrate the concepts behind this methodology, its application to parameter recovery for a variety of models, and apply it to a recently published data set to demonstrate its practical use.

Keywords Markov chain monte carlo · Python · Changing thresholds · Likelihood approximation · Fokker planck

Introduction

Computational modeling has been used to study the properties of decision making for over 50 years. Given the difficulty in both perturbing and observing the brain, models are vital for formally encoding and testing mechanistic hypotheses about decision making processes. For example, how do people process information over time, how do they modulate their levels of caution under different circumstances, or how do they extract information from complex choice sets with multiple alternatives and attributes. These types of questions are difficult to study through direct observation or statistical analysis alone. Models provide a rigorous way to study these types of questions indirectly by comparing the expected patterns of data predicted by the models to experimental observation.

One of the dominant classes of models in this area are Evidence Accumulation Models (EAMs). These models predict the outcome of decisions by modeling the process by which that decision is made. For example, the popular Diffusion Decision Model (DDM) (Ratcliff & McKoon, 2008) hypothesizes that people stochastically sample information over time, additively accumulate evidence based on that information, and make a decision when a critical threshold of evidence has been achieved. This and similar models are mathematically encoded in stochastic differential equations (SDE). A useful feature of this family of models is that they make predictions about both the choices made and the time it takes them to make choices, so-called Choice-Response Time (choice-RT) data. Importantly, the time required to complete the decision provides insight into the underlying decision process. For example, one might expect a strong preference for one alternative to yield a fast decision.

EAMs are a highly successful modeling framework in cognitive psychology. They both qualitatively and quan-

✉ William R. Holmes
wrholmes@iu.edu

[1] Department of Physics and Astronomy, Vanderbilt University, 6301 Stevenson Science Center, Nashville 37212, TN, USA

[2] Cognitive Science Program and Department of Mathematics, Indiana University, 1001 E. 10th St., Bloomington 47405, IN, USA

Published online: 07 August 2023

titatively capture a range of choice and response time benchmarks, including (1) speed-accuracy trade-off, (2) the positive skew of human response time distributions, (3) the relation between the mean and variance in response times, and (4) differences in fast and slow errors (Usher & McClelland, 2001; Ratcliff, Zandt, & McKoon, 1999; Ratcliff & Rouder, 1998; Brown & Heathcote, 2008; Ratcliff, 1978). These models have also been applied to a wide range of behaviors including learning (Evans, Brown, Mewhort, & Heathcote, 2018; Fontanesi, Gluth, Spektor, & Rieskamp, 2019), categorization (Nosofsky & Palmeri, 1997; Nosofsky, Little, Donkin, & Fific, 2011), memory (Ratcliff, 1978; Osth & Farrell, 2019), language processing (Wagenmakers, Ratcliff, Gomez, & McKoon, 2008; Lerche, Christmann, & Voss, 2018), and consumer choice (Evans, Holmes, & Trueblood, 2019; Busemeyer, Gluth, Rieskamp, & Turner, 2019). More recently, researchers have started combining EAMs with psychophysiological data, such as neural recordings (Turner, van Maanen, & Forstmann, 2015; Turner, Rodriguez, Norcia, McClure, & Steyvers, 2016; Turner et al., 2013), motor recordings (Servant, White, Montagnini, & Burle, 2016), and eye movements (Krajbich, Armel, & Rangel, 2010). Despite their popularity and success, most applications use only the simplest forms of EAMs, such as the DDM. Tools for quantitatively fitting general EAMs are still lacking.

All current theoretical modeling approaches in this area make some type of significant sacrifice that limits their use. The typical modeling process in this area involves three essential elements: a theoretical model, experimental data to test it, and efficient and accurate methods for challenging the model with data. Existing methods sacrifice at least one of these. Approaches that build around the DDM (e.g. HDDM, etc.) are fast and accurate, but are limited by the assumptions of the DDM. Alternative approaches (discussed in detail below) allow researchers to study broader theoretical models (e.g. time varying caution) or utilize different experimental designs (e.g. probing behavior with time varying stimuli), but at a cost. In these cases, the absence of an easily obtainable likelihood (in most cases) has led to the development of approaches with either documented accuracy issues, poor efficiency that limits experimental design (i.e. too many experimental conditions = too much compute time), or both (more details in the Background). These limitations restrict the questions researchers can ask and the studies they design.

The main challenge of working with EAMs and choice-RT data is to determine, for a particular model and parameter set, how well the model matches the empirical data (Dutilh et al., 2019). The likelihood function is a natural way to do this. Unfortunately, due to the difficulty in working with SDEs, only the simplest models in this family have sufficiently analytic likelihoods to be tractable. We say "sufficiently" because the likelihoods of even the simplest models (e.g. the DDM) involve infinite series or intractable integrals. For example, Buonocore, Giorno, Nobile, and Ricciardi, 1990 developed an analytic approach to solve for the likelihood function of a range of SDE models (summarized in Smith, 2000). Unfortunately, it tends to be slow to compute since it requires numerical calculation of many integrals, causing it to only be used infrequently when fitting data (Evans, Hawkins, & Brown, 2020). Further, numerical integration produces resolution issues similar to that of numerical solutions. For these reasons, most methods utilize some way of approximating the quantitative agreement between model and data and make some sacrifice in the process. Since much of the work on choice-RT model fitting applies to binary choice models, we will limit this brief discussion to binary decision modeling approaches (see Table 1).

Until recently, the majority of quantitative fitting approaches were based on summary statistics (Turner & Van Zandt, 2018). One common approach is the quantile maximization approach (Ratcliff & Tuerlinckx, 2002; Heathcote, Brown, & Mewhort, 2002). For a stochastic model, the user simulates a large number of outcomes, finds the quantiles of the RT distribution, compares those to the quantiles of the experimental RT distribution, and optimizes parameters for agreement. CHaRTr (Chandrasekaran & Hawkins, 2019), a recently developed R modeling package for general binary EAMs, takes this approach. It is well known however that this compression of the model and data into summary statistics can lead to significant errors in parameter estimates for EAMs (Turner & Sederberg, 2014).

A second approach is the Probability Density Approximation (Turner & Sederberg, 2014; Holmes, 2015; Lin, Heathcote, & Holmes, 2019) method. This method starts much the same way by simulating a large number of stochastic outcomes. Those samples are then used to construct a kernel density estimate approximation of the actual likelihood function. This avoids the problems with summary statistics and is easily generalizable to more complex models (Trueblood, Heathcote, Evans, & Holmes, 2021; Turner & Sederberg, 2014; Evans, Holmes, Dasari, & Trueblood, 2021; Holmes, O'Daniels, & Trueblood, 2020; Evans, Trueblood, & Holmes, 2020; Trueblood et al., 2018; Holmes, Trueblood, & Heathcote, 2016). However, the simulated likelihood function becomes a stochastic entity, which introduces significant problems for Bayesian inference (Holmes & Trueblood, 2018). Specifically, a favorable estimate of the likelihood is easily accepted but difficult to reject in favor of a new parameter set, leading to MCMC chain stagnation, high MCMC rejection rates, and generally poor posterior approximations. Additionally, the large numbers of simulations needed are extremely computationally intensive.

A third approach converts the SDE evidence accumulation models into Fokker-Planck (FP) models. This approach essentially reformulates the SDE description of the model

Table 1 List of existing software for fitting two-threshold, binary choice models to choice-RT data with the proposed characteristics of our approach, PyBEAM, in the final column

Package	Approach	Generality	Language
PyDDM	Max Likelihood	General SDE	Python
HDDM	Bayesian	DDM, LBA, LANs	Python
ChaRTr	Quantile Optimization	General SDE	R
DMAT	Quantile Optimization	DDM	Matlab
fast-dm	Max Likelihood, Kolmogorov-Smirnov, Chi-Square	DDM	Command line
DMC	Bayesian	DDM, LBA	R
PyBEAM	Bayesian	General SDE	Python

Programs include PyDDM (Shinn, Lam, & Murray, 2020), HDDM (Wiecki, Sofer, & Frank, 2013), ChaRTr (Chandrasekaran & Hawkins, 2019), DMAT (Vandekerckhove & Tuerlinckx, 2008), fast-DM (Voss & Voss, 2007), and DMC (Heathcote et al., 2019)

as a probabilistic model described by a Partial Differential Equation (PDE). Solving this PDE then provides an approximation for the likelihood. This was first implemented in a package by Voss and Voss with fast-DM (Voss & Voss, 2007, 2008). Their approach, however, was limited to only fitting DDMs, not the broader class of EAMs. The expansion to EAMs was addressed with the Python package PyDDM (Shinn, Lam, & Murray, 2020). PyDDM takes a similar approach to fast-DM in that it converts SDE models into FP models. Whereas fast-DM uses the "backward FP," PyDDM uses the "forward FP," which readily allows for modification of the threshold and drift rates to increase model flexibility.

These methods are similar to what we propose here, but they have a number of practical drawbacks. fast-DM does not accommodate more complex models involving, for example, time varying thresholds. PyDDM is more general; however, it is a maximum likelihood based approach and their numerical implementation would not scale well for Bayesian estimation. Their maximum likelihood fits can take up to 5 hours (according to their own benchmarks). Since Bayesian methods require sampling far more parameter sets than maximum likelihood (10-100x more), it would be infeasible to directly extend this. Further, more general discussion of a variety of numerical approaches introduced above can be found in Richter, Ulrich, and Janczyk, 2023. This is possibly the most comprehensive comparison of first passage time approximation techniques to date, though they do not address Bayesian implementations.

The fourth and final approach we briefly discuss here are Likelihood Approximation Networks (LAN) (Fengler, Govindarajan, Chen, & Frank, 2021; Fengler, Bera, Pedersen, & Frank, 2022). In the context of parameter estimation, a model is a functional mapping from parameters and input data to a likelihood value. In the LAN approach, a neural network is trained to approximate this. This is accomplished by simulating enormous amounts of choice-RT data from a model to construct a training data set that consists of parameter sets and approximate likelihood functions calculated using, for example, Kernel Density Estimation (KDE). Though this technique has enormous potential going forward, it has a few significant drawbacks. First, the stochastic nature of simulation based data used for training may introduce unknown inaccuracies into the training data itself (Holmes, 2015). From this perspective, the likelihood generation approach we demonstrate in this article could potentially be used to generate higher quality training data for LANs. Second, to utilize an LAN, you must first expend significant computational resources to generate training data from a specific model and experimental structure, then train the LAN to approximate it. For frequently used models, this approach will be useful since the up front cost of training the LAN can be recouped through regular use. However, the up front cost will be prohibitive if the intention is to explore multiple novel models or novel experimental designs. As an example of the importance of experimental design, a LAN trained for experimental trials with fixed information cannot be used to model data from changing information trials. Thus, LANs will likely be less efficient for exploratory studies.

This is only a sub-sample of methods (see Table 1) and there are a number of variations on them. However, they illustrate that there are currently no methods for performing rapid Bayesian choice-RT modeling with general EAMs. Each makes some sacrifice that limits their generality, accuracy, or efficiency. To address this gap in methods, we introduce the Python package PyBEAM (Bayesian Evidence Accumulation Models). The approach we describe here will provide the first general, accurate, and efficient method for (non-hierarchical) Bayesian choice-RT modeling of binary choice EAMs. For discussion of the limitations of this approach, see the "Limitations of approach" Section prior to the discussion.

Methods

In this section, we describe the broad methodological details of our algorithm for performing Bayesian parameter estimation for two-threshold models of binary evidence accumulation. Before continuing, we make a few notes. First, we mainly discuss the general idea behind this method. There are numerous implementation details "under the hood" that are required to make this work; however, we leave this discussion for the Supplementary Information. Second, we do not describe the Pythonic implementation of this algorithm in this article. Instead, we will provide multiple Jupyter notebooks (Kluyver et al., 2016) demonstrating the use of this method for both parameter recovery and application to data. These notebooks are more than just documented code. They are descriptive in nature and should be more effective than writing pseudo-code. These are publicly available and can be found at the PyBEAM GitHub (https://github.com/murrowma/pybeam). Finally, while reading these methods will likely help a reader understand how this method works at a high level, understanding these methods are not required to use the provided Python implementation.

The general two-threshold binary accumulation model

In this article, we develop an approach for calculating choice-RT likelihoods for general two threshold models of binary decision making and integrate them with Bayesian parameter estimation. For this type of model (Fig. 1), the evidence accumulation process begins at point z, corresponding to an initial bias prior to stimulus presentation. Upon stimulus presentation, evidence is noisily accumulated, described by the stochastic differential equation (SDE) (Buonocore, Giorno, Nobile, & Ricciardi, 1990; Smith, 2000),

$$dx(t) = v(x,t)dt + D(x,t)dB(t), \tag{1}$$

where $x(t)$ is the total evidence accumulated at time t and $v(x,t)$ is the rate of evidence accumulation (referred to as the drift rate). The drift is determined by the quality of information in the presented stimulus, where decisions with clear stimulus information produce larger drift rates. The function $D(x,t)$ is the diffusion rate, and though in some models it can depend on (x,t), it is most often constant and fixed for scaling purposes (commonly set to either $D(x,t) = 1, 0.1$). Lastly, $B(t)$ is a Gaussian noise term such that $dB(t)$ is the Gaussian error.

