Amortized Inference with User Simulations

Hee-Seung Moon[1,2], Antti Oulasvirta[2], Byungjoo Lee[1]
[1]Yonsei University, Republic of Korea
[2]Aalto University, Finland

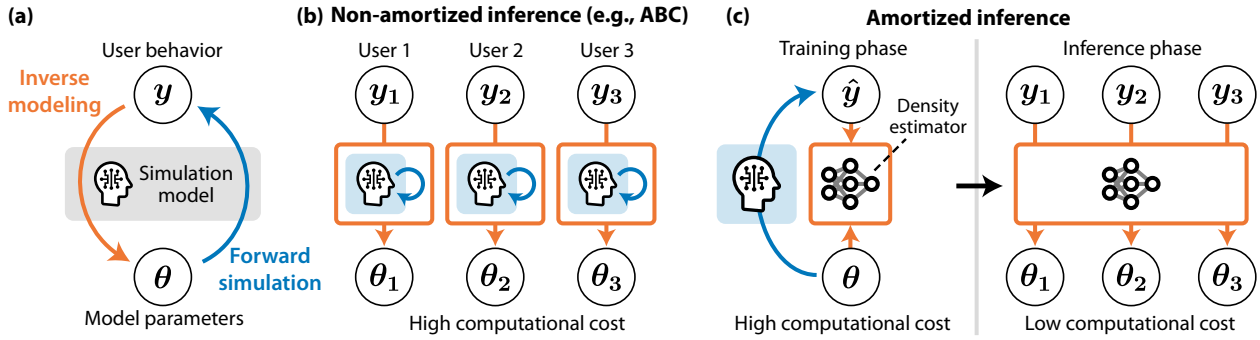

Figure 1: (a) *Inverse modeling* is the process of inferring parameters of user-simulation models from given user-behavior data. (b) Previous inverse modeling approach (e.g., approximate Bayesian computation, or ABC) requires iterating behavior simulations to seek those parameters best describing the given dataset. The iteration comes at a high computational cost (consuming even several days) and must be repeated each time a new user behavior is added. (c) In contrast, *amortized inference* enables rapid inference for every new user behavior. In the one-time training phase, a neural-network-based density estimator learns the probabilistic relationship between the model's parameters and simulated behaviors, at high computational cost. Then, in the inference phase, the trained density estimator takes the user-behavior observations as input and infers the posterior distribution of the parameters, at low computational cost: less than a second.

ABSTRACT

There have been significant advances in simulation models predicting human behavior across various interactive tasks. One issue remains, however: identifying the parameter values that best describe an individual user. These parameters often express personal cognitive and physiological characteristics, and inferring their exact values has significant effects on individual-level predictions. Still, the high complexity of simulation models usually causes parameter inference to consume prohibitively large amounts of time, as much as days per user. We investigated *amortized inference* for its potential to reduce inference time dramatically, to mere tens of milliseconds. Its principle is to pre-train a neural proxy model for probabilistic inference, using synthetic data simulated from a range of parameter combinations. From examining the efficiency and prediction performance of amortized inference in three challenging cases that involve real-world data (menu search, point-and-click, and touchscreen typing), the paper demonstrates that an amortized-inference approach permits analyzing large-scale datasets by means of simulation models. It also addresses emerging opportunities and challenges in applying amortized inference in HCI.

CHI '23, April 23–28, 2023, Hamburg, Germany

ACM ISBN 978-1-4503-9421-5/23/04…$15.00
https://doi.org/10.1145/3544548.3581439

CCS CONCEPTS

• **Human-centered computing** → **User models**; • **Mathematics of computing** → **Density estimation**.

KEYWORDS

simulation models, inverse modeling, amortized inference

ACM Reference Format:
Hee-Seung Moon, Antti Oulasvirta, and Byungjoo Lee. 2023. Amortized Inference with User Simulations. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23), April 23–28, 2023, Hamburg, Germany*. ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3544548.3581439

1 INTRODUCTION

Simulation models of human–computer interaction (HCI) are computer programs that produce moment-by-moment predictions of an interactive user's action (e.g., [30, 31, 49, 51, 54, 68, 74]). In technical terms, they consist of *model parameters* and executable *mechanisms*, which together produce simulations. The mechanisms represent the processes that underlie user behavior, whereas models' parameters in general describe cognitive, physiological, or biomechanical characteristics of a simulated user. For example, parameters in a model of point-and-click behavior [15] characterize the noise of the human visual and motor systems, and a model of menu-search behavior [10] has parameters linked to memory recall. Such parameters are latent (not directly observable) and differ in values between individual users [34] or user groups [69]. Through finding

the appropriate values, the model can accurately reflect behavioral differences between individuals.

In *inverse modeling*, the goal is to infer model parameters – that is, computationally find the most plausible parameters that explain a given set of observed behaviors (see Figure 1(a)). In statistics terms, inverse modeling involves *parameter fitting*. Solving the inverse modeling problem could enrich various applications in HCI. A model with well-inferred parameters can predict individual users' behavior in the given task; therefore, related individual-level parameters often have been regarded as essential ingredients for personalizing user interfaces [68, 69]. Inverse modeling informs not only developing explanations from observations but also devising new simulation models for HCI work. For example, it aids in verifying whether a particular posited mechanism correctly captures the phenomenon (e.g., matching empirical datasets) while the inferred parameters correspond to prior knowledge of user attributes or in identifying the most credible model from among several candidates [34]. To exploit this potential more fully, inverse modeling often emphasizes finding the entire distribution for a parameter rather than just a single point value [35].

However, parameter inference for the HCI field's simulation models has proven prohibitively computation-intensive: inferring parameters from observations takes a very long time. The great complexity of today's user simulation models – arising from their hierarchical structure, the need to optimize policies, etc. – rules out obtaining a probabilistic representation of user behavior for given inputs to the model. Statisticians refer to the likelihood function as intractable. Since these conditions preclude fast likelihood-based inference methods such as maximum likelihood estimation (MLE), we have had to resort to simulation-based inference techniques, such as approximate Bayesian computation (ABC) [4]. Relying on repetitive simulations, they search possible ranges of parameters to estimate the values that best replicate given observations (see Figure 1(b)). These methods scale poorly, though [11]. Researchers report that the parameter-search process requires immense computation time, from several hours [18, 49] to days [34]. For each new datum for inference (e.g., observed behavior of an added user), the entire parameter-search process must be repeated from scratch.

In light of this issue, the paper examines *amortized inference* as a novel workflow that mitigates the computation-cost problem in HCI's inverse modeling. Amortized inference is an emerging approach for machine learning designed to address the "likelihood-free inference problem" [11, 23, 38, 58, 61, 72]. It uses *upfront* computations to speed up future inference; i.e., the computation cost for inference is *amortized*. This is accomplished through training a proxy model – specifically, a *density estimator* – by means of deep neural networks (see Figure 1(c)). The estimator functions to estimate parameters' posterior probability when given observed behavior and to represent the complex posterior distribution over the full range of these parameters. Training a density estimator requires a large synthetic dataset produced by running the model with various parametric inputs that could plausibly occur within the domain. The training phase entails high computation cost but takes place only once. Once trained, the density estimator can, in the second phase (inference), quickly and continually infer individual users' model parameters (with the posterior distributions) from their behavior data.

We studied and developed amortized inference for the purpose of fitting simulation models to individual-level HCI data. Three motives lay behind this: an interest in gauging possible improvements in computation efficiency; our wish to study the prediction accuracy achievable, comparing the levels with those from non-amortization methods such as ABC [34, 35]; and, on the assumption of promising results, aims of exploring amortized inference for scaling individual-level fitting to large datasets for HCI. Our central goal was to see whether amortized inference could aid in estimating full distributions of cognitive parameters in a user population, something beyond the reach of previous methods.

We begin the discussion by presenting the novel workflow for applying amortized inference in HCI. Then, we report on our validation of it via three challenging cases that entailed fitting previously published simulation models to datasets from actual users. For each case, we trained a density estimator from only the model's simulated data, then used it for inference on real user data. For Case 1 (N=18), we fitted a reinforcement-learning-based user model of menu search [10] to infer cognitive parameters (e.g., eye fixation duration) from aggregated user task-performance metrics (e.g., averaged completion time over whole trials). With Case 2 (N=20), we fitted a point-and-click model [15] to infer parameters of the visual and motor system. For this, we employed complex trajectory data (cursor trajectories from multiple trials) from an individual. Case 3 involved fitting a touchscreen-typing model [30] to a large real-world dataset [56] that had not been used for testing that model before. We inferred the distributions of model parameters (e.g., such as the fingers' motor performance) for 1,057 individual users from keylog data in an online typing experiment. Also, we report the results from testing the model on a user task not encompassed by the training data, and we examine the effect of having a strong prior. The paper concludes with suggested practical applications of the low-cost inverse modeling approach, in light of our findings, and discussion of challenges that remain.

The paper offers three main contributions to inverse modeling in the HCI field:

(1) Introduction of a workflow for applying amortized inference in modern user simulation models for HCI, with demonstrated ability to reduce computation costs significantly
(2) Validating the approach across three published case studies from CHI conferences, comparing with known baselines, and reporting efficiency and accuracy measurements
(3) Open-source release of the entire code implementation from the study[1] for future research into amortized inference in HCI models

2 RELATED WORK

Simulation models represent latent processes that are theorized to play out when a user interacts with a computer. This sets the approach apart from mathematical modeling, which is employed most often to predict aggregate outputs such as task-performance metrics [17, 64, 67, 71, 78] and from data-driven models, which consider the dataset rather than the user [28, 46, 50]. Simulations based on a cognitive-modeling architecture (GOMS [8], ACT-R [1],

[1] https://github.com/hsmoon121/amortized-inference-hci

EPIC [36], etc.) represent depict processes as a system of interconnected processing modules. Early work in this field was restricted by the need for task-specific manual description of the program being executed within the given architecture. Recent work aimed at avoiding the limitations of such hand-coded rule systems applies machine-learning (ML) methods: a *control policy* is learned automatically. The policy captures a user's tendency to act in a particular way given a particular (internal or external) state. This approach's principles fall under the concept of *computational rationality*, the notion that those latent processes are organized in service of maximizing expected value via action [20, 44, 53, 59]. Researchers have directed RL and, more recently, deep RL toward estimating computationally rational policies, with applications demonstrated in menu search [10], point-and-click [15], touchscreen typing [30], button pressing [54], mid-air pointing [9, 16], learning of layouts [32], visual search [33], driving [31], and multitasking [18].

The question addressed in this paper is how to determine those parameters in a simulation model that represent unobservable characteristics of the user. Prior simulation models have either used values reported in the literature or set the parameters via fitting to a dataset. With our study, we sought to facilitate user simulation models' construction and, furthermore, enable rapid individual-level simulation by developing and introducing a more efficient method for inferring parameters from real user behavior.

2.1 Inverse Modeling with User Simulations

The inverse modeling process ascertains the best model parameters for a given dataset. After inversion, one can approach the model's parameters as hypotheses about the latent characteristics of interest. Where the model represents the user's behavior via a simple closed-form expression, researchers can, from the parameters, calculate the likelihood of any given observation or a particular discrepancy between a prediction and what actually happened. Fitts' law [6, 7, 17, 42, 63], the diffusion model for binary decisions [64, 65], the classic model for visual perception of speed [71], and the integrated eye-movement model [67] are some of the many traditional user models with parameters determined through MLE or least-squares estimation. This approach does not permit the simulation models' inversion, however: since most user simulation models involve complex relationships without a known likelihood function, likelihood-based fitting methods (MLE etc.) cannot invert them. Hence, many previous studies adopt values from the literature or tune parameters manually.

ABC has emerged as a popular likelihood-free approach for inverting simulation models [4, 11, 27]. It offers a systematic way of finding the parameters that replicate a given dataset well in the model's parameter space and thus revealing the parameters' posterior distribution from the observed data. Kangasrääsiö *et al.* [34] first attempted to apply ABC to fit an RL-based simulation of menu selection. However, its application to user simulation models is plagued by the time-efficiency problem. Running the multiple simulations required with various parameter candidates is time-consuming when the simulation model insists on optimizing its control policy anew for each candidate. Moon *et al.* [49] introduced a way of addressing this. They showed that the control policy of user simulation models can be generalized across varying model parameters by means of multi-task RL. Consequently, the ABC process, which had previously required days of computation [34], may be reduced to one or two hours. Nonetheless, as is hardly surprising, individual-level fitting based on ABC is normally restricted to smaller numbers of users (e.g., 5 [34] or 20 [49]). Though one recent study [18] of task-interleaving behavior employed ABC to fit an RL-based model to the data of 211 individuals, fitting this model to an individual user took 1.5 hours of computation time. Our motivation in delving into amortized inference was to see whether it could overcome these issues, thereby opening the door to larger-scale and real-time applications of inverse modeling in HCI.

2.2 Amortized Inference

Amortized inference consists of "investing upfront computation to support rapid online inference" [72]. With modern ML methods, a complex probability distribution can be estimated through variational inference [5, 38, 77], which uses optimization of tractable and parameterized distributions to approximate the target distribution. One can amortize this variational inference by building a neural proxy model, a *conditional density estimator* that learns an effective mapping from a given observation (e.g., an individual's behavior) to an approximate distribution (e.g., a posterior of model parameters) [38]. Once trained through upfront computation, the conditional density estimator enables rapid inference of the posterior of the parameters conditioned on the given observation.