The evidence accumulation process described by Eq. 1 continues until one of the two opposing decision thresholds ($c_1(t)$ or $c_2(t)$) is reached, triggering a response (Fig. 1). The separation $s(t)$ between thresholds indicates the degree of time dependent caution exhibited by the decision maker. If

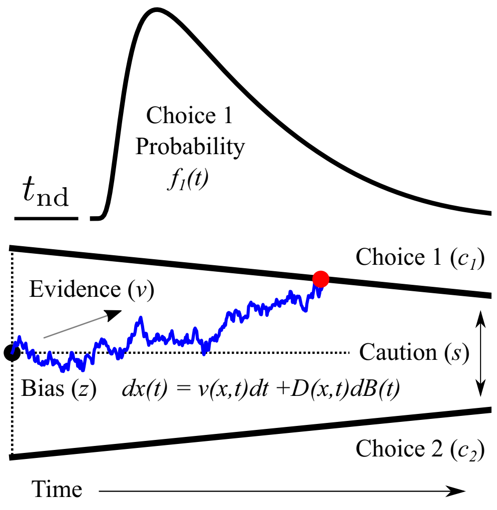

Fig. 1 **Model Schematic**. Evidence is accumulated (blue line) starting from an initial bias (black dot) until one of the two choice thresholds, c_1 or c_2, is reached (red dot), triggering a decision. The distance between thresholds indicates an individuals degree of caution which can change or remain constant as a function of time. Mathematically, this process is described by the noted stochastic differential equation, where $x(t)$ is the total accumulated evidence (or preference), $v(x,t)$ is the rate of evidence accumulation, $D(x,t)$ is the diffusion rate, and $B(t)$ is the noise term. This process results in the choice probability distribution $f_i(x,t)$, which indicates how likely it is for an accumulator to cross the decision threshold at that time. t_{nd} indicates the time it takes for non-decision processes, which is added to the decision time from the response time

thresholds are far apart (close together), the participant makes slower (faster), more (less) cautious decisions. Though in many evidence accumulation models the degree of caution is fixed, time changing caution and accordingly thresholds (as shown in Fig. 1) is increasingly being investigated as a mechanism to optimize reward rates (Drugowitsch, Moreno-Bote, Churchland, Shadlen, & Pouget, 2012; Tajima, Drugowitsch, & Pouget, 2016) or the speed accuracy trade-off (Frazier & Yu, 2007). An additional parameter referred to as the non-decision time, t_{nd}, is also generally included in EAMs. This term describes the time it takes for a participant to encode the stimulus information and the motor processes involved in selecting one of the available choices.

Stochastic to probabilistic model form: The Fokker-Planck formalism

So called choice-RT data is common in this field. In order to determine how well a particular set of parameters for a particular model describe data of this form, it is useful to calculate the "likelihood" function which describes the probability of making the observed choice at the observed time. Though the stochastic form of binary choice evidence accumulation models is simple to formulate and interpret, it does not immediately provide us with this quantity of interest.

This can be circumvented in a number of ways. Buonocore, Giorno, Nobile, and Ricciardi, 1990 devised an analytic integral based solution for this problem, which is summarized for a psychological audience in Smith, 2000; however, it is computationally intensive. Further, since it requires numerical calculation of integrals, it only can approximate the actual solution. Stochastic simulation based methods (Chandrasekaran & Hawkins, 2019; Holmes, 2015) are more general, but are computationally inefficient largely due to the necessity of generating immense quantities of random numbers.

Fortunately, this stochastic process can instead be written as a Fokker-Planck equation, which describes the probability that the accumulator has precisely state (x) at time (t) (Öttinger, 1996). We in particular use the forwards Fokker-Planck (FP) equation,

$$\frac{\partial p(x,t)}{\partial t} = -\frac{\partial\left[v(x,t)p(x,t)\right]}{\partial x} + \frac{1}{2}\frac{\partial^2\left[D(x,t)^2 p(x,t)\right]}{\partial x^2}, \tag{2}$$

where $p(x,t)$ is the probability of accumulated evidence x at time t, and $v(x,t)$ and $D(x,t)$ are the drift and diffusion rates introduced earlier. The EAM response thresholds are modeled as the PDE's boundary conditions. Specifically, we use absorbing boundary conditions where $p(c_1(t),t) = p(c_2(t),t) = 0$, encoding the fact that when the preference state reaches the threshold, it is removed or absorbed by that threshold.

While this is a probability, it is still not quite the quantity we need. This describes the accumulator state rather than the probability of a choice. We require the probability of having first crossed either of the two thresholds at time t. This is often referred to as the first passage time problem. Fortunately, this first passage time probability can be directly calculated from $p(x,t)$ by calculating the flux of probability through the threshold of interest. Here,

$$J(x,t) = v(x,t)p(x,t) - \frac{1}{2}\frac{\partial\left[D(x,t)^2 p(x,t)\right]}{\partial x}, \tag{3}$$

is the probability flux at point (x,t) and the first passage time density at threshold i can be calculated as $f_i(t) = J(c_i(t),t)$.

As a synopsis, here are the steps necessary to calculate the likelihood function. 1) Transform the model from a SDE formalism to the FP formalism. 2) Solve the the FP equation in the relevant (x,t) domain determined by the model thresholds. 3) Use that solution to calculate probability fluxes at the threshold, which ultimately is the likelihood function. Step 1 is straightforward as any SDE of the form discussed here can be directly transformed into a FP. Step 3 is also straightforward once $p(x,t)$ is calculated. Step 2, simulating the FP equation, is the most complex and requires the most care.

A note on the forward versus backward Fokker-Planck approach

This is a technical note that we make for the interested reader. An alternative to using the forwards FP equation is to use the backwards FP equation. The backwards FP encodes the same diffusion process, but is conceptually very different. Whereas the forwards FP provides the probability of accumulator state x and time t given an initial accumulator state, the backwards equation - in the context of a first passage time problem - indicates the probability of an accumulator in state x at time t crossing the threshold at some future time T, and is integrated backwards from T to t. This form has previously been applied to cognitive models, most notably in the package fast-dm (Voss & Voss, 2007, 2008). For our application, it has a few significant drawbacks which prevented us from using it. First, the backwards FP requires two PDEs, one for the upper threshold crossing and another for the lower threshold crossing. These must be solved independently, incurring a two times speed cost to the solution. Secondly, the backwards FP does not have a computationally inexpensive way to implement time changing thresholds or non-constant drift rates. Unlike the forwards FP, where a simple change in coordinates across x eliminates many of the difficulties associated with changing thresholds (discussed in more detail in the "Numerical solution" Section), the backwards FP requires more care. As noted in Voss and Voss, 2008 and discussed in Boehm, Cox, Gantner, and Stevenson, 2021, if the decision thresholds and drift rate do not depend on time, solving the backwards FP provides the likelihood function for all times t. However, if the thresholds or drift rate are time dependent, then a different FP equation is required for each time t. This means that the backwards FP would need to be solved independently for each choice-RT point in order to fit data to experiment.

The backwards FP does however have a couple advantages worth noting. First, unlike the forwards FP, which requires the calculation of the probability flux at the threshold to determine the likelihood function, the solution to the backwards FP gives the likelihood function immediately (Voss & Voss, 2008), potentially reducing numerical error. Second, (Boehm, Cox, Gantner, & Stevenson, 2021, 2022) devised an integral transformation which improves the regularity and numerical properties of the backwards FP equation. This integral transformation, however, is unique for each threshold, meaning that each threshold requires a new integral to be computed. Additionally, this transformation cannot be applied if the diffusion rate is non-constant, like in the Urgency Gating Model. Since the purpose of PyBEAM is to be highly general and allow for users to easily create custom models, it is more convenient to use the forwards FP.

Numerical solution

Here we describe the main ideas of the process for numerically simulating the accumulator state probability $p(x, t)$ from the FP equation. We have added numerous bells and whistles to this scheme for numerical stability, generality, and numerical speed. These are further discussed in the Supplementary Information and we focus on the main ideas only here.

There are three sources of complexity in this model: the state dependent drift $v(x, t)$, the state dependent diffusion $D(x, t)$, and the time dependent bounds $c_i(t)$. It turns out the bounds are the most challenging to deal with. We will be using finite differences to solve this problem. This causes issues like thresholds crossing between discretized grid points and points being inside the threshold at one time and outside at the next, introducing significant error into the solution. Though these can be overcome, there is a better way to address this complexity.

In its current form, the FP model is described by a relatively simple PDE with a time changing bound. Since this time changing bound is complicated to deal with numerically, we make a change of coordinates that flattens the thresholds (Crank, 1984) at the expense of making the PDE more complicated. Note that since this is an exact transformation, no information about the model is lost. We introduce the following change of coordinates,

$$\varepsilon = \frac{x - c_1(t)}{c_1(t) - c_2(t)}. \tag{4}$$

In this new coordinate ε, the thresholds are fixed at values zero and one. Substituting into Eq. 2 yields,

$$\frac{\partial p(\varepsilon, t)}{\partial t} = \frac{1}{s}\left[\varepsilon \frac{ds}{dt} + \frac{dc_2(t)}{dt}\right] \frac{\partial p(\varepsilon, t)}{\partial \varepsilon} \tag{5}$$
$$- \frac{1}{s} \frac{\partial \left[v(\varepsilon, t) p(\varepsilon, t)\right]}{\partial \varepsilon} + \frac{1}{2s^2} \frac{\partial^2 \left[D(\varepsilon, t)^2 p(\varepsilon, t)\right]}{\partial \varepsilon^2},$$

where $p(\varepsilon, t)$ is the probability of accumulated evidence in the new coordinate frame ε, $s = c_1(t) - c_2(t)$ is the separation between thresholds, and $v(\varepsilon, t)$ and $D(\varepsilon, t)$ are the drift and diffusion rates in the new coordinate frame. Though Eq. 5 is more complex, the decision thresholds are now constant in time with the upper threshold at 1 and the lower at 0. This transforms the space domain to $(0, 1)$, dramatically simplifying implementation since the computational domain is now a rectangle.

This transformed problem now amounts to solving a PDE on a rectangular domain. To solve this, we use the second order Crank-Nicolson finite difference scheme (Crank & Nicolson, 1947). This is a highly robust, unconditionally stable numerical scheme used in a wide range of PDE simulation applications. We do not go into detail here, but we have extensively tested the numerical discretization used for this method and introduced adaptive stepping in coordinate t to improve it (described in detail in the Supplementary Information). While we do not focus on these implementation complexities here, we note that they are absolutely critical to the practical application of this method. First, they speed the numerical solution by a factor of 10x or more, which is important when integrating this into a Bayesian framework. Second, these optimizations allow the user to apply this method to a wide array of problems without having to substantially "tune" the algorithm. With this method, we can now robustly and efficiently calculate $p(x, t)$, which can then be used to compute first passage time probabilities and construct the desired likelihood function.

We do make a brief note about one computational complexity. As noted by Shinn, Lam, and Murray, 2020, the Crank Nicholson scheme can introduce a numerical instability arising from the initial condition to the FP equation (i.e. the start point distribution), which can be particularly problematic with time varying thresholds. This is a well known issue (Østerby, 2003). We ameliorate this by starting the PDE simulation with multiple small time steps to essentially smooth the transition from the initial condition to the remaining time domain. This adjustment removes this issue without any intervention by the user, and is discussed in more detail in the Supplementary Information. With this and other similar optimizations, we have a highly efficient and stable algorithm for calculating choice-RT likelihoods.

Across-trial variability

PyBEAM's numerical solution readily allows for the addition of across-trial variability in the start point, non-decision time, and drift rate. To implement across-trial variability in the start point, we alter the FP PDE's initial condition. If no across-trial variability is desired, PyBEAM uses a Dirac delta function as the initial condition, localizing all probability mass at a single point. If across-trial variability is desired, PyBEAM implements one of two functional forms which can be chosen by the user: uniform and truncated normal. We truncate the normal distribution near the decision thresholds so that no values fall outside the boundary conditions. In the future, more flexibility will be provided to the user to implement their own start distributions. Starting point distributions are implemented at little to no computational cost. This trial-to-trial variability can be added to any model in PyBEAM.