Researchers have examined many ML methods in efforts to solve this conditional density estimation problem [11, 19, 45, 55]. One critical factor is the neural networks' representation capacity – they must be able to learn the complex probability distribution. Early methods combined Gaussians, to estimate the posterior distribution, with trained neural-network models (e.g., mixture density networks), to predict the means and covariance of the Gaussians [57, 62]. Though these methods performed sufficiently well for several inverse problems, it shows limited ability to represent more complex posterior distributions (multimodal ones especially). As a emerging approach, *normalizing flows* [58, 66] approximate complex distributions through sequential bijective transformation steps, starting with a simple normal distribution. Here, invertible neural networks (INNs) model the bi-directional conversion between the target distribution and a tractable one. This invertibility constitutes the unique advantage of normalizing flows over other variational inference models (e.g., a variational autoencoder [38]). A tractable distribution that can be converted to the target distribution in a bijective manner renders it possible to calculate that target's posterior density precisely and provides for easy sampling of the parameters from it. Both properties (tractable density and ease of sampling) are beneficial in the training and inference processes for density estimators. Therefore, recent research into amortized inference has welcomed normalizing flows [2, 21, 23, 40, 58]. BayesFlow [61] is a good example of recent work demonstrating the utility of normalizing flow models (RealNVP [14]) for amortized inference.

Non-HCI applications of amortized inference have exploited its flexibility; it is easy to apply to most simulation models yet does not necessitate critical assumptions [13, 22, 60], but the HCI field has been devoid of attempts to amortize inference for simulation models, with the exception of Murray-Smith *et al.*'s successful inference of

a finger's 3D pose from sensor data via a conditional variational autoencoder [51]. We fill this gap with the first presentation of a general workflow for applying amortized inference to the HCI field's simulation models.

3 WORKFLOW FOR AMORTIZED INFERENCE

For presenting the workflow of applying amortized inference to the inverse problem of user simulation models, we formulate the inverse problem thus: Let us suppose there are three components. 1) A *user simulation model* outputs simulated user behavior when given the state of an interaction task as input, with the model's parameters representing cognitive-physiological characteristics of a user; 2) *prior distributions* of the model parameters characterize prior knowledge of the plausible values referring to those user characteristics; 3) and *observation data* capture aspects of user behavior in the form of observational data collected from one or more users. To solve the inverse problem is to identify the posterior distribution of the model parameters via the given observation and arrive at the best estimated parameter values for observed users.

At the heart of amortized inference is training a conditional density estimator such that it can estimate the posterior density quickly with a given observation as input. Accordingly, the workflow can be summarized in terms of the following steps.

(1) Simulating user behaviors: Training of the conditional density estimator requires a large dataset composed of ground-truth pairs: parameters and corresponding user behaviors. With a user simulation model, we generate sufficient numbers of user behaviors with various parameter combinations sampled from the prior distributions.
(2) Building and training the density estimator: The density estimator is built with neural-network structures able to represent the parameters' complex posterior distribution from the given high-dimensional observation data. We train the density estimator to approximate the true posterior by using the simulated dataset from step 1.
(3) Inferring via the trained density estimator: Finally, we deploy the trained density estimator to infer the posterior of model parameters from actual user behaviors. Its inference process consumes very little computation time (e.g., tens of milliseconds).

While the first two steps are computationally expensive (requiring 1–2 days), these one-time actions enable rapid inference in step 3 for every new user's data.

3.1 Notation

We express the operation of a user simulation model as $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$, where $\boldsymbol{y}$ represents the model's output (i.e., user behavior), $\boldsymbol{x}$ represents the input (i.e., task state), and $\boldsymbol{\theta}$ represents the parameters (i.e., a user's cognitive-physiological characteristics). The prior of model parameters is denoted by $p(\boldsymbol{\theta})$, and their posterior when behavior data are given is $p(\boldsymbol{\theta}|\boldsymbol{y})$. We use the hat symbol ($\wedge$) to highlight variables for data *estimated* or *predicted* through models (e.g., $\hat{\boldsymbol{\theta}}$ for the estimated model parameters and $\hat{\boldsymbol{y}}$ for the predicted user behavior). Finally, the subscript "o" refers to data *observed* from real users, as in $\boldsymbol{y_o}$ for the observed user-behavior data.

3.2 Simulation of User Behavior

With user simulation models, we can generate a training dataset for the density estimator, consisting of plausible sets of $\boldsymbol{\theta}$ and simulated $\boldsymbol{y}$ under them, in the form of $(\boldsymbol{\theta}, \boldsymbol{y})$ pairs. Recent simulation models reproduce user behavior ($\boldsymbol{y}$) through the sequential decision-making of an RL agent: the agent obtains partial (or noisy) information about the task environment's state in keeping with an individual's cognitive-physiological bounds (dictated by mechanisms' limits [47, 67, 71] and variable parameters, $\boldsymbol{\theta}$) and chooses an action that achieves maximum utility. In RL terms, the simulated user's internal decision-making process is formulated as a partially observable Markov decision process (POMDP). The agent's decision-making function (i.e., control policy) can be implemented as a neural-network model that, from the observed state, ascertains the action, and deep RL techniques such as the deep Q-network (DQN) algorithm [15, 30, 48] afford optimizing it. One problem that most current simulation models face here is that their control policy is usually optimized within fixed bounds (i.e., with fixed $\boldsymbol{\theta}$) [10, 15, 30]. To collect the training data under various $\boldsymbol{\theta}$ values, one must adjust the control policy for each $\boldsymbol{\theta}$. Rather than take the time-consuming path of optimizing a control policy anew for each $\boldsymbol{\theta}$, this workflow follows recent multi-task RL approaches; i.e., a single generalized control policy is trained for a family of tasks (solving for a POMDP family parameterized by $\boldsymbol{\theta}$). Here, a neural-network-based policy model is implemented that receives $\boldsymbol{\theta}$ as external input alongside the observed task state. Modulated Q-network [49] and optimal control ensembles [41] exemplify such multi-task RL approaches' recent use. By optimizing the policy model subject to the multi-task environment (i.e., episodes with various $\boldsymbol{\theta}$), we are able to achieve a generalized optimal policy. In consequence, the training dataset can be simulated, where $\boldsymbol{\theta} \sim p(\boldsymbol{\theta})$, $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$.

3.3 Building and Training a Density Estimator

The conditional density estimator's objective is rapid estimation of the posterior ($p(\boldsymbol{\theta}|\boldsymbol{y})$) from given behavior data ($\boldsymbol{y}$). For the general case, it has to meet two crucial requirements. Firstly, it must deal with various data types present in user-behavior observations (e.g., a simple summary of multiple task trials or full details of each) and extract meaningful features from them as a fixed-size vector. Secondly, when given the extracted features, it should be capable of representing even highly complex distributions of $p(\boldsymbol{\theta}|\boldsymbol{y})$, including multiple modes (peaks). To address both requirements, we designed the density estimator to consist of an *encoder network* and a *conditional INN* (see Figure 2). The estimator learns bi-directional conversion between a latent variable (denoted as $\boldsymbol{z}$) from a normal distribution $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and $\boldsymbol{\theta}$ from a complex distribution $p(\boldsymbol{\theta}|\boldsymbol{y})$, when given $\boldsymbol{y}$. Therefore, $\boldsymbol{\theta} = g_{\boldsymbol{\phi}}(\boldsymbol{z}; \boldsymbol{y})$ and $\boldsymbol{z} = g_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\theta}; \boldsymbol{y})$, where $g_{\boldsymbol{\phi}}$ represents the forward conversion through the density estimator with neural-network weights $\boldsymbol{\phi}$. Because the latent variable ($\boldsymbol{z}$) follows a normal distribution, it is easy to sample $\boldsymbol{z}$ and calculate its exact density. Therefore, with the bijective transformation between $\boldsymbol{z}$ and $\boldsymbol{\theta}$, we can achieve easy sampling and density estimation for $p(\boldsymbol{\theta}|\boldsymbol{y})$ as well.

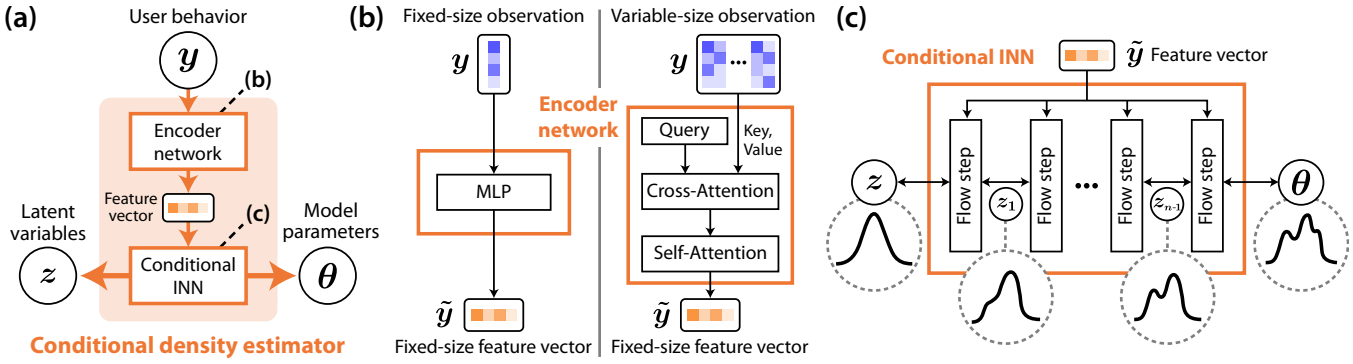

Figure 2: (a) A simplified representation of our conditional density estimator for amortized inference. (b) An encoder network taking user-behavior data y as input and extracting a fixed-size feature vector $\tilde{y}$. The network's design can be varied to suit the type of behavior data, such as summary features of multiple trials (fixed-size) or all information from the trials (variable-size). (c) With the given $\tilde{y}$, a conditional INN providing bi-directional conversion between latent variables z and model parameters θ, by following multiple flow steps. Each step ensures perfect invertibility between two distributions, and stacking the flow steps makes it possible to produce a complex posterior distribution $p(\theta|y)$ from a simple normal distribution $p(z)$.

3.3.1 The encoder network. An encoder network (Figure 2(b)) outputs a fixed-size feature vector (denoted by $\tilde{\boldsymbol{y}}$) from a given observation ($\boldsymbol{y}$). The design of this network can vary, depending on the type of $\boldsymbol{y}$. We consider the most basic $\boldsymbol{y}$ first: simple summary statistics for a user's multiple task trials (e.g., average completion time or task scores). In this case, the observation $\boldsymbol{y}$ from each user has the same size, so the encoder network can be constructed with a multi-layer perceptron, or MLP (that is, multiple linear layers coupled with nonlinear activation). When, on the other hand, $\boldsymbol{y}$ is the full set of observation data from multiple *i.i.d.* task trials (e.g., all the completion times and task scores for each trial), the size for observation set ($\boldsymbol{y}$) can be varied by user on the basis of the number of trials completed. To extract a fixed-size output from this variable-size input in a permutation-invariant manner (i.e., without restrictions imposed by trial order), we utilize recently developed query–key–value (QKV) attention structures [29, 76]. Supplement A.1 provides further details.

3.3.2 The conditional invertible neural network. The conditional INN (Figure 2(c)) targets approximating $p(\boldsymbol{\theta}|\boldsymbol{y})$ in accordance with the feature vector of the given behavior data ($\tilde{\boldsymbol{y}}$). The INN models bijective transformation between $p(\boldsymbol{\theta}|\boldsymbol{y})$ and $p(\boldsymbol{z})$ (=$\mathcal{N}(\mathbf{0}, \boldsymbol{I})$) when given $\tilde{\boldsymbol{y}}$. To this end, it is designed with several *flow* steps [58, 66], each of which is an invertible transformation between inputs and outputs, especially conditioned on $\tilde{\boldsymbol{y}}$. As flow steps stack more deeply, it is possible to express more complex distributions, starting from a normal distribution (observe how $p(\boldsymbol{z})$ transforms to $p(z_1) \rightarrow \cdots \rightarrow p(z_{n-1}) \rightarrow p(z_n) = p(\boldsymbol{\theta}|\boldsymbol{y})$ over n flow steps in Figure 2(c)). We implemented each flow step with a recent normalizing flow model, *Glow* [37] (see Supplement A.2 for details).

3.3.3 The training process. The encoder network and the conditionally reversible network are trained once, from the data generated by the user simulation models[2] (see Figure 3(a)). The training's objective is to minimize the Kullback–Leibler divergence (KLD) between true posterior $p(\boldsymbol{\theta}|\boldsymbol{y})$ and our estimated posterior $\hat{p}_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\boldsymbol{y})$; hence, our goal is to find $\boldsymbol{\phi}^*$, the optimal neural-network weights for the density estimator that approximates the true posterior as closely as possible. That is,

$$\boldsymbol{\phi}^* = \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\, \mathbb{E}_{p(\boldsymbol{\theta})}\left[\mathbb{KLD}(p(\boldsymbol{\theta}|\boldsymbol{y})||\hat{p}_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\boldsymbol{y}))\right].$$

Here, the objective function can be solved via stochastic gradient descent methods with the loss function $\mathcal{L}(\boldsymbol{\phi})$ below (the literature provides details of the derivation and its validated foundation [61]). If we have a batch of M pairs of true $\boldsymbol{\theta}$ and $\boldsymbol{y}$ from the simulated training dataset (per Section 3.2), then

$$\mathcal{L}(\boldsymbol{\phi}) = \frac{1}{M}\sum_{i=1}^{M}\left(\frac{\|g_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\theta}^{(i)};\boldsymbol{y}^{(i)})\|_2^2}{2} - \log\left|\det\left(J_{g_{\boldsymbol{\phi}}}^{(i)}\right)\right|\right), \quad (1)$$

where $\det\left(J_{g_{\boldsymbol{\phi}}}^{(i)}\right)$ denotes the determinant of the Jacobian matrix $\partial g_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\theta};\boldsymbol{y})/\partial\boldsymbol{\theta}$ at $(\boldsymbol{\theta}^{(i)}, \boldsymbol{y}^{(i)})$. The function's first term refers to a negative log of $\mathcal{N}(\boldsymbol{z}|\mathbf{0}, \boldsymbol{I})$; therefore, minimizing $\mathcal{L}(\boldsymbol{\phi})$ causes $\boldsymbol{z}$ to follow $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$.

3.4 Inference with a Trained Density Estimator

With the trained density estimator, we can rapidly infer the posterior of model parameters ($\boldsymbol{\theta}$) from real users' observed behavior ($\boldsymbol{y_o}$), as illustrated in Figure 3(b). Sampling the latent variable $\boldsymbol{z}$ from $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and computing $\boldsymbol{\theta}$ via $g_{\boldsymbol{\phi}}(\boldsymbol{z};\boldsymbol{y_o})$ is equivalent to sampling $\boldsymbol{\theta}$ from its true posterior $p(\boldsymbol{\theta}|\boldsymbol{y_o})$ if one assumes perfect convergence of density-estimator training [61]. Therefore, obtaining a plausible set of $\boldsymbol{\theta}$ from the given $\boldsymbol{y_o}$ comes at very low computational cost, only that of passing the sampled $\boldsymbol{z}$ and $\boldsymbol{y_o}$ through the trained density-estimator network, which can be performed on the order of milliseconds. With the set of plausible $\boldsymbol{\theta}$ candidates, we can approximate the full posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{y_o})$ by means of well-known methods, such as kernel density estimation (KDE), and

[2] Although this paper addresses the simulation of training data and the training of the density estimator separately (in Section 3.2 and Section 3.3, respectively), those simulation and training processes may be combined; for instance, we can simulate M pairs of $(\boldsymbol{\theta}, \boldsymbol{y})$ for every gradient descent step during the training.

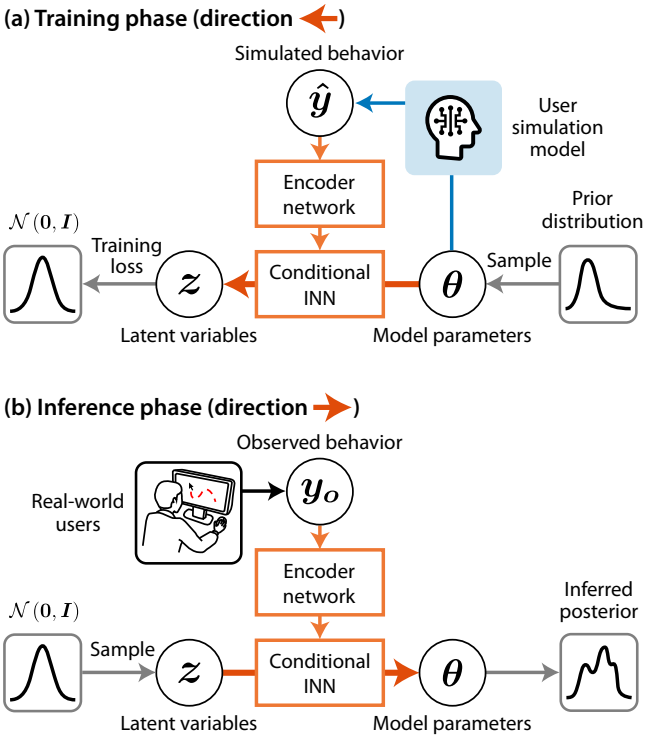

Figure 3: A diagram outlining our amortized inference procedure with density estimator. (a) In the training phase, θ sampled from the prior distribution $p(\theta)$ and y simulated on its basis are fed to the density estimator for obtaining z as output (i.e., $\theta \rightarrow z$ with given y). The density estimator is trained to make the output z follow a unit normal distribution. (b) With the trained density estimator, we can infer model parameters from the observations: behavior data from real users. The observed behavior y_o and z sampled from a unit normal distribution are given to the trained density estimator, which outputs θ (i.e., $z \rightarrow \theta$ with given y_o). Using a batch of the output θ candidates, we can estimate the posterior distribution $p(\theta|y)$.

determine the estimated values of model parameters $\hat{\theta}$, typically using maximum *a posteriori* (MAP) estimation [34].

3.5 Use of a Point Estimator

Should distribution information be unnecessary, one can simplify the estimator model's structure by seeking only point-estimate values of the parameters. This can yield the benefit of decreased size of the neural-network model, thereby reducing the computation costs for training and inference. The workflow presented above supports training a *point estimator* accordingly. Instead of a conditional INN in a density estimator, one can build a point estimator with any feedforward network (e.g., an MLP) that directly outputs a point estimate from a feature vector extracted from the encoder network. With the same training dataset, the process would use a general regression loss (e.g., mean squared error) rather than the training loss of Equation 1. The inference process would consist only of feeding forward the given observation data, without any need for sampling of the latent variable z. Further on (in Section 8.3), we report the quantitative results from testing how using point estimators as opposed to density estimators affects inference performance.

4 OVERVIEW OF CASE STUDIES

The three case studies demonstrate the suitability of the workflow for amortized inference in connection with fitting a range of user simulation models to user-behavior datasets. Our reporting on them below addresses inverse modeling problems that involve progressively more complex observed behaviors (from simple summary features to full information of multiple i.i.d. trials) and larger user datasets (from N of 18 to 1,057). Figure 4 presents the nature of the observed behaviors and inferred model parameters in each case. The case studies are characterized thus:

- Case 1: fitting the menu-search model [10] to a user-behavior dataset where N=18 [3], a setting with relatively simple behavior observations consisting of only summaries of the features of users' task trials overall (e.g., average completion time)
- Case 2: fitting the point-and-click model [15] to a dataset with N=20 [49] that covers more complex observations, encompassing full behavior data (e.g., cursor trajectories) from hundreds of i.i.d. task trials per user
- Case 3: fitting the touchscreen-typing model [30] to a much larger dataset [56], extending to more than a thousand individual users (N=1,057)

To fit the models, we trained the density estimator solely on the basis of synthetic data generated by user simulation models. The learned estimator then was assessed against the empirical dataset in each case. In Case 1, we used the empirical dataset against which the authors of menu-search model assessed their model's (forward) predictions. By contrast, in Case 2 and 3, we chose the datasets that differ from what the authors originally used for their models' assessment, showing that inverse modeling with our density estimator can be well-generalized to various datasets. For Case 2, we assessed the density estimator against a dataset that was collected after the original point-and-click model's release. Our work with Case 3 involved a large-scale empirical dataset not previously used for inverting the touchscreen-typing model.

5 CASE 1: MENU SEARCH

The menu-search model [10] simulates a user's behavior in searching for a target item in a linear menu interface. The goal of the simulated user is to either select the given target item (if it is present in the menu) or declare that the target item does not exist (if it is not). The simulated user can discover each item's information by looking directly at it, by looking at an item above or below it (with peripheral vision), or through memory recall. Via sequential decision-making directed by an optimized control policy, the model replicates a user's eye movements during menu search and, thereby, predicts the completion time of each trial (see prior publications for further details [10, 34]). For our study, we decided to infer four of the menu-search model parameters, following the lines of earlier work [34] that used ABC. These inferred parameters $\boldsymbol{\theta}$ are

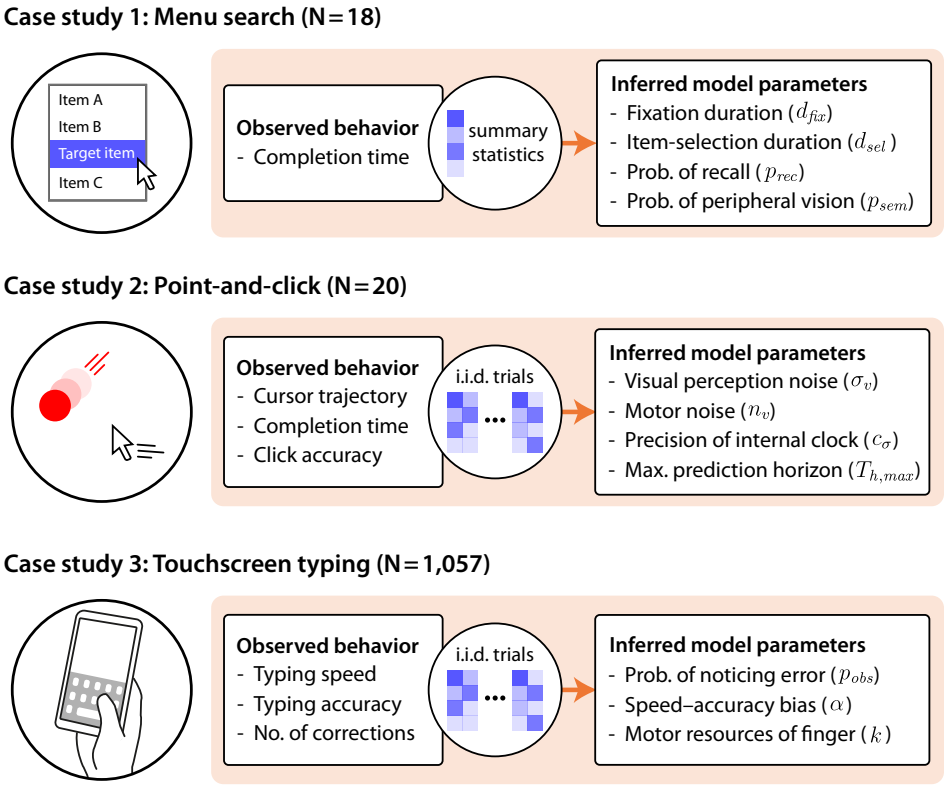

Figure 4: An overview of the three case studies. Each case entails inferring the parameters for a particular user simulation model (menu search [10], point-and-click [15], and touchscreen typing [30]) from the observed behavior of individual users.

- d_{fix}: the duration for eye fixation
- d_{sel}: the duration for item selection
- p_{rec}: probability of recalling all items' information during the first fixation period
- p_{sem}: probability of investigating the item above or below the fixation target via peripheral vision

For the inverse modeling, we used a dataset of menu-search behavior [3] from 21 users (13 of them female; mean age 21.1, σ=3.54). It comprised the task-completion time and eye-movement data for each menu-search trial. From the full dataset, our study considered included only material from menu layouts that the simulation model covers (i.e., with eight items grouped by semantic similarity); hence, we employed a dataset of 18 participants' behavior (150 trials per user, on average). We used the same observation data $\boldsymbol{y}$ as in [34] for model fitting: a 1-D vector containing the task-completion times' mean and standard deviation values for each case, with the target present or absent. This represents one of the simplest and most common settings of inverse modeling problems in HCI, with user behavior visible only from aggregate data.

5.1 Implementation Details

With the workflow described in Section 3, we applied amortized inference as presented below.

5.1.1 Model simulation. We generated a dataset of simulated behavior $\boldsymbol{y}$ under the various values of $\boldsymbol{\theta}$. While the original model used a control policy optimized for one fixed $\boldsymbol{\theta}$, we generalized the policy to ensure optimal decision-making with each value of $\boldsymbol{\theta}$, implementing the control policy via a modulated Q-network [49], which can behave optimally for the various sets of $\boldsymbol{\theta}$ after training. Supplement B.1 elaborates on both the network and its training.

During the model simulation, $\boldsymbol{\theta}$ is sampled from given prior distributions $p(\boldsymbol{\theta})$ for each trial. We chose the same $p(\boldsymbol{\theta})$ as the earlier work [34], thus: the prior for d_{fix} is a truncated normal distribution where (mean, std, min, max) = (300 ms, 100 ms, 0 ms, 600 ms); the prior for d_{sel} is a truncated normal where (mean, std, min, max) = (300 ms, 300 ms, 0 ms, 1000 ms); that for p_{rec} is uniform where (min, max) = (0.0, 1.0); and p_{sem}'s prior is a uniform one where (min, max) = (0.0, 1.0). We sampled 262K distinct $\boldsymbol{\theta}$ values from $p(\boldsymbol{\theta})$ and simulated 256 trials for each sampled $\boldsymbol{\theta}$. In total, the training dataset consists of 67M simulated trials. The entire simulation took 1.5 hours (in wall-clock time) with 16 CPU cores (AMD Ryzen 9 5950X, 3.5 GHz).

5.1.2 Density-estimator training. Then, we built and trained the conditional density estimator by using the simulation dataset formed. In this menu-search scenario, $\boldsymbol{y}$ is a simple one-dimensional fixed-size vector, so we opted for a simple MLP as the encoder network to extract $\tilde{\boldsymbol{y}}$ from given $\boldsymbol{y}$. We implemented the conditional INN to consist of five Glow steps, for all our cases (1–3). A batch of ($\boldsymbol{\theta}$, $\boldsymbol{y}$),

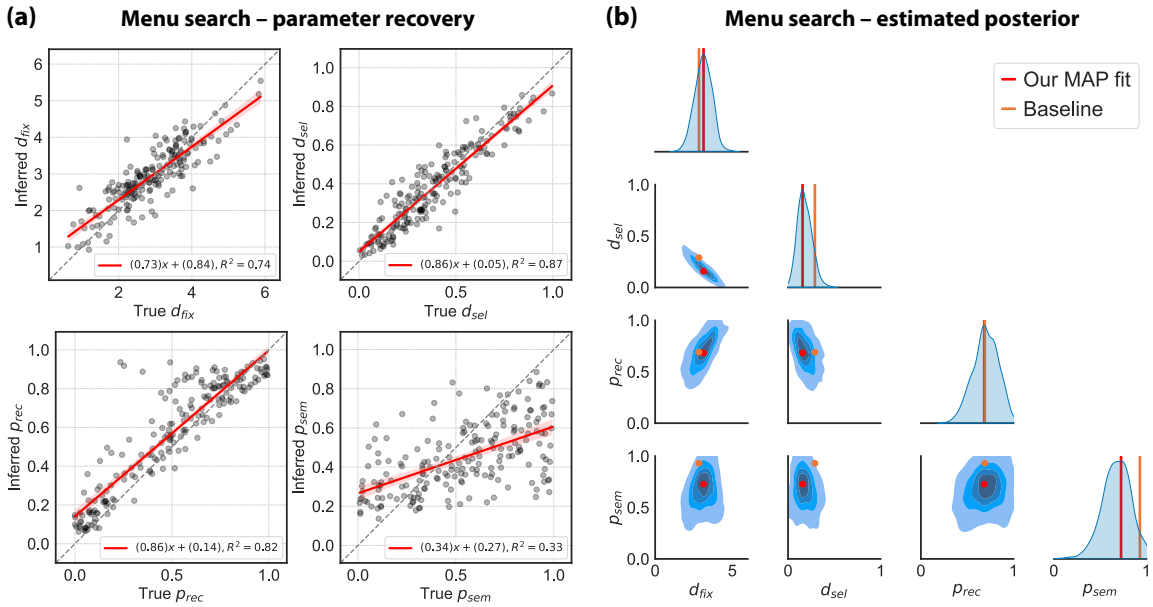

Figure 5: Amortized inference results with the menu-search model (Case 1): (a) parameter recovery (x-axis: ground-truth model parameter values, y-axis: inferred model parameter values) from 100 parameter samples; (b) estimated posterior distributions from the user-behavior dataset. Baseline represents the model parameter values fitted via ABC [34].