To implement across-trial variability in the non-decision time, we follow the procedure described by Ratcliff and Tuerlinckx in Appendix B of their publication (Ratcliff & Tuerlinckx, 2002). First, we generate the likelihood function by solving the FP equation as described in Section

"Numerical solution." Then, we take the likelihood function and integrate across the non-decision time distribution. Since our likelihood function is calculated using finite difference and thus discrete, we numerically integrate the two functions together using the trapezoidal rule. Two non-decision time distributions are pre-coded in PyBEAM: uniform and truncated normal. We choose a truncated normal distribution instead of a normal distribution to guarantee that no non-decision time values end up below zero. If a uniform distribution is chosen, we integrate across the entire distribution. If a truncated normal distribution is chosen, we set the integration bounds at four standard deviations away from the mean. The result of this integration is the likelihood function with across-trial variability in the non-decision time included. This integration step is highly efficient in PyBEAM, incurring only minimal speed losses. This trial-to-trial variability can be added to any model in PyBEAM.

Trial-to-trial variability in "drift rate" parameters is more complex since in some cases a models drift rate is a parameter and in others it is a function. For this reason, we do not add this feature generally. However, since it is important to the dynamics of the "full DDM," we include it for this model for completeness. To incorporate this, we again follow the procedure described by Ratcliff and Tuerlinckx in Appendix B of Ratcliff and Tuerlinckx, 2002 to integrate over the drift rate distribution. Since our likelihood function is calculated using finite difference and thus discrete, we numerically integrate the two functions together using the trapezoidal rule. PyBEAM by default implements two drift distributions: uniform and normal. We select N_μ equally spaced drift rates from these distributions. By default, PyBEAM sets this value to $N_\mu = 10$, but this can be altered by the user if more or less precision is required. If the drift rate is uniformly distributed, we select N_μ drift rates such that both bounds of the distribution are included. If normally distributed, we choose N_μ drift rates with bounds four standard deviations away from the mean drift rate. For each of these points from the numerical discretization of the drift rate distribution, a likelihood function is generated. These are then integrated over the discretized drift rate distribution using standard numerical integration. The result of this integration is the likelihood function with across-trial variability in the drift rate included. Since we by default generate $N_\mu = 10$ likelihood functions, this incurs an ~10x slowdown compared to the base models without this complexity. This across-trial variability is only included for the DDM.

Parameter inference

To perform Bayesian parameter estimation, we integrate this method of likelihood construction into the Python package PyMC (Salvatier, Wiecki, & Fonnesbeck, 2016). PyMC is a highly robust, well supported Python package designed specifically to perform Markov chain Monte Carlo. Using this Bayseian platform comes with a number of benefits. First, it has been tested by a wide array of researchers over a number of years. Second, it has a number of different MCMC samplers integrated into it, which is important for this application. Third, it simplifies posterior analysis since PyMC has a number of functionalities built in (e.g. trace plotting and posterior summary statistics) and supporting packages such as ArviZ (Kumar, Carroll, Hartikainen, & Martin, 2019) are available.

Though PyMC is known principally for implementing gradient based Monte Carlo methods like NUTS (Hoffman & Gelman, 2014), we find that this algorithm is sub-optimal for this type of problem. Since these methods require the calculation of model derivatives with respect to every parameter in each step of the inference process - something that is very slow when performed numerically - they work best when fast, analytic solutions are available. We instead utilize an MCMC sampler which does not require gradients, history based Differential Evolution Markov Chain (DE-MCz) (Braak & Vrugt, 2008), a variant of the popular Differential Evolution Markov Chain (DE-MC) algorithm (Braak, 2006). In brief, DE-MC runs many chains in parallel and uses information shared between chains to auto-sense the structure of the posterior and intelligently construct parameter proposals. The history based version, DE-MCz, works similarly except that each chain only looks at its own history, not other chains. Based on PyMC's analysis of this algorithm, DE-MCz is able to achieve comparable results to DE-MC with many fewer chains (DEMetropolis(Z): Population vs. History efficiency comparison, 2022), making it an attractive algorithm for use on a personal computer with a limited number of cores.

As demonstrated in the following results, coupling this approach to constructing model likelihoods with the capabilities of PyMC yields a robust and efficient tool for performing Bayesian inference with general binary choice-RT models.

EAMs implemented in PyBEAM

To facilitate use of this approach while allowing maximum flexibility, we have implemented two sections of the Python package. First, we have constructed pre-coded versions of a number of common models. This allows for rapid use with minimal interaction using the PyBEAM framework. Second, we implement a general model class that allows users to go beyond the common models. This requires more interaction with the modeling framework, but provides increased flexibility. We address both approaches in the remainder of this section.

Pre-coded EAMs

PyBEAM contains pre-coded versions of EAMs commonly used in the literature. The basis for all pre-coded models is the EAM described in Eq. 1, which is then modified to match the specific model needs.

The simplest of these is an EAM with constant non-decision time t_{nd}; constant drift rate $v(x,t) = \mu$; constant diffusion $D(x,t) = \sigma$, usually set to either $\sigma = 0.1, 1.0$; accumulation start point z; and flat, symmetric decision thresholds $c_1(t) = -c_2(t) = b$. For convenience, we replace z with $w = z/(2b)$, referred to as the relative start point, which sets the accumulator start as a ratio of the threshold separation (ranging from 0 to 1). This EAM is sometimes referred to in the literature as the "simple DDM" since it is a DDM without across-trial variability parameters (Evans, Hawkins, & Brown, 2020). We also refer to it in PyBEAM as the simple DDM.

In addition to the simple DDM, PyBEAM also includes a pre-coded version of the full DDM. The full DDM is the simple DDM with across-trial variability in the non-decision time, start point, and drift rate included. As discussed in Section "Methods," the non-decision time and start point have two pre-coded distributions: uniform and truncated normal. Across-trial variability in the drift rate can be either uniform or normally distributed.

The next two pre-coded EAMs add additional model assumptions to the simple DDM described above to account for other psychological factors. The first, referred to as the "leakage" model in PyBEAM, adds leaky integration to the simple DDM. This changes the drift rate from a constant to $v(x,t) = \mu - lx$, where l is the strength of leaky integration and x is the total accumulated evidence. The second, referred to as "changing thresholds," adds time varying decision thresholds to the simple DDM. They are free to either collapse towards zero or move outwards, though collapse is the more common behavior in the literature. By default, three time varying decisions thresholds are implemented in PyBEAM. The first, linear, defines the decision thresholds as,

$$c_1(t) = -c_2(t) = b - mt, \tag{6}$$

where b indicates the threshold location at time zero and m is the thresholds' slope. The second, exponential, has thresholds defined as,

$$c_1(t) = -c_2(t) = b\exp(-t/\tau), \tag{7}$$

where b, as before, is the threshold location at time zero and τ describes how quickly the threshold collapses. The last, Weibull, uses a Weibull distribution function for the decision threshold, given by,

$$c_1(t) = -c_2(t) = b - \frac{b(1-c)}{2}\left[1 - \exp\left(-\left(\frac{t}{\lambda}\right)^{\kappa}\right)\right]. \tag{8}$$

Here, b has the same meaning as in the linear and exponential models. The next parameter, λ, is the scale parameter and approximately sets the time at which threshold collapse or expansion occurs. Parameter κ is referred to as the shape parameter and indicates whether the threshold behavior is logistic ($\kappa > 1$) or exponential ($\kappa < 1$). Lastly, c is the collapse ratio and sets the amount the threshold collapses or expands. If $c = -1$, then the thresholds collapse towards zero. If $c > 1$, thresholds expand away from their initial location b. The Weibull distribution is a commonly used threshold due to its flexibility in behavior (Hawkins, Forstmann, Wagenmakers, Ratcliff, & Brown, 2015). Depending upon the choice of parameters, it is able to model early collapse, late collapse, and no collapse, while simultaneously replicating logistic and exponential threshold behavior.

Though the Weibull decision threshold still maintains the same functional form, we modify it slightly when running PyBEAM's MCMC sampler. Instead of sampling the shape λ and scale κ parameters directly, we instead sample from the base ten logarithm of these parameters. Since the Weibull threshold can produce both logistic and exponential behavior, λ and κ have large functional parameter ranges. This makes it difficult for the MCMC algorithm to evenly sample from all parts of parameter space. For example, common κ values for an exponential model are between 0.5 and 1, while κ values for a logistic model generally range from 1 to 10. This disparity in reasonable parameter range makes its difficult to sample from accurately. Thus, by default, PyBEAM samples from and reports the base ten logarithm for parameters λ and κ.

The final EAM currently pre-coded in PyBEAM is the Urgency Gating Model, referred to as the "UGM" (Cisek, Puskas, & El-Murr, 2009). As in Trueblood, Heathcote, Evans, and Holmes, 2021, it starts with the simple DDM discussed above, then adds an urgency signal and leakage to the drift rate and an urgency signal to the diffusion rate. The drift rate is defined as,

$$v(x,t) = \mu(1+kt) + \left(\frac{k}{1+kt} - l\right)x. \tag{9}$$

Here, μ and l are the drift and leakage rates from the simple DDM and leakage EAMs described above. The additional parameter, k, is referred to as the urgency ratio parameter and describes the strength of urgency in the model. If $k = 0$, the model encodes no urgency and is instead the leakage EAM. If $k \to \infty$, urgency dominates the decision process. In addition to altering the drift rate, the UGM modifies the diffusion rate, $D(x,t)$. It is given by,

$$D(x,t) = \sigma(1+kt), \tag{10}$$

where k is still the urgency ratio and σ is the noise parameter from the simple DDM.

Note that the option to include across-trial variability in the non-decision time and start point is available for all pre-coded models. For example, if you would like a UGM to have a normally distributed non-decision time and uniform start distribution, PyBEAM allows this with a simple change to the model input. We do not allow for across-trial variability in the drift rate outside of the full DDM in the present implementation.

Custom models in PyBEAM

In addition to providing users with pre-coded models, PyBEAM also allows for the creation of user-defined custom scripts. Instructions for creating a script can be found on the PyBEAM GitHub (https://github.com/murrowma/pybeam), but we provide a useful example here which we test in the results section of this paper.

Though we provide the UGM by default in PyBEAM (see Section "Pre-coded EAMs"), it is known that parameter recovery for it can be challenging. The urgency parameter k is correlated with a and l, causing multiple combinations of each parameter to generate similar likelihood functions. To address this problem and improve parameter recovery, Trueblood, Heathcote, Evans, and Holmes, 2021 proposed an experiment with time changing stimulus information. Specifically, using a grid of pixels flashing one of two colors (blue / orange), they altered the fraction of each color on the screen at a given time while a decision was being made. So, early in the decision process, the grid may be 55 percent blue while later it might be 55 percent orange.

We implement a similar model as a custom script in PyBEAM. In the UGM, stimulus strength is encoded in the parameter μ. Thus, we modify the UGM drift rate from Eq. 9 to account for the time changing stimulus information, giving,

$$v(x,t) = \begin{cases} \mu(1+kt) + \left(\frac{k}{1+kt} - l\right)x, & \text{if } t < t_{\text{flip}}, \\ -\mu(1+kt) + \left(\frac{k}{1+kt} - l\right)x, & \text{if } t \geq t_{\text{flip}}, \end{cases} \tag{11}$$

where t_{flip} is the time when stimulus information changes. For example, at time $t < t_{\text{flip}}$, the grid may be dominated by blue which corresponds to positive μ. Then, at time t_{flip}, they flip to orange dominated, modeled by making μ negative.

Results

In this section, we test PyBEAM's approach for fitting EAMs to data. First, we validate our method of constructing the likelihood function. Second, we validate parameter recovery. Lastly, we use PyBEAM to fit EAMs to experimental data and compare our results to prior published work.

Validating likelihood construction

We first validate PyBEAM's likelihood construction. We do this for three of the models discussed in Section "Pre-coded EAMs": the simple DDM, the leakage EAM, and the Weibull changing thresholds model. For each model, we generate 100 random parameter sets similar to those seen in experiment, then simulate 250 data points for each set using Eq. 1. Using the data generated from each model, we calculated the total log-likelihood of the data sets using both the analytic solution discussed in the introduction and the PyBEAM numerical approach. The analytic solution was pulled from Drugowitsch's lab github (https://github.com/DrugowitschLab/dm). The results of this are shown in Fig. 2.