Table 1: Behavior-prediction accuracy with the menu-search model (Case 1), characterized via distances between the observation and predictions (baseline, group-level, and individual-level fit with amortized inference). Baseline represents prediction using the model fitted via ABC [34]. Amortized inference leads to more accurate predictions for empirical data by allowing fitting the user model to the level of an individual. For each row, the closest results are presented in green and the second-closest ones are shown in light green.

Behavior	Distance metric	Baseline [34]	Amortized inference	
			Group-level fit	Individual-level fit
Completion time (with target)	Mean diff.	76.691	65.810	43.597
	KLD	0.0741	0.0454	0.0108
	MMD	3.4193	2.8592	0.4787
No. of fixations (with target)	Mean diff.	0.0635	0.1399	0.0018
	KLD	0.8681	0.3735	0.7285
	MMD	0.3850	0.2098	0.3395
Completion time (no target)	Mean diff.	65.972	267.240	186.825
	KLD	0.1007	0.0901	0.0524
	MMD	6.8355	5.9524	0.9976
No. of fixations (no target)	Mean diff.	0.3397	0.1662	0.0798
	KLD	3.9804	3.8756	1.8807
	MMD	0.3603	0.5503	0.1288

with size 512, was sampled from the dataset and used to compute the loss (Eq. 1) for gradient descent updating. We performed 600K training steps (i.e., gradient descent updates), with a total training time of approximately 10 hours on a standard workstation with the aforementioned CPU and an NVIDIA GeForce RTX 3080 GPU. Supplement B.2 gives further details of the density-estimator model implemented and its training.

5.1.3 Inference. By sampling 1,000 latent variables $\boldsymbol{z}$ from a unit normal distribution and taking $\boldsymbol{y_o}$ as input to the density estimator, we sampled 1,000 $\boldsymbol{\theta}$ candidates. We estimated $p(\boldsymbol{\theta}|\boldsymbol{y_o})$ by applying KDE with the sampled candidates for visualization, and we determined our $\hat{\boldsymbol{\theta}}$ as the MAP value of $p(\boldsymbol{\theta}|\boldsymbol{y_o})$. The processes for cases 1–3 all applied the same inference procedure.

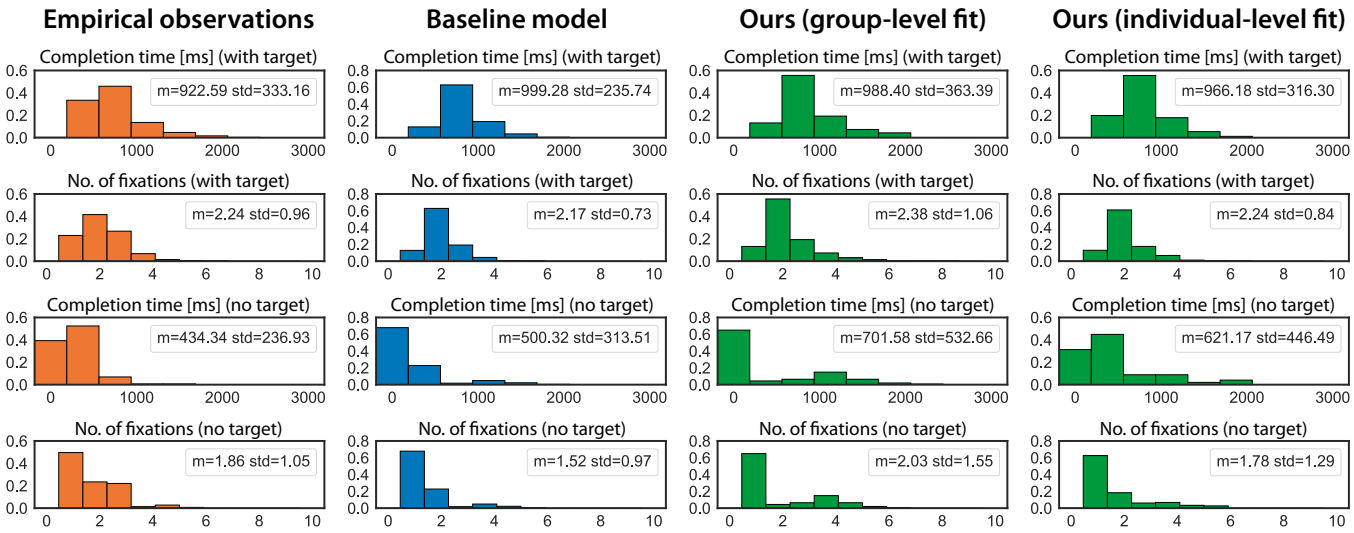

Figure 6: Comparison of empirical observations and behavior predictions in the case of the menu-search model (Case 1).

5.2 Evaluations

We evaluated the performance of amortized inference from the following two perspectives: how well the conditional density estimator can recover ground-truth model parameters from the simulated data (*parameter recovery*) and how well the parameter values inferred from the empirical user data can actually predict the user's behavior (*behavior prediction*).

To judge the former, we randomly sampled 100 sets of $\boldsymbol{\theta}$ from $p(\boldsymbol{\theta})$ and simulated $\boldsymbol{y}$ from them. Then, we measured the coefficient of determination (R^2) between the ground-truth $\boldsymbol{\theta}$ and the density estimator's prediction ($\hat{\boldsymbol{\theta}}$). Here, R^2 represents the proportion of the variation in the predicted $\hat{\boldsymbol{\theta}}$ that can be explained by variation in the true $\boldsymbol{\theta}$. Hence, higher values are better; the range is 0 to 1, so $R^2 = 1$ denotes a perfect prediction for $\boldsymbol{\theta}$.

Next, we assessed the model's behavior-prediction performance with real users' data when using the fitted parameters with amortized inference. We investigated the two scenarios for applying amortized inference: group-level and individual-level fitting. For group level, one model parameter set was estimated from the all-users dataset. In the individual-level fitting, on the other hand, model parameter sets were estimated for each individual user's behavior data. We measured the prediction accuracy as the distance between the simulated behavior under the estimated parameters ($\hat{\boldsymbol{y}}'$) and the actual behavior of the same users ($\boldsymbol{y}'_{\boldsymbol{o}}$).[3] We considered three distinct distance metrics: 1) mean difference, 2) KLD,[4] and 3) maximum mean discrepancy (MMD) [24]. Mean difference captures the absolute difference between the mean values from the behavior features in the trials in each case (ground-truth observation vs. model prediction). In contrast, KLD and MMD estimate the difference between distributions. The closer KLD and MMD are to 0, the closer the behavior-feature distributions of $\hat{\boldsymbol{y}}'$ and $\boldsymbol{y}'_{\boldsymbol{o}}$ are.

We measured the distance with regard to four features of menu-search behavior: each trial's completion time and number of fixations, when the target is present and when it is absent (2×2). We compared the resulting fit (group- and individual-level) from amortized inference with the *baseline*: use of the earlier model parameters fitted via ABC [34]. This baseline represents applying conventional non-amortized inference to fit the parameters at group level.

5.3 Results

The inference process, run a single time for each dataset, took about 10 ms with our trained density estimator (i.e., both group- and individual-level fitting). Accordingly, individual-level inference for 18 users consumed only about 200 ms in total. This is a significant improvement over previous (ABC) methods, which took 37 h for group-level fitting for the same user dataset [34].

Figure 5(a) depicts parameter-recovery performance with our trained density estimator, showing the correlation between the ground-truth values and our predicted values for each model parameter. The results show that the trained estimator predicts most parameters with a high coefficient of determination (R^2=0.74 for d_{fix}, 0.87 for d_{sel}, and 0.82 for p_{rec}). It was more difficult to estimate p_{sem} from the given behaviors (R^2=0.33).

Figure 5(b) shows the estimated posterior distribution from the actual behavior measured from all users. Our estimated posterior distribution closely matches the baseline model parameters, which lie within its high-probability region. Table 1 shows the distance between the observed behavior and the three model predictions. While the baseline, group-level, and individual-level model fits with amortized inference all replicate the observations reasonably well (see Figure 6 for the full histograms of observed and simulated behaviors over trials), from among the three it was our individual-level

[3] We split the user-behavior data used for model parameter estimation from the material for distance measurement; that is, $\boldsymbol{y}'_{\boldsymbol{o}}$ comes from those trials not used for parameter estimation.

[4] In strict terms, this is not a distance metric, since it does not satisfy the two distributions' symmetry condition. However, for simplicity, the paper presents KLD as a distance metric alongside other metrics.

fitting that made the most accurate prediction, with the smallest distance by nine out of the 12 metrics (4 behavior features × 3 distance metrics). On the other hand, there was no noticeable difference between the baseline and our group-level fit, which showed the smallest distance by one and two metrics, respectively; that is, the amortizing approach offered inference performance comparable to ABC's (in the same group-fitting cases) while vastly improving the computation cost (from 37 h to 10 ms).

6 CASE 2: POINT-AND-CLICK

Case 2's point-and-click model [15] simulates the user behavior of selecting a distant target on a computer screen by means of a mouse device. In each trial, one circular target with a random size appears at a random location onscreen and moves at a random but constant speed in a random direction. The goal of a simulated user is to click the mouse button with the cursor positioned within the target. The agent visually perceives the target's and cursor's information at every timestep. Proceeding from the perceived information, the simulated user iteratively updates its motor plan and decides, per decision-making by its control policy, whether to click during the motor plan. The original publication presents the simulation model more fully. In this study, we inferred four parameters of the model:

- σ_v: visual perception noise
- n_v: signal-dependent motor noise
- c_σ: precision of one's internal clock
- $T_{h,max}$: maximum length of the motor plan a user can build (i.e., prediction horizon)

In a previous study [49], using ABC, the authors attempted to infer the first three parameters, each of which represents a user capability – for visual perception, motor movement, and click action, respectively. The final parameter ($T_{h,max}$) can represent a user's motor-planning capability in addition.

For the inverse modeling, we used their dataset from the point-and-click behavior of 20 users (13 of them female; mean age 30.4, σ=8.65) [49]. The following data were collected for each trial: the user's task performance (success/failure and completion time), the trial's initial state (the radius, position, and velocity of the target and the cursor's position and velocity), and the trajectories of the target and cursor (positions every 50 ms). We used $\boldsymbol{y}$ encompassing all of the aforementioned data from 600 trials. This represents a more complex class of inverse modeling problem in HCI than Case 1, in that it involves complex user behaviors, of multiple types (with different semantics and data types), in each of the i.i.d. trials. In addition, the empirical data include values, measured participant-specifically through separate experiments, for the three cognitive-physiological capabilities (σ_v, n_v, and c_σ) The previous study showed that inferred parameter values for the point-and-click model could describe the measured abilities.

6.1 Implementation Details

Most aspects of the implementation for amortized inference were the same as in Case 1. We implemented the control policy of the point-and-click model with a modulated Q-network, using the prior work's network structure and optimization method [49]. This generalization of the control policy enables single-model simulation of point-and-click behavior $\boldsymbol{y}$ under the various values of $\boldsymbol{\theta}$. Supplement C.1 details the control-policy structure and optimization.

The simulation's prior distributions $p(\boldsymbol{\theta})$ too were determined from the earlier work [49]: the prior for σ_v is a log-uniform distribution where (min, max) = (0.069, 0.415); that for n_v is log-uniform with (min, max) = (0.145, 0.413); and that for c_σ is a log-uniform one where (min, max) = (0.055, 0.400). The previous report did not cite any prior knowledge of the range for $T_{h,max}$, so we used a simple uniform prior for it where (min, max) = (0.5 s, 2.5 s). We sampled 262K distinct $\boldsymbol{\theta}$ values from $p(\boldsymbol{\theta})$ and simulated 32 trials for each sampled $\boldsymbol{\theta}$. In total, 8M simulated point-and-click trials were generated for density-estimator training. With 16 CPU cores, the simulation process took seven hours.

In contrast against Case 1's $\boldsymbol{y}$, consisting of only summary features across trials, Case 2's consists of multiple trials' data. The number of observed trials can be varied across users (i.e., $\boldsymbol{y}$ size can be varied). Accordingly, we implemented the attention-based encoder network to extract fixed-size features ($\tilde{\boldsymbol{y}}$) from variable-size observations ($\boldsymbol{y}$). One million gradient descent steps (with batch size 32) were performed with the simulated training dataset. The total training time, approximately 30 h, was longer than Case 1's because of the more complex observation data and network structure involved. Supplement C.2 elaborates on the density-estimator model implementation and training.