In Panel A, we display the analytic log-likelihood ($\text{LL}_{\text{analytic}}$) versus the PyBEAM numerical log-likelihood ($\text{LL}_{\text{PyBEAM, Def}}$) for all parameter sets in red dots. The solid black line indicates the graphical location where the analytic and PyBEAM solutions are identical. PyBEAM has several spatial resolution settings, and we calculate the log-likelihood for this panel using the default spatial resolution. PyBEAM also allows the user to increase the time resolution; however, it has an insignificant effect on the final result. In Panel B, we report the error in the log-likelihood as boxplots for each model type and level of computational resolution. The error is calculated by taking the difference between the numerical (LL_{P}) and analytic (LL_{a}) solutions. We do this for three different PyBEAM resolution settings: Def (default), High, and Max. These results show that PyBEAM's numerical approach to calculating likelihood functions is highly accurate in all cases. The biggest errors occur when either the initial threshold separation is large ($b \geq 1.5$), or when the drift rate is large ($\mu \geq 4$). In our experience, these deviations from the analytic solution have only a small effect on parameter estimation, but if it is a concern, increasing PyBEAM's spatial resolution mitigates the problem. Notably, the PyBEAM numerical solution is 5-10x faster than the analytic solution with only minimal losses in precision.

Parameter recovery

We next assess PyBEAM's ability to perform model and parameter recovery for EAMs of the type discussed in Section "EAMs implemented in PyBEAM ." To do this, we first simulate data from the model. Then, we use PyBEAM to fit the model to the simulated data set. Lastly, we compare the posterior distributions output by PyBEAM to the generating parameters and the best fit choice-RT distribution to the true distribution.

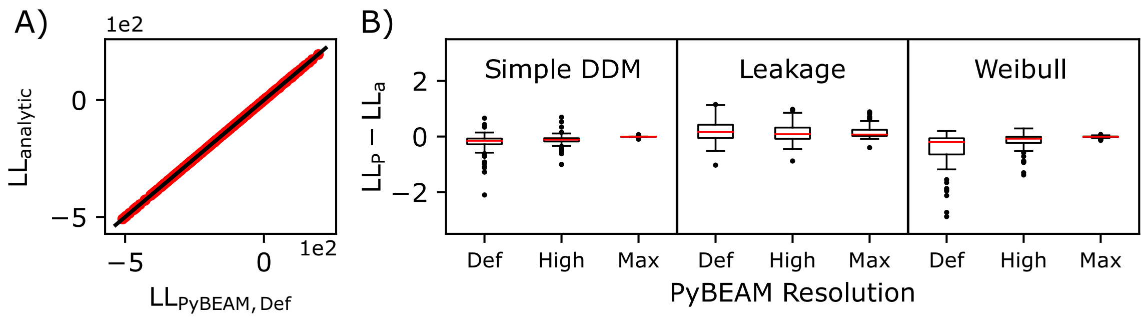

Fig. 2 Validation of PyBEAM likelihood construction. **A)** Analytic log-likelihood (LL_{analytic}) versus PyBEAM's numerical log-likelihood ($LL_{\text{PyBEAM,def}}$) of 250 simulated data points (red dots). Calculated at PyBEAM's default spatial resolution setting. 300 total parameter sets were chosen, 100 each for simple DDM, the leakage EAM, and the Weibull changing thresholds EAM. The black line indicates the graphical location where the analytic and numerical solutions are equal. **B)** Boxplot of the error in log-likelihood calculation for each parameter set. Calculated as the difference between the analytic log-likelihood (LL_a) and PyBEAM numerical log-likelihood (LL_P) for each model. Def (short for default), High, and Max indicate PyBEAM's spatial resolution settings, with higher resolutions resulting in smaller errors

We make a brief note that there are multiple ways to assess the efficacy of parameter recovery. The first and most straightforward is to determine whether the posterior distribution found matches the generating parameters. However, models, particularly more complex ones, often present parameter indeterminacy issues where a subspace of parameters can produce nearly identical model outputs. In this case, the estimation procedure may fail to recover the exact generating parameters but still fit the data well. This is a consequence of the model structure and is not a failing of the parameter estimation procedure. For this reason we visualize both posterior parameter distributions as well as the quality of fit to the data itself.

Simple DDM

We first demonstrate parameter recovery for the simple DDM described in Section "Pre-coded EAMs." Following the procedure outlined in Section "Parameter recovery," we first simulate 1000 data points from the model using Eq. 1. Then, we use PyBEAM to fit the model to this synthetic data. Results for this process are shown in Fig. 3.

Panel A of Fig. 3 compares the analytic (dashed blue) and PyBEAM (red) likelihood functions using the known generating parameters. The results exactly overlap, demonstrating once again that the PyBEAM approach produces highly accurate choice-RT distributions. Panel B displays the simulated data and the posterior predictive fit given by PyBEAM. In this example and all going forwards, we generate the posterior predictive by drawing 100 random parameter sets from the PyBEAM posteriors and plotting the likelihood function generated by each parameter set (red). The posterior predictive displays the range of likelihood distributions which fit the data, providing an approximation of the variance in fit. In this case, the posterior predictive describes the data set well, with little variation between the predicted likelihood function and data.

The remaining panels C1-C4 display the model parameter posteriors generated by PyBEAM. The vertical blue lines indicate the true parameter values of the simulated data sets, while the grey histograms are the posteriors. These results show parameter recovery is excellent for the simple DDM. In addition, PyBEAM is able to recover parameters quickly, with an approximate run time for this experiment of less than one minute on a 2020 Macbook Pro.

Full DDM

The second model we recover parameters from is the full DDM discussed in Section "Pre-coded EAMs." This is a simple DDM with across-trial variability included in the drift rate, start point, and non-decision time. For this validation, we choose to use a normal distribution for the drift rate, a uniform distribution for the start point, and a truncated normal distribution for the non-decision time. We truncate the normal distribution in the non-decision time so that no values go below zero. We use the same procedure as the simple DDM validation, simulating 1000 data points which we then fit with PyBEAM. Results for this validation are shown in Fig. 4.

Panel A of Fig. 4 compares the likelihood function generated by PyBEAM (red) to 100,000 simulated data points (blue histogram) using the known generating parameters. PyBEAM closely matches the simulated data, demonstrating that PyBEAM accurately reproduces the choice-RT distribution. Panel B displays the simulated data (grey) and PyBEAM posterior predictive (red), demonstrating that PyBEAM's parameter estimation closely fits the simulated data.

The remaining Panels C1-C7 display the model parameter posteriors generated by PyBEAM. Vertical blue lines indi-

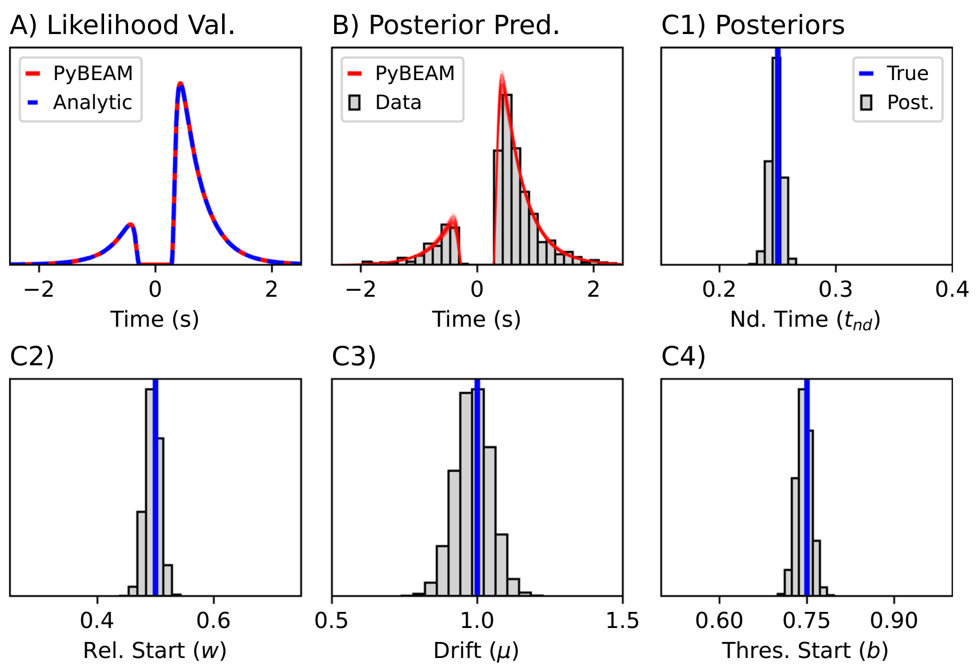

Fig. 3 Results from the simple DDM parameter recovery example. **A)** Likelihood function validation for the simple DDM. The red line is the PyBEAM numerical solution, while the blue dashed line is the analytic solution (Navarro & Fuss, 2009). **B)** PyBEAM posterior predictive. Red lines are the PyBEAM solutions, while the grey bars are the 1000 simulated data points. **C)** PyBEAM posteriors for each of the simple DDM's parameters. Grey bars are the PyBEAM posteriors, while the blue lines are the true values used to generate the data. C1 displays the non-decision time posterior, with a true value of $t_{nd} = 0.25$. C2 displays the relative start point posterior, with a true value of $w = 0.5$. C3 displays the drift rate posterior, with a true value of $\mu = 1.0$. C4 displays the decision threshold posterior, with a true value of $b = 0.75$

cate the true parameter values of the simulated data sets, while the grey histograms are the PyBEAM posteriors. Recovery of non-decision time, start point, and drift parameters is good. Recovery of across-trial variability parameters, which are known to be difficult to recover (Boehm et al., 2018), is however more complex and beyond the scope of this article.

The run time needed to recover parameters is approximately 7 minutes on a 2020 Macbook Pro. This is about a 10x slowdown when compared to the simple DDM recovery. This increase in run time is due to inclusion of across-trial variability in the drift rate. As discussed in Section "Across-trial variability," including across-trial variability in the drift rate involves an integration over many likelihood functions which covers the distribution of drift rates. We find that $N_\mu = 10$ integration points is sufficient for parameter recovery, meaning that we need to solve for 10 likelihood functions, leading to our 10x slowdown. Across-trial variability in the start point contributes nothing to the run time, while the non-decision variability causes only a tiny increase in run time.

Leaky integration

The third model we recover parameters from is the leakage model discussed in Section "Pre-coded models." Recovering the leakage parameter from a single condition EAM can be challenging, so we instead simulate a slightly more complicated data set with two conditions: low decision threshold b_1 (condition 1) and high decision threshold b_2 (condition 2). This mimics a speed/accuracy trade-off scenario. The remaining parameters are shared between the two conditions and only the threshold parameter b differs between the two simulated conditions.

To validate PyBEAM's parameter recovery for the leakage EAM, we follow the process outlined at the beginning of the results section. We simulate 500 data points for each condition (1000 total data points). Then, we use PyBEAM to find the model parameters which best describe this data. We report the results of parameter recovery in Fig. 5.

In panel groups A and B of Fig. 5, we report the likelihood validation and posterior predictive plots for condition 1 (low b) and condition 2 (high b), respectively. As with the previous example, likelihood validation compares the PyBEAM likelihood to the analytic (Buonocore, Giorno, Nobile, & Ricciardi, 1990; Smith, 2000) for the known generating parameters while the posterior predictive panels compare the data to the RT distributions resulting from the PyBEAM model fits. Results demonstrate that once again, PyBEAM produces highly accurate likelihood functions and fits the generating data well.