6.2 Evaluations

We evaluated the performance of our amortized inference from the same angles considered in Case 1: parameter recovery and behavior prediction (detailed in Section 5.2). With regard to the first, we assessed how well the trained density estimator could infer 100 sets of ground-truth parameter values from the simulated behaviors. For Case 2, the investigation additionally considered the estimator's ability to recover real users' measured cognitive-physiological capabilities from each user's point-and-click behavior.

In this case too, we assessed behavior-prediction performance with regard to baseline, group-level, and individual-level fit. We measured the distance between the point-and-click behavior simulated via the fitted model parameters ($\hat{\boldsymbol{y}}'$) and actual user behavior ($\boldsymbol{y}'_o$) in terms of three features: 1) trial-completion time; 2) the cursor's final click location relative to the target, with the target's radius as the unit (hence, a value below 1 indicates that the click fell within the target); and 3) the cursor's total travel distance in each trial. The baseline case for comparison came from simulated behavior where the participant-specific cognitive-physiological capability values [49] furnished the model's parameters; therefore, the baseline here represents obtaining each individual user's unique characteristics to inform the simulation, through laborious measurement processes.

6.3 Results

Since each dataset's inference process took 125 ms with our trained density estimator, individual-level inference for 20 participants' datasets used just 2.5 s. In contrast, the previous study reported 30 h for the same individual-level inference (using ABC alongside a policy-modulation technique for rapid model simulation).

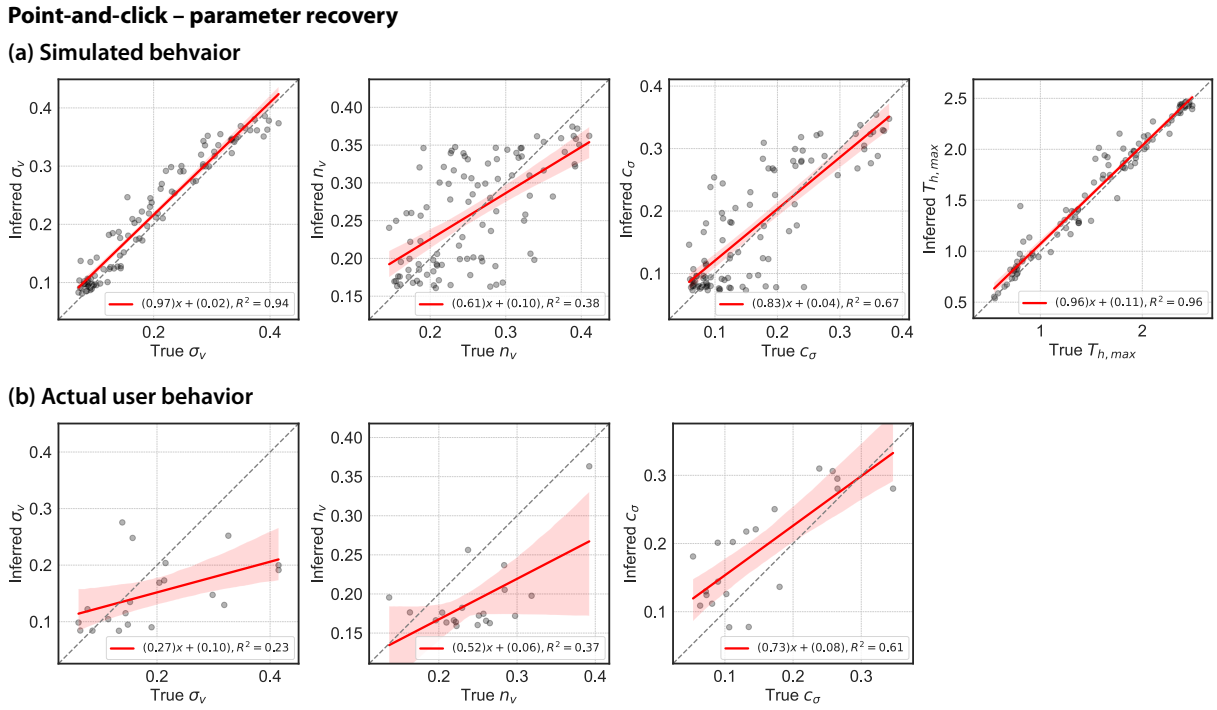

Figure 7: Parameter-recovery results with the point-and-click model (Case 2): the results with (a) simulated behavior data (x-axis: ground-truth-model parameter values, y-axis: inferred values) and (b) the actual user-behavior data (x-axis: each user's measured capability values, y-axis: inferred values). Note that, since the earlier study [49] did not measure $T_{h,max}$ for each participant, parameter recovery for $T_{h,max}$ was not examined with actual user behavior.

Table 2: Behavior-prediction accuracy under the point-and-click model (Case 2), represented via distance between the actual observations and the model's predictions. Baseline represents prediction of the model with participant-specific values of the measured capabilities [49]. For each row, the closest results are presented in green and the second-closest are in light green.

Behavior	Distance metric	Baseline [49]	Amortized inference	
			Group-level fit	Individual-level fit
Completion time	Mean diff.	0.0110	0.0751	0.0433
	KLD	6.5330	8.7802	5.3777
	MMD	0.0868	0.2040	0.1041
Click endpoint (normalized)	Mean diff.	0.5189	0.1384	0.0021
	KLD	0.3568	0.3077	0.2012
	MMD	0.0334	0.0059	0.0072
Cursor travel distance	Mean diff.	0.0339	0.0288	0.0302
	KLD	16.874	14.173	13.608
	MMD	0.0302	0.0261	0.0252

Figure 7 presents the parameter-recovery performance with the density estimator, for both simulated behavior data and actual users' behavior data. With the former, the density estimator predicts the model parameters with high (R^2=0.94 for σ_v, 0.67 for c_σ, and 0.96 for $T_{h,max}$) and moderate (R^2=0.38 for n_v) coefficients of determination. From the actual user behavior, it was possible to predict their cognitive-physiological capability values moderately well (R^2 = 0.37 for n_v and 0.61 for c_σ) and at low levels (R^2=0.23 for σ_v). This inference performance was comparable with that of the ABC-based prior technique [49], which estimated the same participants' capability values σ_v and c_σ moderately well (with R^2 values of 0.50 and 0.62, respectively) but failed to infer n_v. The R^2 values for the inferred σ_v are slightly lower in our amortized inference case, but we

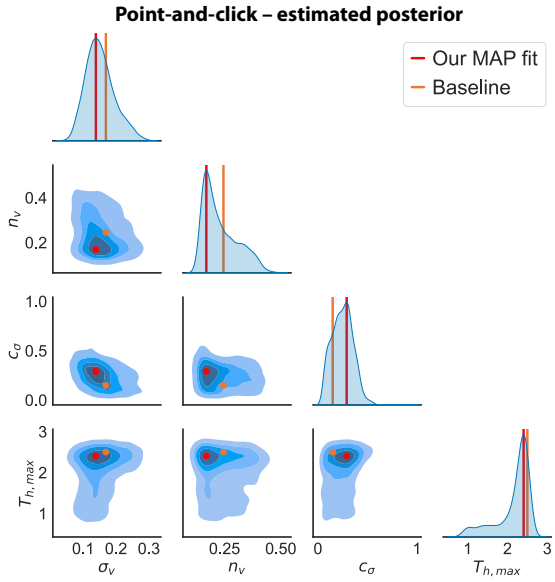

Figure 8: Estimated posterior distributions from the empirical dataset with amortized inference in Case 2. Baseline represents the averaged values for the users' measured capabilities [49].

succeeded in inferring n_v and achieved a huge speed improvement via amortized inference (to 2.5 s from 30 h).

Figure 8 plots the estimated posterior distributions of each model parameter from using our density estimator on the aggregated behavior data of all participants. The averaged values of the participants' measured capabilities (σ_v, n_v, and c_σ) lie well within the high-probability region of the estimated posterior distributions. Table 2 shows the distance between the actual observation and the three model predictions: baseline, group-level, and individual-level fits (Figure 9 presents the full histogram for each simulated behavior). From among the three models, the individual-level fit (with amortized inference) showed the results closest to actual user behavior by the majority of metrics (5 out of the 9). That is, the individual-level model fit outperformed the baseline method, in which the parameters were determined in line with each individual user's measured capabilities.

7 CASE 3: TOUCHSCREEN TYPING

The touchscreen-typing model examined, by Jokinen *et al.* [30], simulates user behavior of typing on a cell phone or other touchscreen device with a finger. Here, the simulated user's goal is to reach optimal typing performance by allocating two limited resources: finger movement and visual attention. Sharing these resources, the simulated user can either type a character in the target sentence or proofread the text produced so far (as dictated by the control policy's decisions). If proofreading reveals an error, the model simulates the finger movement to correct the typed text (i.e., backspace over it). The finger-movement simulation balances speed against accuracy (faster finger movements are noisier) by applying a weighted homographic (WHo) model [25]. Thus, the model can simulate aggregate typing performance (e.g., typing speed and accuracy), error-correction processes during a trial, and finger and eye movements over time. Further details can be found in [30]. Our study inferred the following three parameters:

- p_{obs}: probability of noticing an error directly from finger movement (without visual attention to the typed text)
- α: finger movements' speed–accuracy bias (for WHo model)
- k: all motor resources for finger movement (for WHo model)

In the original publication, the authors adopted values of α (0.6) and k (0.12) from prior literature [69], and the p_{obs} value was hand-tuned (to 0.7).

Our inverse modeling employed a dataset much larger than those in cases 1–2. It comprises 37,000 users' touchscreen-typing behaviors with mobile devices [56]. We filtered this for only data from English-speaking users who typed with one finger used a QWERTY layout, and we also excluded trials wherein participants employed intelligent text-entry techniques. In consequence, the material for parameter fitting in Case 3 consisted of 14,277 trials' data, from 1,057 users (665 of them female; mean age 28.10, σ=10.67), with 13.5 trials per user, on average. As in Case 2, $\boldsymbol{y}$ consisted of the behavior data from all the i.i.d. trials. From each typing trial, we used the following data: 1) words per minute (WPM), obtained by dividing typed-text word count by trial-completion time; 2) error rate, calculated by dividing the Levenshtein distance [43] between the given and typed text by the longer one's length; 3) the number of backspace presses per trial; 4) keystrokes per character (KSPC), consisting of the number of press inputs divided by that of characters typed; 5) and the length of the given text piece.

7.1 Implementation Details

The control policy of the original touchscreen typing model, composed of two distinct network models (actor and critic networks), was obtained by proximal policy optimization (PPO) [70]. We generalized this control policy to operate under a range of model parameters, in the same manner as for the modulated Q-network used in Case 1 and 2. We implemented and trained both networks to take given model parameters ($\boldsymbol{\theta}$) in addition to the observed task state as inputs. Such a method of conditioning for actor-critic networks on the basis of given parameters has been verified [41] under the name "optimal control ensembles." Supplement D.1 gives further details.

The prior distributions of the model parameters were set such that the one for p_{obs} is a uniform distribution where (min, max) = (0, 1); the prior for α is a truncated normal where (mean, std, min, max) = (0.6, 0.3, 0.4, 0.9); and that for k is a truncated normal where (mean, std, min, max) = (0.12, 0.08, 0.04, 0.20). We sampled 66K distinct $\boldsymbol{\theta}$ values from $p(\boldsymbol{\theta})$ and simulated 16 trials for each one sampled. Therefore, 1M simulated typing trials were obtained for density-estimator training. The simulation process took 14 hours for 16 CPU cores.

Since $\boldsymbol{y}$ consists of multiple trials' data (as in Case 2), we implemented our attention-based encoder network to extract fixed-size features from the variable-size observations; for the training, we used 600K gradient descent steps (batch size = 100), and training took 15 h in all. Supplement D.2 provides details.

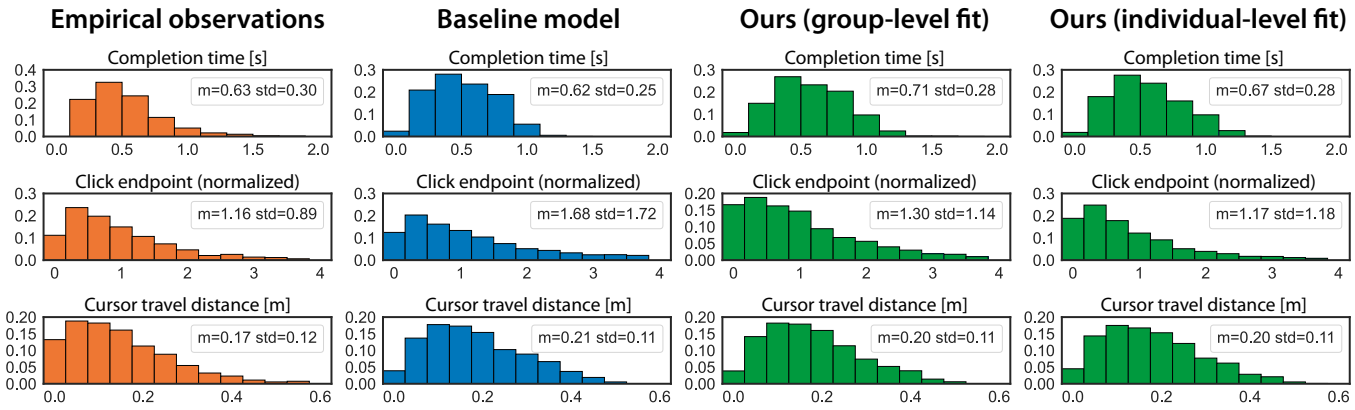

Figure 9: Comparison of empirical observations and behavior predictions with the point-and-click model (Case 2).