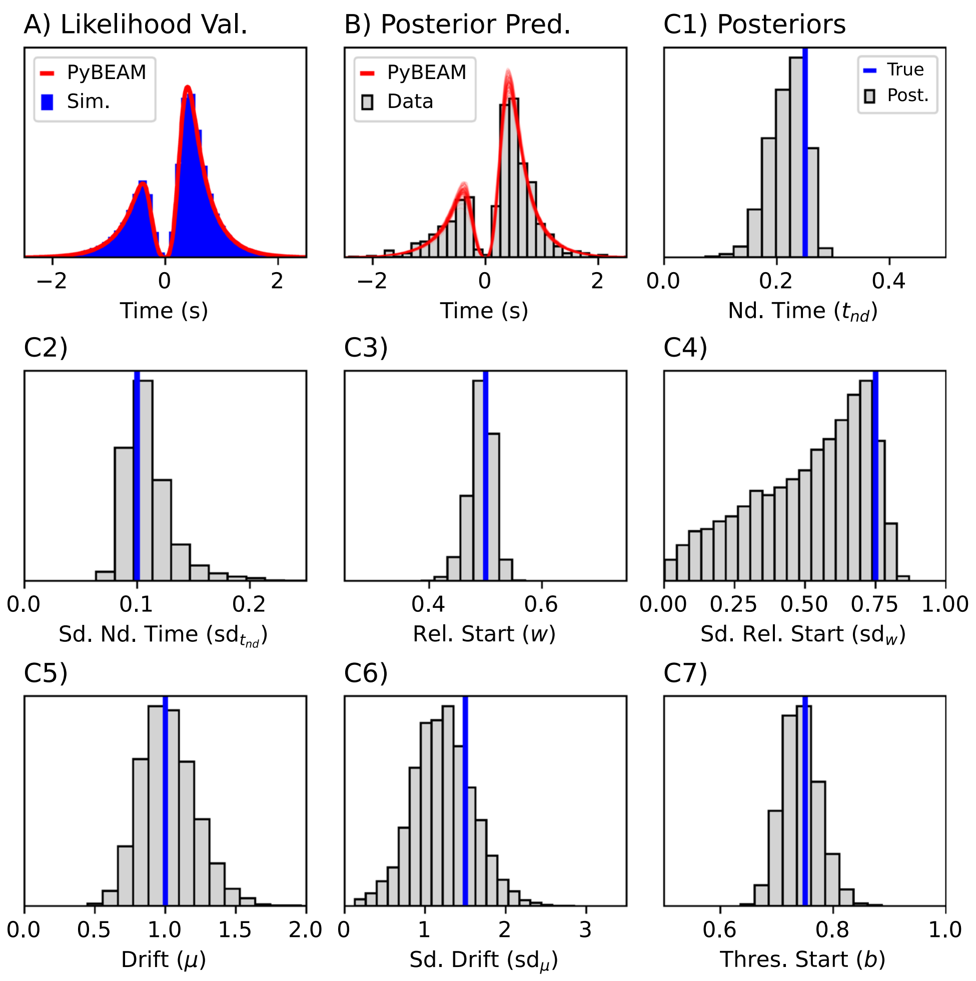

Fig. 4 Results from the DDM parameter recovery example. **A)** Likelihood function validation for the DDM. The red line is the PyBEAM numerical solution, while the blue histogram contains 100,000 simulated data points. **B)** PyBEAM posterior predictive. Red lines are the PyBEAM solutions, while the grey bars are the 1000 simulated data points. **C)** PyBEAM posteriors for each of the DDM's parameters. Grey bars are the PyBEAM posteriors, while the blue lines are the true values used to generate the data. C1 displays the posterior for the mean of the non-decision time distribution, with true value $t_{nd} = 0.25$. C2 displays posterior for the standard deviation in the non-decision time, with true value $sd_{t_{nd}} = 0.1$. C3 displays the posterior for the relative start distribution's center, with true value $w = 0.5$. C4 displays the posterior for the relative start distribution's width, with true value $sd_w = 0.75$. C5 displays the posterior for the drift rate distribution's mean, with true value $\mu = 1.0$. C6 displays the posterior for the standard deviation in the drift rate, with true value $sd_\mu = 1.5$. C7 displays the decision threshold posterior, with true value $b = 0.75$

Panel group C displays the posteriors for both fits. We find that the shared parameters - non-decision time, relative start, drift, and leakage - have posteriors (grey) which closely match the true values (blue lines). The individual caution parameters b_1 and b_2 also match expectation closely. Parameter recovery is once again highly efficient, taking approximately a minute on a 2020 MacBook Pro.

Changing decision thresholds

We next use PyBEAM to recover parameters from the changing thresholds model. Specifically, we use the Weibull decision thresholds described in Section "Pre-coded models." To validate PyBEAM's parameter recovery for this model, we again follow the methodology described at the beginning of the results. First, we choose our parameters. Since the Weibull threshold is capable of producing both logistic and exponential thresholds, we choose two parameter sets to replicate these behaviors: one with $\kappa > 1$ and one with $\kappa < 1$. The Weibull threshold is also able to produce varying amounts of collapse via the c parameter. For convenience, for both parameter sets, we fix this at $c = -1$, indicating that the thresholds will collapse to zero. We simulate 1000 data points for each parameter set. We then use PyBEAM to recover parameters from the simulated data. The parameter recovery results are shown in Fig. 6.

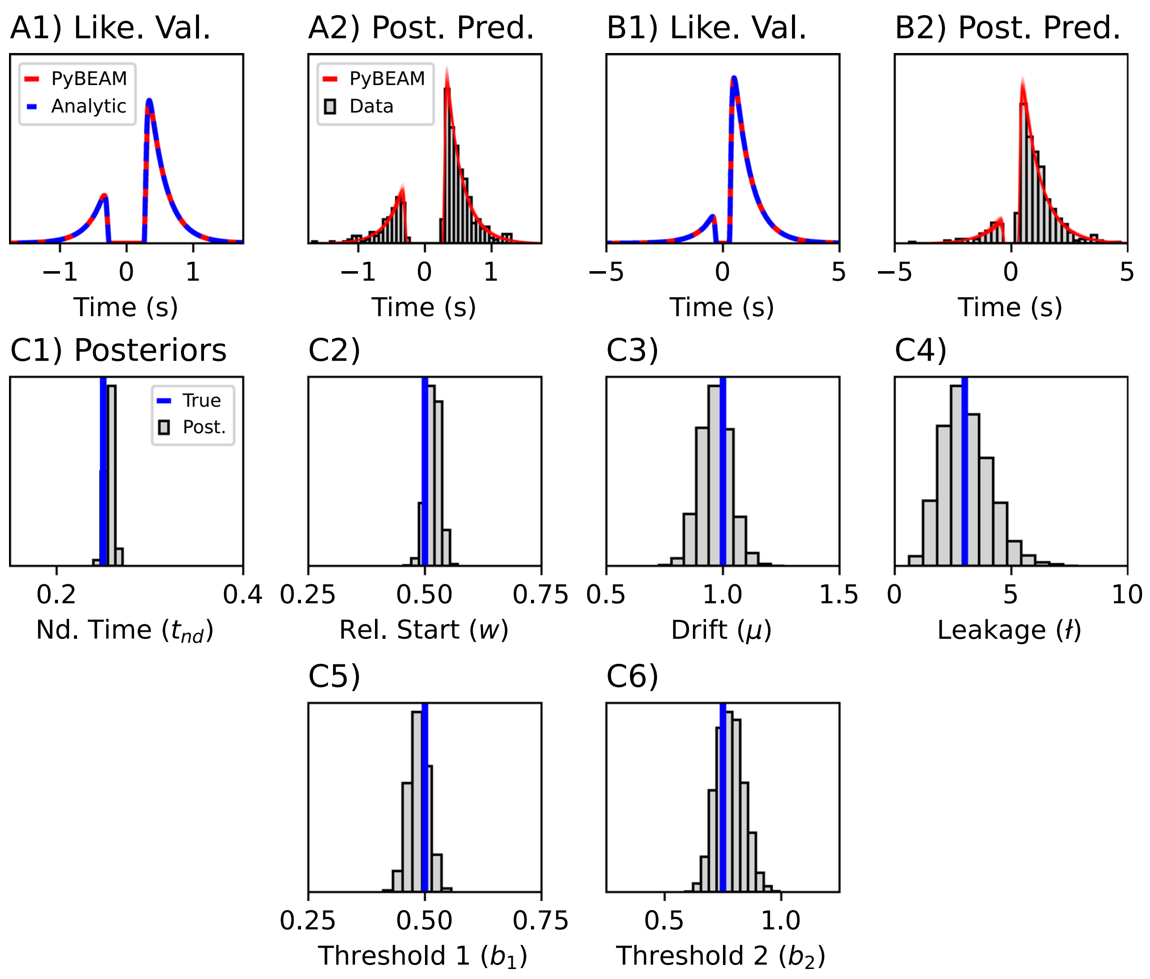

Fig. 5 Results from the leaky integration example. **A)** Likelihood validation (A1) and posterior predictive (A2) for condition 1, low caution. In A1, we plot the PyBEAM solution in red and the analytic solution in blue. In A2, the PyBEAM data fit is in red while the simulated data is grey. **B)** Likelihood validation (B1) and posterior predictive (B2) for condition 2, high caution. Colors have same meaning as in A. **C)** Posteriors for both conditions. Vertical blue lines indicate the true parameter values, while the grey bars are the PyBEAM posteriors. The non-decision time (C1), relative start (C2), drift (C3), and leakage (C4) posteriors are shared between conditions with true values of $t_{nd} = 0.25$, $w = 0.5$, $\mu = 1$, and $l = 3$, respectively. The threshold locations for the low caution b_1 and high caution b_2 conditions are displayed in C5 and C6, with true values of $b_1 = 0.5$ and $b_2 = 0.75$, respectively

Panel groups A and B display the likelihood validation and posterior predictive (described in Section "Simple DDM") for the $\kappa > 1$ and $\kappa < 1$ parameter sets, respectively. In both cases, the PyBEAM numerical solution produces the same result as the analytic solution, and the posterior predictive matches the data set closely. Panel group C displays the posterior predictive for the threshold itself in each case. The blue line corresponds to the true threshold, while the red lines are generated by drawing 100 random random parameter sets from the posterior and plotting their threshold shape. In both the $\kappa > 1$ and $\kappa < 1$ cases, the posterior produces time varying thresholds that closely match that used to generate the data.

Panel groups D and E display the posteriors for the $\kappa > 1$ and $\kappa < 1$ parameter sets, respectively. As discussed in Section "Pre-coded models," for the scale λ and shape κ parameters, we sampled from and report here the log of these parameter values. Both have large reasonable parameter regimes, so sampling in log space guarantees we get good coverage of all possible parameter space.

For the $\kappa > 1$ case, we see good agreement between the true parameter values (blue) and the PyBEAM prediction (grey histogram). However, for the $\kappa < 1$ case, the posteriors predictive for the threshold is slightly different than that of the simulated values, even though the likelihood and threshold recovery from panel groups B and C are very good. This is the result of parameter sloppiness inherent to the Weibull threshold model, where multiple parameter combinations can produce similar likelihood functions. Though this is a common problem with the Weibull model, particu-