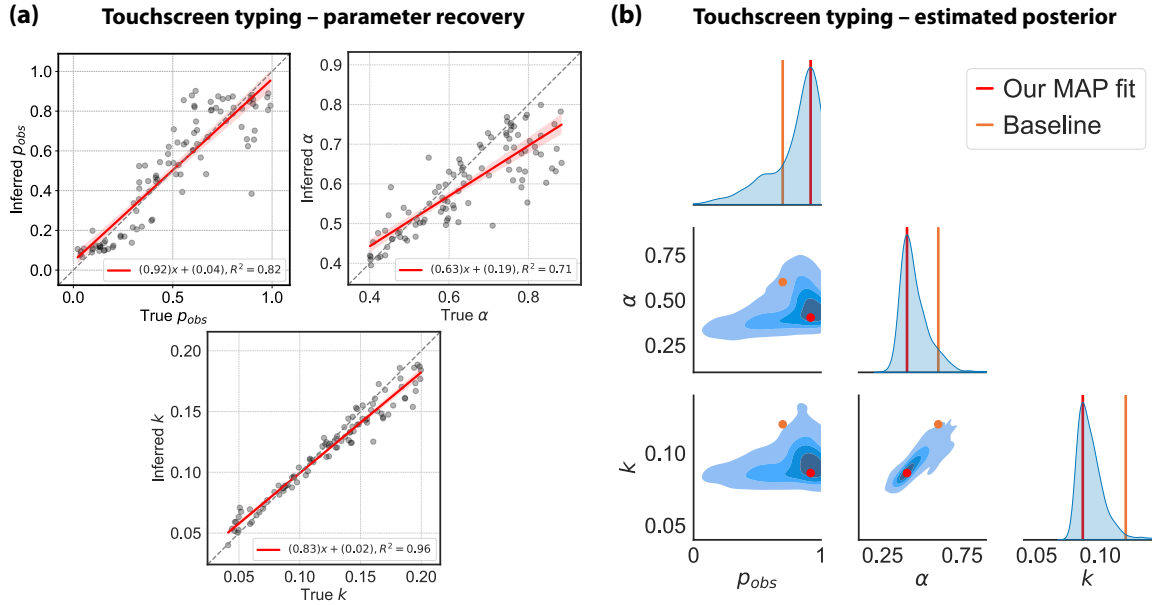

Figure 10: Amortized inference results with the touchscreen-typing model (Case 3): (a) parameter recovery (x-axis: ground-truth parameter values, y-axis: inferred values) from 100 parameter samples; (b) estimated posterior distributions from the user-behavior dataset. Baseline represents the parameter values from the original study [30] which were imported from literature.

7.2 Evaluations

As with the other cases, we assessed parameter recovery (the accuracy of inferring 100 sets of ground-truth model parameters from the given simulated behavior) and behavior prediction (the accuracy of replicating real users' behavior through simulation of group- and individual-level fitted models). For evaluating the latter, we measured the distance between the real user's behavior and the model's prediction, considering the following four behavior features: 1) WPM, 2) error rate, 3) backspace presses, and 4) KSPC. We used behavior simulated on the basis of the previously reported model parameters [30] as a baseline for comparison. In that the original study borrowed the parameter values from prior literature without fitting the model to empirical datasets, the baseline in Case 3 represents common practice in the HCI field.

With Case 3, we showcase one of the promising areas for applying high-computational-efficiency amortized inference: large-scale analysis of user data. With reference to the large-scale dataset [56], we can report on how the inferred cognitive parameters of individuals are distributed in a population (N=1,057), in terms of two demographic factors: age and gender.

Table 3: Behavior-prediction accuracy with the touchscreen-typing model (Case 3). Baseline represents model prediction with imported parameter values from literature as in [30]. Each row's reporting of the distance between the observed and model-predicted values presents the closest results in green and the second-closest in light green.

Behavior	Distance metric	Baseline [30]	Amortized inference	
			Group-level fit	Individual-level fit
WPM	Mean diff.	6.8432	5.4317	1.7769
	KLD	1.8028	1.0935	0.1787
	MMD	1.0835	0.5457	0.0376
Error rate	Mean diff.	0.9548	0.4310	0.2327
	KLD	1.5111	1.1527	0.4968
	MMD	1.3805	1.2656	1.2410
Backspace count	Mean diff.	0.8204	3.5884	2.0648
	KLD	0.1328	0.3374	0.1059
	MMD	0.1202	0.3785	0.2295
KSPC	Mean diff.	0.0865	0.0439	0.0098
	KLD	3.5048	2.5738	0.9203
	MMD	0.4665	0.4284	0.4007

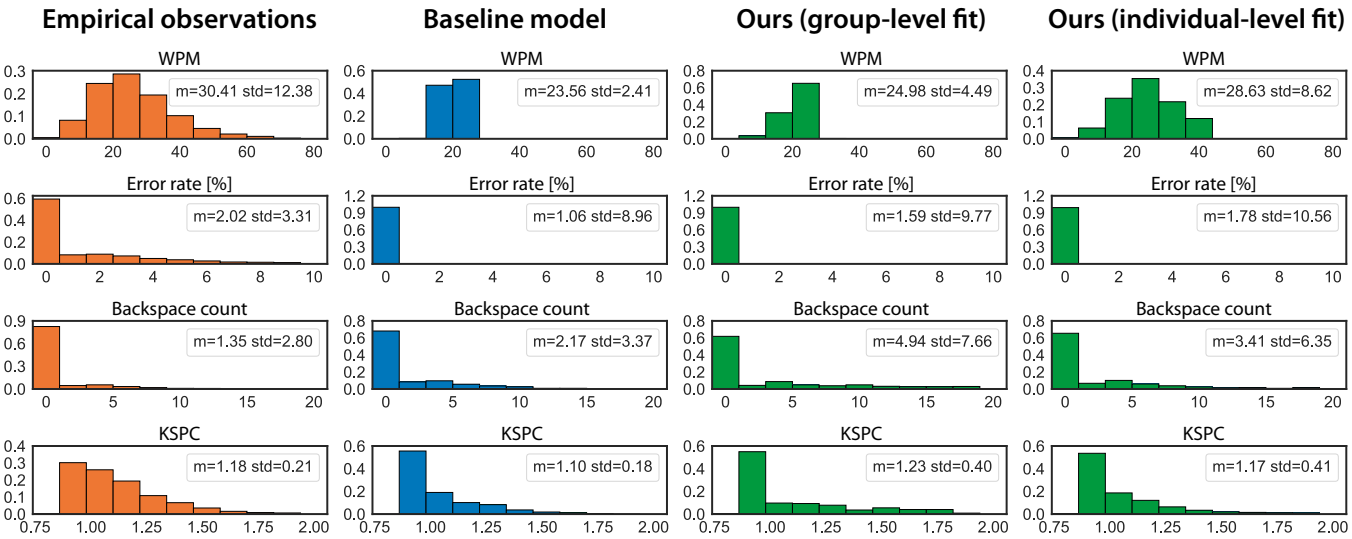

Figure 11: Comparison of the actual behaviors observed and those predicted under the touchscreen-typing model (Case 3).

7.3 Results

The inference process took about 15 ms per given behavior dataset. Accordingly, the time for the individual-level inference of 1,057 users' data with amortized inference was only 20 seconds in total.

Figure 10(a) depicts the parameter-recovery performance from the simulated behavior with our trained density estimator. Our trained estimator predicted all three model parameters with a high level of coefficient determination (R^2=0.82 for p_{obs}, 0.71 for α, and 0.96 for k). Figure 10(b) plots the estimated posterior distributions from the aggregated behavior data of all 1,057 users. The baseline model parameters (adopted from the literature) fall within the estimated posterior distribution but not its high-probability region. We interpret this as related to a limitation of using values imported from the literature: they are not always entirely representative of the dataset, and other confounding factors may be present. Our inferred parameters' superiority to the literature-derived baseline ones for behavior prediction may support this interpretation. Table 3 attests that, by most metrics, our group-level fitted model parameter values replicate real user behavior better than the baseline parameter values do. Furthermore, fitting the model at individuals' level yielded even better prediction performance, with the predictions best approaching real user behavior. The full set of behavior histograms is reproduced in Figure 11.

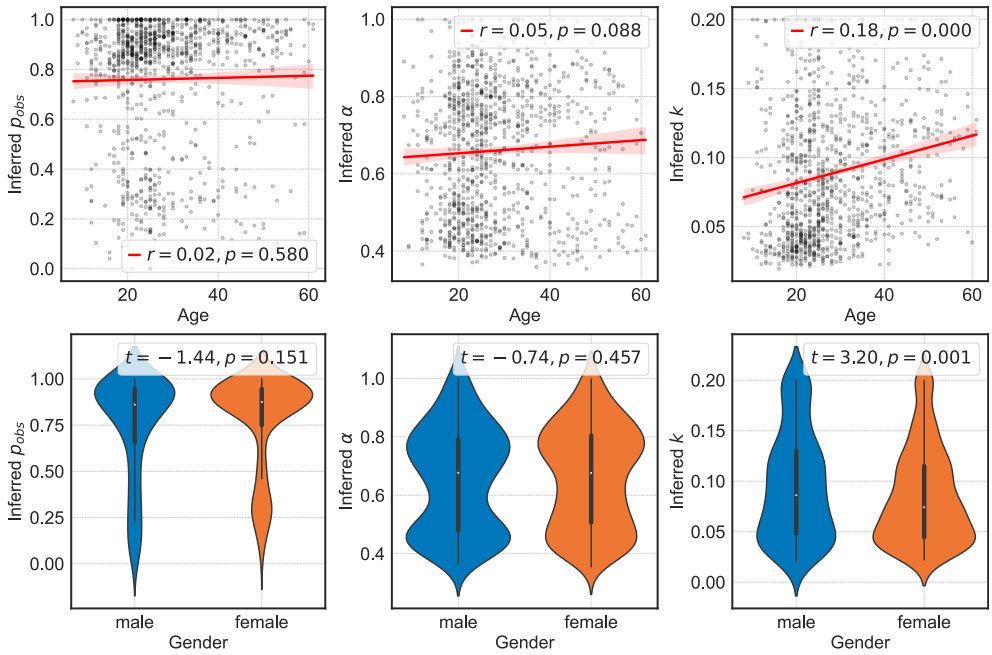

Figure 12: Amortized inference can facilitate analyzing large-scale user datasets, identifying how the inferred cognitive-physiological parameters of individuals are distributed in the population. Here, inferred touchscreen-typing model parameters for 1,057 individual users are plotted against user age (upper row) and gender (lower row).

Figure 12 shows how the parameter values inferred for each individual participant are distributed population-wise in terms of both age and gender. We detected a significant correlation between the users' age and inferred k, which represents the motor resources behind a user's finger (Pearson's r=0.18, with p<0.001). Therefore, data from typing behavior might reveal that younger people show more extensive finger-related motor resources (i.e., lower k). This result is consistent with previous work [69]. We found no significant age correlation for the other two parameters, p_{obs} (probability of noticing an error from finger movement) and α (the movement's speed–accuracy bias), with p-values of 0.580 and 0.088, respectively. On the other hand, regarding gender, we discovered a significant difference between male and female user groups' inferred k values (independent-samples t-test: t=3.20, with p<0.001). This result revealed finger motor resources to be stronger in female than in male users, even though the latter were older (the female users' mean age being 29.75 years and males' 25.10). The other two parameters did not differ significantly in either respect between the gender groups (p=0.151 for p_{obs}; 0.457 for α).

Table 4: Parameter-recovery performance (R^2 values) with the point-and-click model (Case 2) for the three conditions: *full data*, *same split*, and *crossed split*. The trained density estimator showed a certain degree of robustness when tested on a user task not covered by the training data, a few dips in performance aside. Δ represents the value obtained by subtracting the result in the first of these conditions from each under the split conditions; in two cases, $|\Delta| \geq 0.1$ (shown in red) or $0.1 > |\Delta| \geq 0.05$ (light red).

Inferred parameter	Full data	Training on split dataset			
		Same split	Δ	Crossed split	Δ
σ_v	0.930	0.937	0.007	0.930	0.000
n_v	0.406	0.429	0.023	0.294	−0.112
c_σ	0.680	0.718	0.038	0.698	0.018
$T_{h,max}$	0.857	0.854	−0.003	0.798	−0.059

8 ROBUSTNESS ANALYSIS

Robustness is constancy of performance irrespective of the variations in the internal and external factors [39]. For example, inference performance might be sensitive to the researcher's choice of training settings or to distributional shifts in test data. By examining robustness, we can verify how reliably an approach performs with uncertain, noisy real-world data or copes with changing the training setting to meet particular needs. Our discussion here examines how inference performance gets affected when the estimators tackle unseen user behavior, from different spaces of task variables and prior distributions of model parameters. Then, we consider the performance effects of employing not a density estimator (with full posteriors) but a point estimator (predicting point values for the parameters).

8.1 Out-of-Distribution Data

Testing with out-of-distribution data – data subject to distributions different from the training data's – can clarify the robustness to

Table 5: Performance (R^2 values) with the touchscreen-typing model (Case 3) for various prior settings, demonstrating that the density estimator's performance suffered when evaluated on the basis of priors different from the training data. Δ represents the value obtained by subtracting the result of the model learned under *Literature-prior* from the result of that learned under *Uniform-prior*; the results where $|\Delta| \geq 0.1$ are shown in green or red, and those where $0.1 > |\Delta| \geq 0.05$ are in light green or red.

Inferred parameter	Evaluated under Uniform-prior			Evaluated under Literature-prior		
	Uniform → Uniform	Literature → Uniform	Δ	Uniform → Literature	Literature → Literature	Δ
p_{obs}	0.869	0.466	0.403	0.638	0.818	−0.180
α	0.767	0.514	0.253	0.099	0.705	−0.606
k	0.951	0.308	0.643	0.888	0.958	−0.070

unseen user behavior. To test the density estimator with out-of-distribution data, we assumed a situation wherein differences in the task variables given to a user in each trial cause different user behavior to emerge. We split the dataset used with the point-and-click model (Case 2) in half in the task-variable space consisting of the given pointing target's radius and speed. Three distinct experiment setups were prepared, *full data*, in which the estimator was trained on the entire dataset and tested on each split dataset; *same split*, with the estimator trained on and tested on the same subset; and *crossed split*, with testing on one portion of the two-part split and testing on the other. Table 4 shows the results for averaged parameter-recovery performance (i.e., R^2 values) for both split datasets in each condition. The R^2 values for both parameters (n_v and $T_{h,max}$) were lower in the *crossed split* condition than in *full data* or *same split*. Despite the performance decreases, however, the density estimator in *crossed split* was able to recover all parameters, with high (σ_v, c_σ, and $T_{h,max}$) or low (n_v) R^2 levels, from the data of the unseen task.

8.2 Prior Sensitivity

Having strong priors of model parameters can enhance inference performance. In inverse modeling problems in HCI, the priors are the scientifically plausible distributions of a user's cognitive-physiological characteristics. Therefore, they are often set to have a peak at the most plausible external values validated by the literature [34]. To measure the impact of priors, we compared the following two conditions, for ways in which a practitioner may set the priors: *Literature-prior*, with distributions weighted at plausible values adopted from the literature (as in the case studies), and *Uniform-prior*, using flattened distributions from the literature prior, without any weighting for external values. We evaluated the density estimator trained with each prior for all three cases. The results for the touchscreen-typing task (Case 3), which is the most challenging scenario, are presented below (Supplement E presents the results for cases 1 and 2). Each estimator's parameter-recovery performance degrades when it grapples with synthetic data from the other priors (see Table 5). In fitting of the model to real user data at the level of individuals, using the literature-based prior outperformed use of the uniform prior by most behavior-prediction metrics (see Table E5 in Supplement E). However, we discovered that even the naive *Uniform-prior* density estimator could outperform previous work's fitting baselines (ABC- or hand-tuning-based).

Table 6: Parameter-recovery performance (R^2 values) for the touchscreen-typing model (Case 3) with the point estimator and density estimator, which performed comparably in their parameter recovery. Δ represents the value obtained by subtracting the result of the density estimator from that of the point estimator; there were no cases where $|\Delta| \geq 0.05$.

Inferred parameter	Point estimator	Density estimator	Δ
p_{obs}	0.855	0.818	0.037
α	0.677	0.705	−0.028
k	0.956	0.958	−0.002

8.3 Choice between Point and Density Estimators

As researcher needs dictate, one can train a point estimator instead of a density estimator, using the same workflow (see Section 3.5 for details). We compared the two estimator types' inference performance, with all three case studies. Tables 6 shows the results for Case 3 (Supplement F presents the results for the other two cases). We implemented a point estimator by replacing the conditional INN with an MLP that has two hidden layers (of 512 units each), which directly output the optimal parameter values. There was no noticeable difference in parameter recovery between the two estimator types when tested on a simulated dataset. In addition, when fitted to the actual user dataset at individuals' level, they were comparable in performance, both proving superior to the baseline (See Table F5 in Supplement F). One notable difference was in the computation cost per inference: the point estimator, taking about 5 ms, was three times faster than the density estimator.

9 DISCUSSION

Across multiple cases, we have demonstrated four key benefits of amortized inference in inverse modeling for HCI research:

- High predictive accuracy for individual users: Our reported levels of accuracy were on par with or better than with non-amortized approaches (e.g., ABC).
- Computational efficiency: Fitting a model to an individual user's data, which used to demand hours or days (with ABC), took only 10–125 ms for the same data (cases 1–2).

- Large-scale parameter inference: Thanks to high efficiency, amortized inference aids in estimating how model parameters are distributed in a population and how they correlate with factors not accounted for in the model (e.g., age and gender in Case 3).
- A principled approach to uncertainty: As ABC does but unlike regular optimization-based approaches such as Nelder–Mead [52], inverse modeling with amortized inference affords estimating posterior distributions of model parameters. This holds value for HCI work with small-scale or noisy observations of users.

The following review of core aspects of the case studies' results discusses our findings' implications and what might affect performance.

Inference efficiency. The complexity of the behavioral data behind inference ($\boldsymbol{y}$) affects computational efficiency. In Case 1's conditions, $\boldsymbol{y}$ consists of summary features, 1-D data with only a feature dimension. The other cases had a trial dimension added (since all data, from multiple trials, were used). Case 2 involved a time dimension additionally (because each trial's data included a time-series trajectory), so it had the most complex $\boldsymbol{y}$ (3-D data, with trial, time, and feature dimensions). This necessitated a more complicated encoder network, which, in turn, increased the inference time. Nonetheless, inference time did not grow with the number of observed trials in $\boldsymbol{y}$ (Figure 13(a)), thanks to the parallel computation of neural networks (i.e., multiple trials' computations can be processed as a batch).

Parameter recovery. We discovered that parameter-recovery performance may differ not only between models but also within the parameters of a model. While our trained density estimator could achieve decent recovery performance for most parameters, a few parameters showed lower R^2 values (e.g., p_{sem} in Case 1 and n_v in Case 2). One influential factor is the *identifiability* of the model parameters. If there are multiple solutions for a certain parameter that could produce the same model output, that parameter can be difficult to isolate from the given data. This is sometimes due to the model itself (e.g., model sloppiness [26]) or small quantities of given data. One can put amortized inference to use for rapidly revealing gaps in the given data as one examines the convergence of parameter-recovery performance with increasing dataset sizes. For example, from Figure 13(b) we can identify the number of observed trials at which performance starts to converge; we need at least 16 trials to recover k with nearly optimal accuracy and 128 trials for α in Case 3. The *approximation gap* and *amortization gap* are two additional elements that could influence parameter-recovery performance [12]. The approximation gap refers to error resulting from limits to the density estimator's representation power; that is, it results from our attempt to approximate the true posterior distribution by using a surrogate (parameterized) distribution. The amortization gap refers to error caused by training a density estimator for the entire dataset, as opposed to optimizing an approximate distribution for each individual user's data.

Individual-level fitting. Amortized inference fitted the parameters better at the level of individuals than at that of the whole user group. Across all three cases, parameters fitted at individuals' level

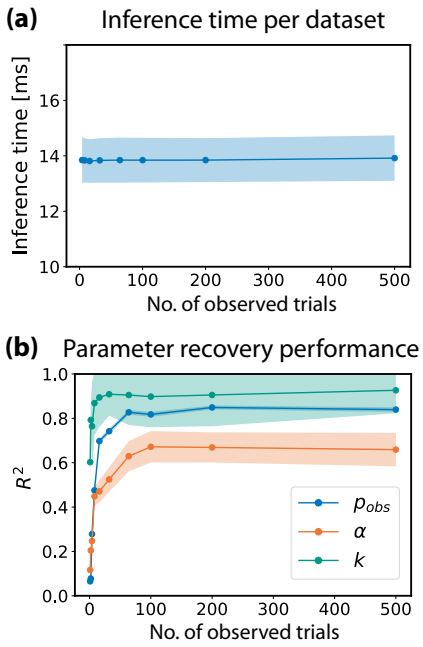

Figure 13: The effect of dataset size (i.e., the number of task trials observed) on (a) inference time per dataset and (b) parameter recovery from simulated data, in Case 3 (touchscreen typing). We obtained the result lines by averaging 1,000 iterations for each size of dataset. Shading denotes the standard deviation across iterations.

showed the greatest accuracy. That said, there were exceptions (e.g., individual-level fitting was not superior to group-level fitting for backspacing in Case 3). This might be a result of constraints in the behavior space that the simulation model can express; for example, some behavioral features reproduced by the model may display inherent mutual dependencies. If the constraints are not aligned with real users' behavior space, it will be impossible to replicate their behavior features in full. Then, one must find the closest possible point instead.

Scaling up the number of users. We were able to infer model parameters for 1,057 individuals by means of amortized inference, whereas the previous highest number was 211, with ABC [18]. The inference process for all users took only 20 seconds in total. This level of efficiency is remarkable and signals that the method scales up well. It affords analysis that captures even smaller effects in datasets and can connect them with parameters of a simulation model. The previously unreported difference we revealed between male and female users' motor resources in finger movement (k) is a case in point (Figure 14).

Robustness to distributional shifts. The density estimator showed some robustness to distribution shifts between training and test data, especially in dealing with data from unseen task conditions (e.g., the pointing target's speed in Case 2) or different priors. Still, to achieve better inference performance against empirical data, one needs to hone the priors from the literature and make the training

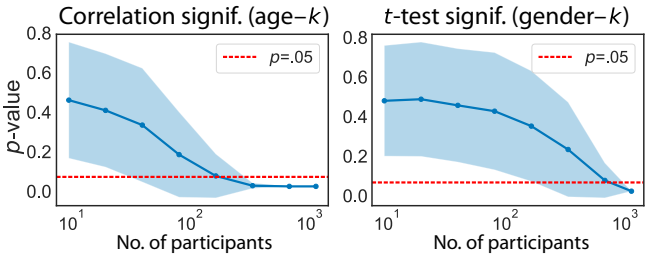

Figure 14: Amortized inference helps simulation modeling explain smaller effects contained in datasets. The plot shows how increasing the number of participants in individual-level inference affects statistically reliable findings: the correlation between inferred k and the participants' age (at left) and the difference in inferred k between male and female participants, in Case 3, touchscreen typing (at right). The result lines were obtained by averaging 1,000 iterations with participants randomly sampled from the full dataset (N=1,057). Shading denotes the standard deviation across iterations.

data as comprehensive as possible. Cases of other types can inform testing of density-estimator robustness. Questions remain with regard to, for instance, training data generated by mechanisms that do not perfectly match real-world data.

Point estimator vs. density estimator. Our findings suggest that the two estimator types manifest a tradeoff relationship. A point estimator can further reduce the computation cost of inference, yet a density estimator might be more beneficial in several cases: 1) rigorously checking the reliability of the inferred values, 2) identifying multiple plausible explanations for observations, and 3) using the inferred posterior as a new prior to perform multi-round inference that is expected to achieve higher accuracy (e.g., in sequential neural posterior estimation [23, 57]).

9.1 Applications

Together, the robustness and low computation cost brought by amortized inference offer an exciting vista for future application of simulation models in 1) adaptive user interfaces, 2) recommendation systems, 3) diagnostic tools for user modeling, and 4) large-scale behavior analysis tools.

For the first of these, parameters inferred for a user could form the basis for adapting an interface optimally for the individual. For example, simulation models aid in optimizing a keyboard layout for people with such impairments as dyslexia or tremor [68]. Adaptation of this sort via simulation models has been handled mainly on a non-real-time basis thus far [68, 73, 74]. Our approach could open the door to developing adaptive user interfaces in a new way, enabling repetitive processes of inference from observations and optimization in real time. Secondly, amortized inference could advance interactive recommendation systems through inferring a user's temporal intention and interest. For instance, future simulation models could assist in better disentangling various underpinnings of click data (individuals' preference, intention, curiosity, etc.). As for developing user models with theory-based mechanisms, parameter inference enables a proposed model to be evaluated with actual user data, and amortized inference can speed up the investigation. Also, parameter identifiability is often of interest to modelers; the shape of the inferred posterior distribution informs how well each parameter can be distinguished via what is observable. With this approach, one can ensure the validity of datasets in future experiments and identify the number of observations required for valid inferences. Finally, low-cost inverse modeling enables applying simulation models to identify humans' latent capabilities from datasets orders of magnitude larger than feasible ever before. For example, by inversely modeling an individual's behavior, one could recover the attributes of the user's motor and cognitive systems – muscle activation, memory recall, and many more.

10 CONCLUSION AND FUTURE WORK

We have found great promise in the proposed workflow for applying amortized inference to solve the inverse modeling problem in HCI at lower computational cost. However, research still must address several challenges, for wider applications of amortized inference. Firstly, the field lacks best practice for choosing which of the various observable-behavior data to use for inference and in what form. Several tradeoffs merit further study too; e.g., using all the observable data and letting the network extract effective features may lead to better performance [61] but slows the training and inference processes. The second problem remaining is that of appropriate prior distributions for model parameters, especially where the literature's reference values are unapplicable or otherwise limited. Thirdly, it would be worthwhile to investigate the density estimator's upper limits and potential losses when dealing with more model parameters. A recent project outside the HCI field [22] succeeded in inferring as many as 31 parameters, thus attesting to the neural network's promise. A final issue, by no means least, is compensating for discrepancies between models' simulations and the real-world behavior data described (e.g., from imperfect mechanisms or the "reality gap" [75]) and making the amortized inference more resilient.

ACKNOWLEDGMENTS

The research was supported by Korea's Institute of Information and Communications Technology Planning and Evaluation (2020-0-01361), the National Research Foundation of Korea (NRF-2020R1A2-C400214612), the Korea Creative Content Agency (R2021040105), the Finnish Center for AI, the Academy of Finland (under "Human Automata" and "BAD"), and the Department of Information and Communications Engineering at Aalto University. The source code of this work is available at https://github.com/hsmoon121/amortized-inference-hci.