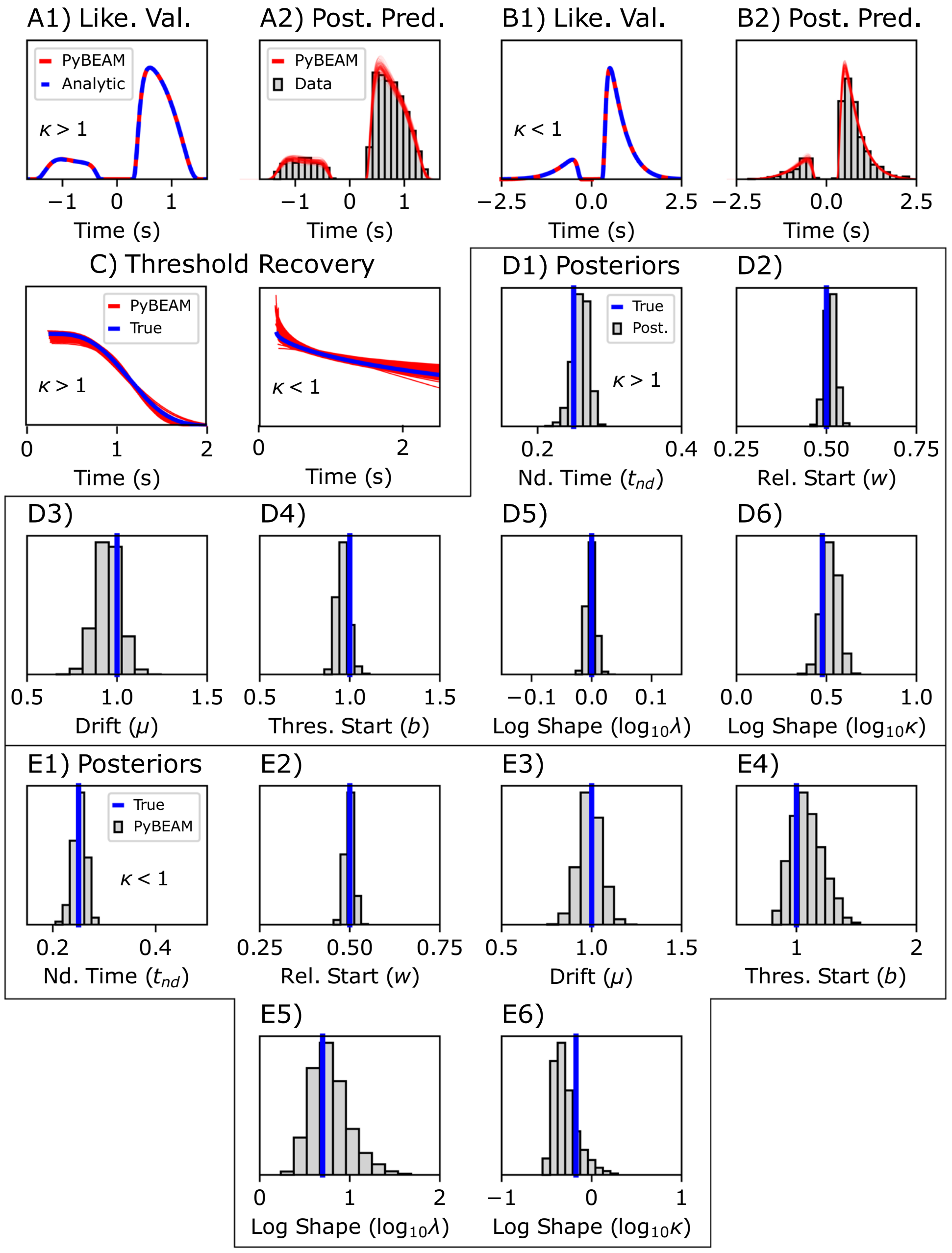

Fig. 6 Results from the changing thresholds example. **A)** Likelihood validation and posterior predictive for the $\kappa > 1$ parameter set. In Panel A1, we plot the PyBEAM likelihood function in red and the analytic likelihood function in blue. In Panel A2, the PyBEAM likelihood predictions are in red while the simulated data is grey. **B)** Likelihood validation (B1) and posterior predictive (B2) for the $\kappa < 1$ parameter set. Colors have same meaning as in A. **C)** Predicted thresholds for both the $\kappa > 1$ and $\kappa < 1$ parameter sets. Red lines show the PyBEAM predicted thresholds, while the blue line is the true threshold. **D)** Posteriors for the $\kappa > 1$ parameter set. Blue lines indicate the true parameter values, while the PyBEAM posteriors are grey. D1 contains the non-decision time posterior, with true value $t_{nd} = 0.25$. D2 contains the relative start posterior, with true value $w = 0.5$. D3 contains the drift rate posterior, with true value $\mu = 1$. D4 contains the threshold start posterior, with true value $b = 1$. D5 contains the log of the shape parameter, with true value $\lambda = 1$. D6 contains the log of the scale parameter, with true value $\kappa = 3$. **E)** Posteriors for the $\kappa < 1$ parameter set. Blue lines indicate the true parameter values, while the PyBEAM posteriors are grey. E1 contains the non-decision time posterior, with true value $t_{nd} = 0.25$. E2 contains the relative start posterior, with true value $w = 0.5$. E3 contains the drift rate posterior, with true value $\mu = 1$. E4 contains the threshold start posterior, with true value $b = 1$. E5 contains the log of the shape parameter, with true value $\lambda = 5$. D6 contains the log of the scale parameter, with true value $\kappa = 0.67$

larly in the $\kappa < 1$ parameter regime, it doesn't significantly affect the interpretation of the results. Different parameter sets produce qualitatively similar decision thresholds.

Urgency gating model with changing information

The final model we recover parameters from is the UGM with time changing stimulus information described in Section "Custom models in PyBEAM." In this model, the drift rate takes the form of Eq. 11, where stimulus information changes from positive to negative at time t_{flip}. Note that since this is a more complex synthetic experimental design (time changing stimuli), fits here are implemented with the custom component of PyBEAM. We fit the model for two data conditions: high drift (condition 1) and low drift (condition 2), corresponding to high and low quality stimulus information, respectively. The reamining parameters are shared between conditions. To validate PyBEAM's parameter recovery for the UGM with changing information, we simulate 500 data points for each condition (1000 total data points). As before, we then use PyBEAM to find the model parameters which best describe this data. We report the results of parameter recovery in Fig. 7.

In Panels A1 and B1, we display the likelihood validation for data conditions 1 and 2, respectively. Since the UGM has no analytic solution, we instead simulate 100,000 data points from the model for each condition (blue histograms) to compare our numerical result to (red). In both cases, the PyBEAM likelihood is once again highly accurate. In Panels A2 and B2, we report the posterior predictive for conditions 1 and 2, respectively. PyBEAM once again performs well at fitting the data.

In Panel group C we report the posteriors output by PyBEAM. In Panels C3 and C4, we report the drift rate posteriors for condition 1 and condition 2, respectively. Panel C7 contains the posterior for the time at which stimulus information changes. The other panels contain the posteriors for the remaining UGM parameters shared between conditions.

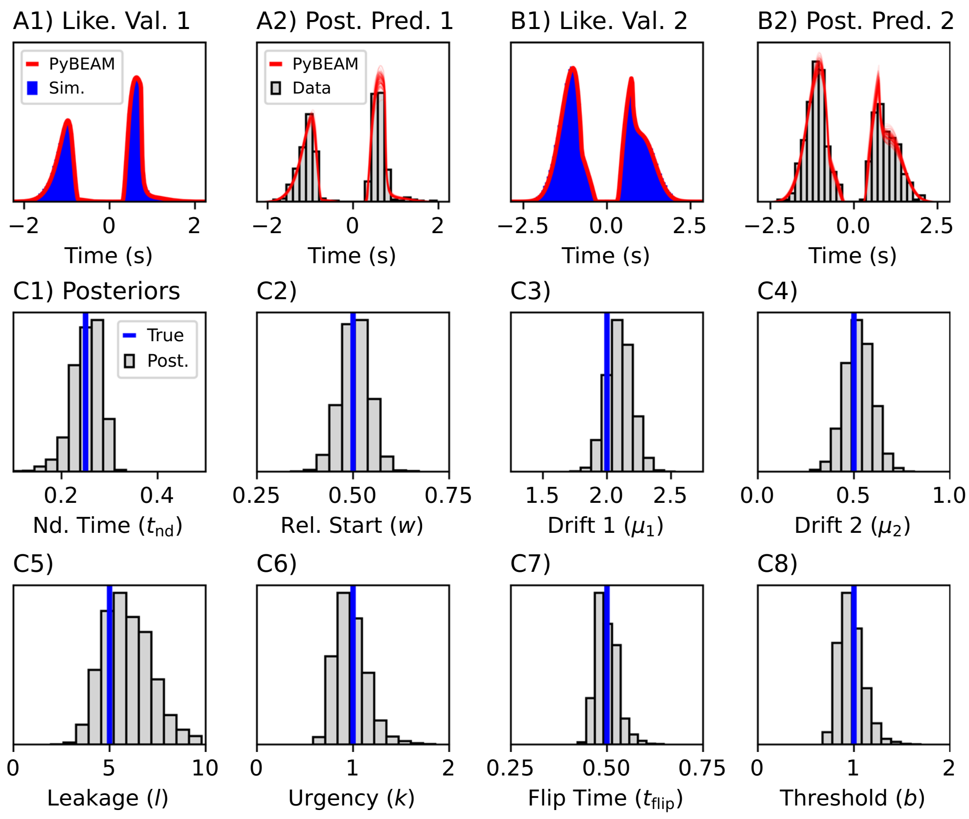

Fig. 7 Results from the UGM with changing information example. **A)** Likelihood validation (A1) and posterior predictive (A2) for condition 1, high drift rate. In panel A1, we plot the PyBEAM solution in red and 100,000 simulated data points in blue. In panel A2, the PyBEAM data fit is in red while the simulated data is grey. **B)** Likelihood validation (B1) and posterior predictive (B2) for condition 2, low drift rate. Colors have same meaning as in A. **C)** Posteriors for all conditions. Blue indicates the true parameter value, while grey are the PyBEAM posteriors. C1 contains the non-decision time posterior, with true value $t_{nd} = 0.25$. C2 contains the relative start posterior, with true value $w = 0.5$. C3 contains the drift rate posterior for condition 1, with true value $\mu = 2$. C4 contains the drift rate posterior for condition 2, with true value $\mu = 0.5$. C5 contains the leakage posterior, with true value $l = 5$. C6 contains the urgency posterior, with true value $k = 1$. C7 contains the flip time posterior, with true value $t_{\text{flip}} = 0.5$. C8 contains the decision threshold posterior, with true value $b = 1$

For all cases, the posteriors (grey) match the true data values (blue) well, indicating effective parameter recovery.

Application to data

Since real data is always messier than simulated data, we last demonstrate the use of PyBEAM to perform parameter inference using choice-RT experimental data collected by Evans, Hawkins, and Brown, 2020. This data set consists of three different experiments. In each, participants made decisions about direction of dot motion in a random dot kinematogram presented at different coherences: 0%, 5%, 10%, and 40%. In the first experiment, sixty-three participants were instructed to maximize reward rate; in the second experiment, seventy-one participants were given a decision deadline; and in the third experiment, one hundred and fifty-four participants were instructed to emphasize speed. The goal of these experiments was to examine if and when participants might utilize collapsing thresholds. To answer this questions, they used a hierarchical Bayesian approach to fit three evidence accumulation models to their data: the simple DDM discussed in Section "Pre-coded EAMs," the full DDM, and a simple DDM with Weibull decision thresholds.

Our goal here is to fit models with and without changing thresholds to each of these data sets and compare results to those obtained by Evans, Hawkins, and Brown, 2020. Specifically, we choose to fit the DDM (excluding across-trial variability in the drift rate, but including across-trial variability in the start point and non-decision times) and Weibull changing thresholds models. We refer to these in the rest of this section as the flat and Weibull models, respectively. The purpose here is not to study the psychological question motivating their work. Rather, the purpose is to assess the efficacy of this method and compare its conclusions to theirs. Their modeling approach would be considered the "gold standard" for this type of modeling. Thus, we use it as a point of comparison while noting that: 1) their implementation utilized fully custom code compared to PyBEAM's more user friendly package approach, and 2) PyBEAM is likely faster than their analytic solution approach

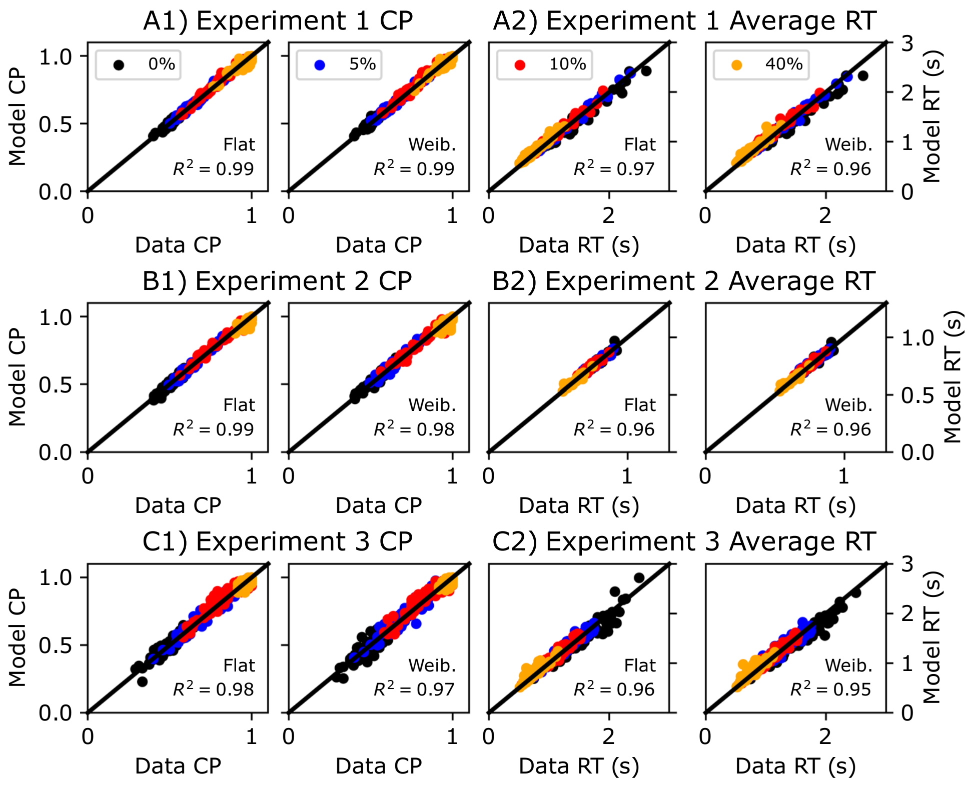

Fig. 8 Choice proportion (CP) and average choice-RT results from PyBEAM for both the flat threshold and Weibull threshold models. The flat model is a DDM (excluding across-trial variability in drift), while the Weibull model is a simple DDM with Weibull decision thresholds. Panels plot the model predictions on the vertical axis and the data values on the horizontal axis. All three experiments share the same legend displayed in the plot's first row. The text on the figure indicates which model is used, either the flat threshold or Weibull thresholds (Weib.), and the correlation coefficient R^2 of the model and data. **A)** CP (A1) and choice-RT (A2) results for Experiment 1. **B)** CP (B1) and choice-RT (B2) results for Experiment 2. **C)** CP (C1) and choice-RT (C2) results for Experiment 3

(Buonocore, Giorno, Nobile, & Ricciardi, 1990; Smith, 2000). Though we cannot directly compare the computational speed of PyBEAM to the Evan's approach, the numerical integration required by it scales quadratically with changes in resolution (Buonocore, Nobile, & Ricciardi, 1987), while Crank-Nicolson's speed scales linearly with resolution. In our experience, this results in a 5-10x slowdown when compared to the PyBEAM approach.

We do make a few notable changes in our fitting methodology compared to that in Evans, Hawkins, and Brown, 2020. First, we do not use a hierarchical approach and instead fit the model to each individual. It is in principle possible to extend PyBEAM to fit hierarchical models, but this is left for future development. Second, we model across-trial variability in only the non-decision time and start point. We do this since, 1) two of the models fit by Evans did not include any across-trial variability, 2) across-trial variability in the drift rate is computationally taxing using our methodology, and 3) across-trial variation in the drift rate is usually not recoverable (Boehm et al., 2018).

We first examine the Choice Proportion (CP) and choice-RT predictions of the model in Fig. 8. Panel groups A1, B1, and C1 compare the CP predicted by the model to that of the data for Experiments 1, 2, and 3, respectively. For all three Experiments, we find good model fits, with correlation coefficients R^2 far exceeding 0.9. Additionally, for all three models, the correlation between model and data is very similar for both the flat and Weibull decision thresholds.

Panel groups A2, B2, and C2 compare the average choice-RT of the model to that of the data for Experiments 1, 2, and 3, respectively. For all three Experiments, both models describe the data well, with R^2 values for both exceeding 0.9. As with the CP graphs, we find that correlation between model and data is very similar for both the flat and Weibull decision thresholds.

In addition to reporting CP and choice-RT data comparison, we also report DIC values for each model in Fig. 9. Specifically, we plot the absolute difference in DIC between the flat thresholds ($\mathrm{DIC_f}$) and the Weibull thresholds ($\mathrm{DIC_w}$). If this value is positive, the changing thresholds model is preferred; if it is negative, the flat threshold is preferred.

We see that, for Experiment 1, the majority of participants favored the flat thresholds model. For Experiment 2, the majority of participants favored the changing thresholds model. For Experiment 3, there is roughly an even split between those which favored the flat and changing threshold models, with a majority preferring the changing thresholds model. Our DIC results also match closely to those determined by Evans. Similar numbers of participants were found to prefer the flat and Weibull threshold models, as shown by Evans in Fig. 2 of their publication (Evans, Hawkins, & Brown, 2020).

We last report the decision thresholds predicted by both the flat and Weibull models in Fig. 10. In the first row, we display the threshold which produces the max log-likelihood for each participant. In row two, we display the average threshold, calculated by averaging together all of the participants' decision thresholds. We find that, for Experiment 1, little collapse occurs for most participants early on in the decision process. It is not until the tail end of the RT distributions that substantial collapse is seen. Thus, for most data points, the decision threshold is similar to that of the flat threshold model. This matches well with the CP and choice-RT analyses which indicated that there was no distinct preference for the changing threshold model over the flat thresholds. Notably, the shape of the average threshold is comparable to that of Evan's results (shown in Fig. 2 of their publication (Evans, Hawkins, & Brown, 2020)).

In Experiment 2, we see that all participants exhibit thresholds which collapse substantially, suggesting that a changing threshold may best describe the data. This matches the results of the CP, choice-RT, and DIC results which indicated that changing thresholds are the preferred model. This also matches the results of Evans, which indicated that

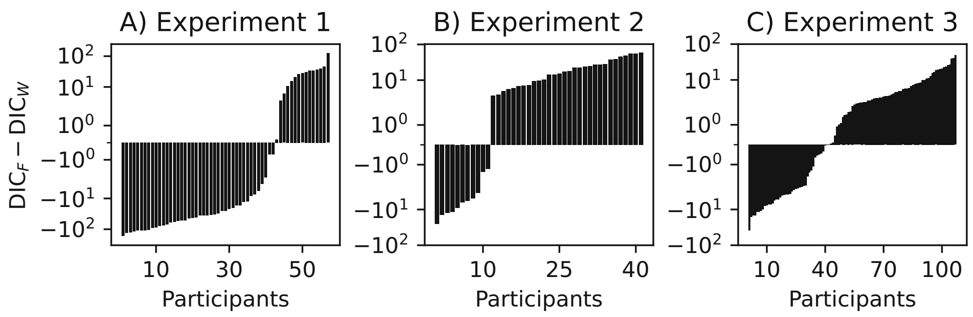

Fig. 9 Absolute difference between the flat threshold DIC ($\mathrm{DIC_f}$) and the Weibull threshold DIC ($\mathrm{DIC_w}$) for Experiment 1 (A), Experiment 2 (B), and Experiment 3 (C). The flat model is a DDM (excluding across-trial variability in drift), while the Weibull model is a simple DDM with Weibull decision thresholds. Participants are ordered by how the large this difference is, with smallest on the left and largest on the right. If the absolute difference is positive, the Weibull model is favored. If negative, the flat model is favored

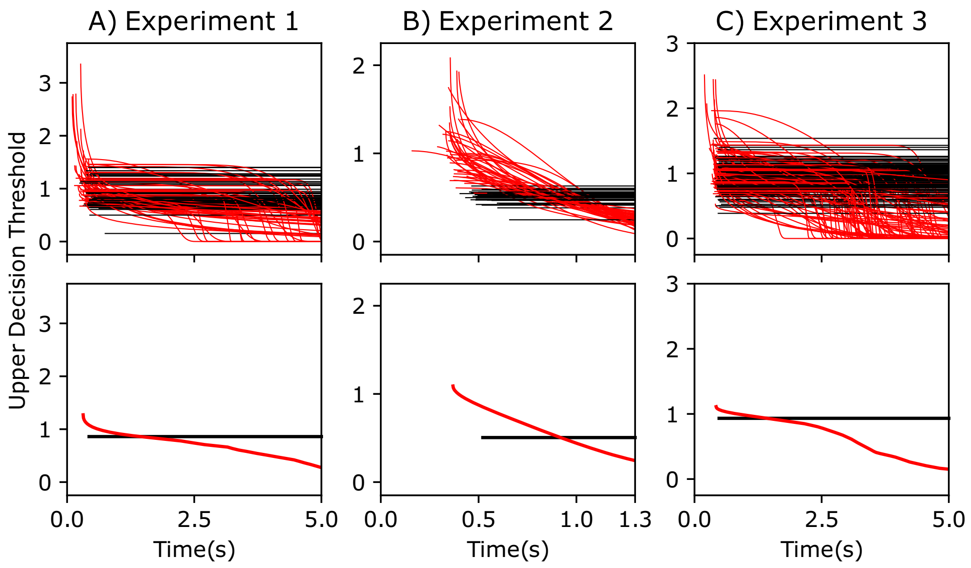

Fig. 10 Upper decision thresholds predicted by the model for Experiment 1 (A), Experiment 2 (B), and Experiment 3 (C) versus time. The upper and lower thresholds are symmetric, so the upper is only displayed here. Black lines are for the flat model (DDM excluding across-trial variability), while the red lines are for the Weibull model (simple DDM with Weibull thresholds). The first row displays the decision thresholds for each participant, while the second row averages all decision thresholds together

Experiment 2 had the clearest support for changing decision thresholds.

Lastly, in Experiment 3, we observe decision thresholds which vary substantially from one participant to the next. For this data set, some participants had mean choice-RTs near a second, while others had mean choice-RTs upwards of three and four seconds. This difference in response time produces dramatically different thresholds for each individual which makes describing the average behavior challenging. In spite of this, the average threshold we calculate is very similar to that determined by Evans (shown in Fig. 2 of their publication (Evans, Hawkins, & Brown, 2020)). Further, it also supports the conclusions of the CP, choice-RT, and DIC results which suggested that Experiment 3 had no clear model preference.

Limitations of approach

While, based on our understanding of the literature, the approach presented here has the greatest combination of accuracy, efficiency, and generality of methods for fitting EAMs, it does have a number of limitations. The largest current limitation is that we do not yet implement hierarchical models, which are increasingly common in this literature. This is neither a theoretical nor a computational limitation. Rather, it is a result of the way PyMC computationally handles the back-end of hierarchical models with its relatively new differential evolution sampler. We do not go into detail here since it is mainly a computational structure issue. However, we investigated this in detail, and the likelihood generation approach and implementation here can be directly used with hierarchical models without alteration with an appropriate sampler. We could write such a sampler (and have one for personal use in Matlab), however that would reduce the usability of this approach significantly and we wanted to work within the confines of PyMC capabilities.

A second limitation is that this approach does not easily allow for across-trial variability in drift rates, which is commonly invoked for the DDM. As discussed, variability in non-decision time and start point are easily included with little appreciable computational cost. Drift rate variability is however more complicated. We could take the same approach of HDDM (Wiecki, Sofer, & Frank, 2013) and integrate over these distributions; however, that would significantly increase compute cost (as it does with HDDM). For general models, it is not clear if such a feature would be identifiable and thus we have not yet included it. There is however a direct path do doing so in the future if it becomes necessary or compute capabilities reduce the cost of doing so.

Discussion

Evidence Accumulation Models are one of the most dominant classes of computational models used to study decision making behavior. However, to date, few tools are available which allow researchers to study complex EAMs. Further, Bayesian methods have generally been too slow for practical use, restricting model fitting to techniques like max log-likelihood or quantile maximization.

In this work, we introduced the Python package PyBEAM to provide an easy to use tool for Bayesian inference of complex EAMs. Unlike previous methods which have either generated likelihood functions using simulated SDEs or slow analytic solutions, PyBEAM instead uses a Fokker-Planck equation, dramatically improving both accuracy and speed. In addition to being fast, it increases flexibility by allowing custom inputs for the drift rate, diffusion rate, and decision thresholds. PyBEAM then pairs this likelihood construction method with the Python package PyMC (Salvatier, Wiecki, & Fonnesbeck, 2016) for rapid Bayesian inference of model parameter sets. Using PyBEAM, more complex models can be fit to data with both high precision and high speed. Further, to make this type of modeling accessible to others, we provide a Python based package which allows exploration of complex models with a relatively low cost of entry. Our hope is that this framework will allow a wider array of researchers to explore more complex hypotheses (encoded in models) using more complex experimental designs.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.3758/s13428-023-02162-w.

References

Boehm, U., Annis, J., Frank, M. J., Hawkins, G. E., Heathcote, A., Kellen, 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

Boehm, U., Cox, S., Gantner, G., & Stevenson, R. (2021). Fast solutions for the first-passage distribution of diffusion models with space-time-dependent drift functions and time-dependent boundaries. *Journal of Mathematical Psychology, 105*, 102613. https://doi.org/10.1016/.jmp.2021.102613

Boehm, U., Cox, S., Gantner, G., & Stevenson, R. (2022). Efficient numerical approximation of a non regular fokker-planck equation associated with first passage time distributions. *BIT Numerical Mathematics, 62*, 1355–1382. https://doi.org/10.1007/s10543-022-00914-2

Braak, C. J. F. T. (2006). A markov chain monte carlo version of the genetic algorithm differential evolution: Easy bayesian computing for real parameter spaces. *Statistics and Computing, 16*, 239–249. https://doi.org/10.1007/s11222-006-8769-1

Braak, C. J. F. T., & Vrugt, J. A. (2008). Differential evolution markov chain with snooker updater and fewer chains. *Statistics and Computing, 18*, 435–446. https://doi.org/10.1007/s11222-008-9104-9

Brown, S. D., & Heathcote, A. (2008). The simplest complete model of choice response time: Linear ballistic accumulation. *Cognitive Psychology, 57*(3), 153–178. https://doi.org/10.1016/j.cogpsych.2007.12.002

Buonocore, A., Giorno, V., Nobile, A. G., & Ricciardi, L. M. (1990). On the two-boundary first-crossing-time problem for diffusion processes. *Journal of Applied Probability, 27*(1), 102–114. https://doi.org/10.2307/3214598

Buonocore, A., Nobile, A. G., & Ricciardi, L. M. (1987). A new integral equation for the evaluation of first-passage-time probability densities. *Advances in Applied Probability, 19*(4), 784–800. https://doi.org/10.2307/1427102

Busemeyer, J. R., Gluth, S., Rieskamp, J., & Turner, B. M. (2019). Cognitive and neural bases of multi-attribute, multi-alternative, value-based decisions. *Trends Cogn Sci, 23*(3), 251–263

Chandrasekaran, C., & Hawkins, G. E. (2019). Chartr: An r toolbox for modeling choice and response times in decision-making tasks. *Journal of Neuroscience Methods, 328*. https://doi.org/10.1016/j.jneumeth.2019.108432

Cisek, P., Puskas, G. A., & El-Murr, S. (2009). Decisions in changing conditions: The urgency-gating model. *Psychological Review, 29*(37), 11560–11571. https://doi.org/10.1523/JNEUROSCI.1844-09.2009

Crank, J. (1984). *Free and moving boundary problems*. Oxford University Press.

Crank, J., & Nicolson, P. (1947). A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. *Cambridge University Press, 50–67*,. https://doi.org/10.1017/S0305004100023197

Demetropolis(z): Population vs. history efficiency comparison. (2022, March). https://docs.pymc.io/en/v3/pymc-examples/examples/samplers/DEMetropolisZ_EfficiencyComparison.html

Drugowitsch, J., Moreno-Bote, R., Churchland, A. K., Shadlen, M. N., & Pouget, A. (2012). The cost of accumulating evidence in perceptual decision making. *The Journal of Neruoscience, 32*(11), 3612–3628. https://doi.org/10.1523/JNEUROSCI.4010-11.2012

Dutilh, G., Annis, J., Brown, S. D., Cassey, P., Evans, N. J., Grasman, R. P., ... others (2019). The quality of response time data inference: A blinded, collaborative assessment of the validity of cognitive models. *Psychonomic bulletin & review, 26*(4), 1051–1069. https://doi.org/10.3758/s13423-017-1417-2

Evans, N. J., Brown, S. D., Mewhort, D. J. K., & Heathcote, A. (2018). Refining the law of practice. *Psychological review, 125*(4), 592–605. https://doi.org/10.1037/rev0000105

Evans, N. J., Hawkins, G. E., & Brown, S. D. (2020). The role of passing time in decision-making. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 46*(2), 316–326. https://doi.org/10.1037/xlm0000725

Evans, N. J., Holmes, W. R., Dasari, A., & Trueblood, J. S. (2021). The impact of presentation order on attraction and repulsion effects in decision-making. *Decision, 8*(1), 36. https://doi.org/10.1037/dec0000144

Evans, N. J., Holmes, W. R., & Trueblood, J. S. (2019). Response-time data provide critical constraints on dynamic models of multi-alternative, multi-attribute choice. *Psychon Bull Rev, 26*(3), 901–933. https://doi.org/10.3758/s13423-018-1557-z

Evans, N. J., Trueblood, J. S., & Holmes, W. R. (2020). A parameter recovery assessment of time-variant models of decision-making. *Behavior research methods, 52*(1), 193–206. https://doi.org/10.3758/s13428-019-01218-0

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

Fontanesi, L., Gluth, S., Spektor, M., & Rieskamp, J. (2019). A reinforcement learning diffusion decision model for value-based decisions. *Psychon Bull Rev, 26*, 1099–1121. https://doi.org/10.3758/s13423-018-1554-2

Frazier, P.I., & Yu, A.J. (2007). Sequential hypothesis testing under stochastic deadlines. *NIPS,* 465–472. Retrieved from https://proceedings.neurips.cc/paper/2007/file/9c82c7143c102b71c593d98d96093fde-Paper.pdf

Hawkins, G. E., Forstmann, B. U., Wagenmakers, E. J., Ratcliff, R., & Brown, S. D. (2015). Revisiting the evidence for collapsing boundaries and urgency signals in perceptual decision-making. *The Journal of Neuroscience, 35*(6), 2476–2484. https://doi.org/10.1523/JNEUROSCI.2410-14.2015

Heathcote, A., Brown, S., & Mewhort, D. J. K. (2002). Quantile maximum likelihood estimation of response time distributions. *Psychon Bull Rev, 9*(2), 394–401. https://doi.org/10.3758/bf03196299

Heathcote, A., Lin, Y.-S., Reynolds, A., Strickland, L., Gretton, M., & Matzke, D. (2019). Dynamic models of choice. *Behavior Research Methods, 51*, 961–985. https://doi.org/10.3758/s13428-018-1067-y

Hoffman, M. D., & Gelman, A. (2014). The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research, 15*, 1593–1623.

Holmes, W. R. (2015). A practical guide to the probability density approximation (pda) with improved implementation and error characterization. *Journal of Mathematical Psychology, 68*(69), 13–24. https://doi.org/10.1016/j.jmp.2015.08.006

Holmes, W. R., O'Daniels, P., & Trueblood, J. S. (2020). A joint deep neural network and evidence accumulation modeling approach to human decision-making with naturalistic images. *Computational Brain & Behavior, 3*(1), 1–12. https://doi.org/10.1007/s42113-019-00042-1

Holmes, W. R., & Trueblood, J. S. (2018). Bayesian analysis of the piecewise diffusion decision model. *Behavior Research Methods, 50*(2), 730–743. https://doi.org/10.3758/s13428-017-0901-y

Holmes, W. R., Trueblood, J. S., & Heathcote, A. (2016). A new framework for modeling decisions about changing information: The piecewise linear ballistic accumulator model. *Cognitive psychology, 85*, 1–29. https://doi.org/10.1016/j.cogpsych.2015.11.002

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., ... Willing, C. (2016). Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides & B. Schmidt (Eds.), Positioning and power in academic publishing: Players, agents and agendas (p. 87–90). https://doi.org/10.3233/978-1-61499-649-1-87

Krajbich, I., Armel, C., & Rangel, A. (2010). Visual fixations and the computation and comparison of value in simple choice. *Nature Neuroscience, 13*(10), 1292–1298. https://doi.org/10.1038/nn.2635

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

Lerche, V., Christmann, U., & Voss, A. (2018). Impact of context information on metaphor elaboration a diffusion model study. *Experimental Psychology, 65*(6), 370–384. https://doi.org/10.1027/1618-3169/a000422

Lin, Y.-S., Heathcote, A., & Holmes, W. R. (2019). Parallel probability density approximation. *Behavior research methods, 51*(6), 2777–2799. https://doi.org/10.3758/s13428-018-1153-1

Navarro, D. J., & Fuss, I. G. (2009). Fast and accurate calculations for first-passage times in wiener diffusion models. *Journal of Mathematical Psychology, 53*(4), 222–230. https://doi.org/10.1016/j.jmp.2009.02.003

Nosofsky, R. M., Little, D. R., Donkin, D., & Fific, M. (2011). Short-term memory scanning viewed as exemplar-based categorization. *Psychological review, 118*(2), 280–315. https://doi.org/10.1037/a0022494

Nosofsky, R. M., & Palmeri, T. J. (1997). An exemplar-based random walk model of speeded classification. *Psychological review, 104*(2), 266–300. https://doi.org/10.1037/0033-295x.104.2.266

Østerby, O. (2003). Five ways of reducing the crank-nicolson oscillations. *BIT Numerical Mathematics, 43*, 811–822. https://doi.org/10.1023/B:BITN.0000009942.00540.94

Osth, A. F., & Farrell, S. (2019). Using response time distributions and race models to characterize primacy and recency effects in free recall initiation. *Psychological review, 126*(4), 578–609. https://doi.org/10.1037/rev0000149

Öttinger, H.C. (1996). Stochastic processes in polymeric fluids. Springer-Verlag Berlin Heidelberg. Retrieved from https://link.springer.com/book/10.1007/978-3-642-58290-5

Ratcliff, R. (1978). A theory of memory retrieval. *Psychological review, 85*(2), 59–108. https://doi.org/10.1037/0033-295X.85.2.59

Ratcliff, R., & McKoon, G. (2008). The diffusion decision model: Theory and data for two-choice decision tasks. *Neural Computation, 20*(4), 873–922. https://doi.org/10.1162/neco.2008.12-06-420

Ratcliff, R., & Rouder, J.N. (1998). Modeling response times for two-choice decisions. psychological science. Psychological Science, 9(5), 347–356. https://doi.org/10.1111/1467-9280.00067

Ratcliff, R., & Tuerlinckx, F. (2002). Estimating parameters of the diffusion model: Approaches to dealing with contaminant reaction times and parameter variability. *Psychon Bull Rev, 9*(3), 438–481. https://doi.org/10.3758/BF03196302

Ratcliff, R., Zandt, T. V., & McKoon, G. (1999). Connectionist and diffusion models of reaction time. *Psychological Review, 106*(2), 261–300. https://doi.org/10.1037/0033-295x.106.2.261

Richter, T., Ulrich, R., & Janczyk, M. (2023). Diffusion models with time-dependent parameters: An analysis of computational effort and accuracy of different numerical methods. *Journal of Mathematical Psychology, 114*, 102756. https://doi.org/10.1016/j.jmp.2023.102756

Salvatier, J., Wiecki, T.V., & Fonnesbeck, C. (2016). Probabilistic programming in python using pymc3. *PeerJ Computer Science, 2*(55). https://doi.org/10.7717/peerj-cs.55

Servant, M., White, C., Montagnini, A., & Burle, B. (2016). Linking theoretical decision-making mechanisms in the simon task with electrophysiological data: A model-based neuroscience study in human. *Journal of Cognitive Neuroscience, 28*(10), 1501–1521. https://doi.org/10.1162/jocn_a_00989

Shinn, M., Lam, N.H., & Murray, J.D. (2020). A flexible framework for simulating and fitting generalized drift-diffusion models. *eLife, 9*. https://doi.org/10.7554/eLife.56938

Smith, P. L. (2000). Stochastic dynamic models of response time and accuracy: A foundational primer. *The Journal of Mathematical Psychology, 44*(2), 408–463. https://doi.org/10.1006/jmps.1999.1260

Tajima, S., Drugowitsch, J., & Pouget, A. (2016). Optimal policy for value-based decision-making. *Nature Communications, 7*, 12400. https://doi.org/10.1038/ncomms12400

Trueblood, J. S., Heathcote, A., Evans, N. J., & Holmes, W. R. (2021). Urgency, leakage, and the relative nature of information processing in decision-making. *Psychological Review, 128*(1), 160–186. https://doi.org/10.1037/rev0000255

Trueblood, J. S., Holmes, W. R., Seegmiller, A. C., Douds, J., Compton, M., Szentirmai, E., ... Eichbaum, Q. (2018). The impact of speed and bias on the cognitive processes of experts and novices in medical image decision-making. *Cognitive Research: Principles and Implications, 3*(1), 1–14. https://doi.org/10.1186/s41235-018-0119-2

Turner, B. M., Forstmann, B. U., Wagenmakers, E. J., Brown, S. D., Sederberg, P. B., & Steyvers, M. (2013). A bayesian framework for simultaneously modeling neural and behavioral data. *Neuroimage, 72*, 193–206. https://doi.org/10.1016/j.neuroimage.2013.01.048

Turner, B. M., Rodriguez, C. A., Norcia, T. M., McClure, S. M., & Steyvers, M. (2016). Why more is better: Simultaneous modeling of eeg, fmri, and behavioral data. *Neuroimage, 128*, 96–115. https://doi.org/10.1016/j.neuroimage.2015.12.030

Turner, B. M., & Sederberg, P. B. (2014). A generalized, likelihood-free method for posterior estimation. *Psychon Bull Rev, 21*(2), 227–250. https://doi.org/10.3758/s13423-013-0530-0

Turner, B. M., van Maanen, L., & Forstmann, B. U. (2015). Informing cognitive abstractions through neuroimaging: The neural drift diffusion model. psychological review. *Psychological Review, 122*(2), 312–336. https://doi.org/10.1016/j.tics.2018.12.003

Turner, B. M., & Van Zandt, T. (2018). Approximating bayesian inference through model simulation. *Trends in cognitive sciences, 22*(9), 826–840. https://doi.org/10.1016/j.tics.2018.06.003

Usher, M., & McClelland, J. L. (2001). The time course of perceptual choice: The leaky, competing accumulator model. *Psychological Review, 108*(3), 550–592. https://doi.org/10.1037/0033-295x.108.3.550

Vandekerckhove, J., & Tuerlinckx, F. (2008). Diffusion model analysis with matlab: A dmat primer. *Behavior Research Methods, 40*, 61–72. https://doi.org/10.3758/BRM.40.1.61

Voss, A., & Voss, J. (2007). Fast-dm: A free program for efficient diffusion model analysis. *Behavior Research Methods, 39*, 767–775. https://doi.org/10.3758/BF03192967

Voss, A., & Voss, J. (2008). A fast numerical algorithm for the estimation of diffusion model parameters. *Journal of Mathematical Psychology, 52*(1), 1–9. https://doi.org/10.1016/j.jmp.2007.09.005

Wagenmakers, E. J., Ratcliff, R., Gomez, P., & McKoon, G. (2008). A diffusion model account of criterion shifts in the lexical decision task. *Journal of Memory and Language, 58*(1), 140–159. https://doi.org/10.1016/j.jml.2007.04.006

Wiecki, T., Sofer, I., & Frank, M. (2013). Hddm: Hierarchical bayesian estimation of the drift-diffusion model in python. *Frontiers in Neuroinformatics, 7*. https://doi.org/10.3389/fninf.2013.00014