REFERENCES

[1] John R Anderson. 1996. ACT: A simple theory of complex cognition. *American Psychologist* 51, 4 (1996), 355–365.
[2] Lynton Ardizzone, Jakob Kruse, Carsten Rother, and Ullrich Köthe. 2018. Analyzing inverse problems with invertible neural networks. In *International Conference on Learning Representations*.
[3] Gilles Bailly, Antti Oulasvirta, Duncan P Brumby, and Andrew Howes. 2014. Model of visual search and selection time in linear menus. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 3865–3874.
[4] Mark A Beaumont, Wenyang Zhang, and David J Balding. 2002. Approximate Bayesian computation in population genetics. *Genetics* 162, 4 (2002), 2025–2035.

[5] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. 2017. Variational inference: A review for statisticians. *J. Amer. Statist. Assoc.* 112, 518 (2017), 859–877.

[6] George Erich Brogmus. 1991. Effects of age and sex on speed and accuracy of hand movements: And the refinements they suggest for Fitts' law. *Proceedings of the Human Factors Society Annual Meeting* 35, 3 (1991), 208–212.

[7] Rachael A Burno, Bing Wu, Rina Doherty, Hannah Colett, and Rania Elnaggar. 2015. Applying Fitts' law to gesture based computer interactions. *Procedia Manufacturing* 3 (2015), 4342–4349.

[8] Stuart K Card, Thomas P Moran, and Allen Newell. 1983. *The psychology of human–computer interaction.* Lawrence Erlbaum Associates.

[9] Noshaba Cheema, Laura A Frey-Law, Kourosh Naderi, Jaakko Lehtinen, Philipp Slusallek, and Perttu Hämäläinen. 2020. Predicting mid-air interaction movements and fatigue using deep reinforcement learning. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.

[10] Xiuli Chen, Gilles Bailly, Duncan P Brumby, Antti Oulasvirta, and Andrew Howes. 2015. The emergence of interactive behavior: A model of rational menu search. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. 4217–4226.

[11] Kyle Cranmer, Johann Brehmer, and Gilles Louppe. 2020. The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences* 117, 48 (2020), 30055–30062.

[12] Chris Cremer, Xuechen Li, and David Duvenaud. 2018. Inference suboptimality in variational autoencoders. In *International Conference on Machine Learning*. PMLR, 1078–1086.

[13] Maximilian Dax, Stephen R Green, Jonathan Gair, Jakob H Macke, Alessandra Buonanno, and Bernhard Schölkopf. 2021. Real-time gravitational wave science with neural posterior estimation. *Physical Review Letters* 127, 24 (2021), 241103.

[14] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. 2017. Density estimation using Real NVP. In *International Conference on Learning Representations*.

[15] Seungwon Do, Minsuk Chang, and Byungjoo Lee. 2021. A simulation model of intermittently controlled point-and-click behaviour. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–17.

[16] Florian Fischer, Miroslav Bachinski, Markus Klar, Arthur Fleig, and Jörg Müller. 2021. Reinforcement learning control of a biomechanical model of the upper extremity. *Scientific Reports* 11, 1 (2021), 1–15.

[17] Paul M Fitts. 1954. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology* 47, 6 (1954), 381–391.

[18] Christoph Gebhardt, Antti Oulasvirta, and Otmar Hilliges. 2021. Hierarchical reinforcement learning explains task interleaving behavior. *Computational Brain & Behavior* 4, 3 (2021), 284–304.

[19] Samuel Gershman and Noah Goodman. 2014. Amortized inference in probabilistic reasoning. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, Vol. 36.

[20] Samuel J Gershman, Eric J Horvitz, and Joshua B Tenenbaum. 2015. Computational rationality: A converging paradigm for intelligence in brains, minds, and machines. *Science* 349, 6245 (2015), 273–278.

[21] Manuel Glöckler, Michael Deistler, and Jakob H Macke. 2022. Variational methods for simulation-based inference. In *International Conference on Learning Representations*.

[22] Pedro J Gonçalves, Jan-Matthis Lueckmann, Michael Deistler, Marcel Nonnenmacher, Kaan Öcal, Giacomo Bassetto, Chaitanya Chintaluri, William F Podlaski, Sara A Haddad, Tim P Vogels, et al. 2020. Training deep neural density estimators to identify mechanistic models of neural dynamics. *eLife* 9 (2020), e56261.

[23] David Greenberg, Marcel Nonnenmacher, and Jakob Macke. 2019. Automatic posterior transformation for likelihood-free inference. In *International Conference on Machine Learning*. PMLR, 2404–2414.

[24] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. 2012. A kernel two-sample test. *Journal of Machine Learning Research* 13, 1 (2012), 723–773.

[25] Yves Guiard and Olivier Rioul. 2015. A mathematical description of the speed/accuracy trade-off of aimed movement. In *Proceedings of the 2015 British HCI Conference*. 91–100.

[26] Ryan N Gutenkunst, Joshua J Waterfall, Fergal P Casey, Kevin S Brown, Christopher R Myers, and James P Sethna. 2007. Universally sloppy parameter sensitivities in systems biology models. *PLoS Computational Biology* 3, 10 (2007), e189.

[27] Michael U Gutmann and Jukka Corander. 2016. Bayesian optimization for likelihood-free inference of simulator-based statistical models. *Journal of Machine Learning Research* 17 (2016).

[28] Peter C Humphreys, David Raposo, Tobias Pohlen, Gregory Thornton, Rachita Chhaparia, Alistair Muldal, Josh Abramson, Petko Georgiev, Adam Santoro, and Timothy Lillicrap. 2022. A data-driven approach for learning to control computers. In *International Conference on Machine Learning*. PMLR, 9466–9482.

[29] Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. 2021. Perceiver: General perception with iterative attention. In *International Conference on Machine Learning*. PMLR, 4651–4664.

[30] Jussi Jokinen, Aditya Acharya, Mohammad Uzair, Xinhui Jiang, and Antti Oulasvirta. 2021. Touchscreen typing as optimal supervisory control. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–14.

[31] Jussi PP Jokinen, Tuomo Kujala, and Antti Oulasvirta. 2021. Multitasking in driving as optimal adaptation under uncertainty. *Human Factors* 63, 8 (2021), 1324–1341.

[32] Jussi PP Jokinen, Sayan Sarcar, Antti Oulasvirta, Chaklam Silpasuwanchai, Zhenxin Wang, and Xiangshi Ren. 2017. Modelling learning of new keyboard layouts. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 4203–4215.

[33] Jussi PP Jokinen, Zhenxin Wang, Sayan Sarcar, Antti Oulasvirta, and Xiangshi Ren. 2020. Adaptive feature guidance: Modelling visual search with graphical layouts. *International Journal of Human–Computer Studies* 136 (2020), 102376.

[34] Antti Kangasrääsiö, Kumaripaba Athukorala, Andrew Howes, Jukka Corander, Samuel Kaski, and Antti Oulasvirta. 2017. Inferring cognitive models from data using approximate Bayesian computation. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 1295–1306.

[35] Antti Kangasrääsiö, Jussi PP Jokinen, Antti Oulasvirta, Andrew Howes, and Samuel Kaski. 2019. Parameter inference for computational cognitive models with Approximate Bayesian Computation. *Cognitive Science* 43, 6 (2019), e12738.

[36] Davis E Kieras and Davis E Meyer. 1997. An overview of the EPIC architecture for cognition and performance with application to human–computer interaction. *Human–Computer Interaction* 12, 4 (1997), 391–438.

[37] Durk P Kingma and Prafulla Dhariwal. 2018. Glow: Generative flow with invertible 1×1 convolutions. *Advances in Neural Information Processing Systems* 31 (2018).

[38] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114* (2013).

[39] Hiroaki Kitano. 2004. Biological robustness. *Nature Reviews Genetics* 5, 11 (2004), 826–837.

[40] Jakob Kruse, Lynton Ardizzone, Carsten Rother, and Ullrich Köthe. 2021. Benchmarking invertible architectures on inverse problems. *arXiv preprint arXiv:2101.10763* (2021).

[41] Minhae Kwon, Saurabh Daptardar, Paul R Schrater, and Xaq Pitkow. 2020. Inverse rational control with partially observable continuous nonlinear dynamics. *Advances in Neural Information Processing Systems* 33 (2020), 7898–7909.

[42] Gary D Langolf, Don B Chaffin, and James A Foulke. 1976. An investigation of Fitts' law using a wide range of movement amplitudes. *Journal of Motor Behavior* 8, 2 (1976), 113–128.

[43] Vladimir I Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics – Doklady* 10, 8 (1966), 707–710.

[44] Richard L Lewis, Andrew Howes, and Satinder Singh. 2014. Computational rationality: Linking mechanism and behavior through bounded utility maximization. *Topics in Cognitive Science* 6, 2 (2014), 279–311.

[45] Jan-Matthis Lueckmann, Pedro J Goncalves, Giacomo Bassetto, Kaan Öcal, Marcel Nonnenmacher, and Jakob H Macke. 2017. Flexible statistical inference for mechanistic models of neural dynamics. *Advances in Neural Information Processing Systems* 30 (2017).

[46] Julieta Martinez, Michael J Black, and Javier Romero. 2017. On human motion prediction using recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2891–2900.

[47] Matthew Millard, Thomas Uchida, Ajay Seth, and Scott L Delp. 2013. Flexing computational muscle: Modeling and simulation of musculotendon dynamics. *Journal of Biomechanical Engineering* 135, 2 (2013).

[48] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[49] Hee-Seung Moon, Seungwon Do, Wonjae Kim, Jiwon Seo, Minsuk Chang, and Byungjoo Lee. 2022. Speeding up inference with user simulators through policy modulation. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–21.

[50] Hee-Seung Moon and Jiwon Seo. 2021. Fast user adaptation for human motion prediction in physical human–robot interaction. *IEEE Robotics and Automation Letters* 7, 1 (2021), 120–127.

[51] Roderick Murray-Smith, John H Williamson, Andrew Ramsay, Francesco Tonolini, Simon Rogers, and Antoine Loriette. 2021. Forward and inverse models in HCI: Physical simulation and deep learning for inferring 3D finger pose. *arXiv preprint arXiv:2109.03366* (2021).

[52] John A Nelder and Roger Mead. 1965. A simplex method for function minimization. *The Computer Journal* 7, 4 (1965), 308–313.

[53] Antti Oulasvirta, Jussi PP Jokinen, and Andrew Howes. 2022. Computational rationality as a theory of interaction. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–14.

[54] Antti Oulasvirta, Sunjun Kim, and Byungjoo Lee. 2018. Neuromechanics of a button press. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–13.

[55] Brooks Paige and Frank Wood. 2016. Inference networks for sequential Monte Carlo in graphical models. In *International Conference on Machine Learning*. PMLR, 3040–3049.

[56] Kseniia Palin, Anna Maria Feit, Sunjun Kim, Per Ola Kristensson, and Antti Oulasvirta. 2019. How do people type on mobile devices? Observations from a study with 37,000 volunteers. In *Proceedings of the 21st International Conference on Human–Computer Interaction with Mobile Devices and Services*. 1–12.

[57] George Papamakarios and Iain Murray. 2016. Fast ε-free inference of simulation models with Bayesian conditional density estimation. *Advances in Neural Information Processing Systems* 29 (2016).

[58] George Papamakarios, Eric T Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. 2021. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research* 22, 57 (2021), 1–64.

[59] Stephen J Payne and Andrew Howes. 2013. Adaptive interaction: A utility maximization approach to understanding human interaction with technology. *Synthesis Lectures on Human-Centered Informatics* 6, 1 (2013), 1–111.

[60] Stefan T Radev, Frederik Graw, Simiao Chen, Nico T Mutters, Vanessa M Eichel, Till Bärnighausen, and Ullrich Köthe. 2021. OutbreakFlow: Model-based Bayesian inference of disease outbreak dynamics with invertible neural networks and its application to the COVID-19 pandemics in Germany. *PLoS Computational Biology* 17, 10 (2021), e1009472.

[61] Stefan T Radev, Ulf K Mertens, Andreas Voss, Lynton Ardizzone, and Ullrich Köthe. 2020. BayesFlow: Learning complex stochastic models with invertible neural networks. *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[62] Stefan T Radev, Ulf K Mertens, Andreas Voss, and Ullrich Köthe. 2020. Towards end-to-end likelihood-free inference with convolutional neural networks. *British Journal of Mathematical and Statistical Psychology* 73, 1 (2020), 23–43.

[63] Robert G Radwin, Gregg C Vanderheiden, and Mei-Li Lin. 1990. A method for evaluating head-controlled computer input devices using Fitts' law. *Human Factors* 32, 4 (1990), 423–438.

[64] Roger Ratcliff. 1978. A theory of memory retrieval. *Psychological Review* 85, 2 (1978), 59–108.

[65] Roger Ratcliff and Francis Tuerlinckx. 2002. Estimating parameters of the diffusion model: Approaches to dealing with contaminant reaction times and parameter variability. *Psychonomic Bulletin & Review* 9, 3 (2002), 438–481.

[66] Danilo Rezende and Shakir Mohamed. 2015. Variational inference with normalizing flows. In *International Conference on Machine Learning*. PMLR, 1530–1538.

[67] Dario D Salvucci. 2001. An integrated model of eye movements and visual encoding. *Cognitive Systems Research* 1, 4 (2001), 201–220.

[68] Sayan Sarcar, Jussi PP Jokinen, Antti Oulasvirta, Zhenxin Wang, Chaklam Silpasuwanchai, and Xiangshi Ren. 2018. Ability-based optimization of touchscreen interactions. *IEEE Pervasive Computing* 17, 1 (2018), 15–26.

[69] Sayan Sarcar, Jussi Joklnen, Antti Oulasvirta, Chaklam Silpasuwanchai, Zhenxin Wang, and Xiangshi Ren. 2016. Towards ability-based optimization for aging users. In *Proceedings of the International Symposium on Interactive Technology and Ageing Populations*. 77–86.

[70] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[71] Alan A Stocker and Eero P Simoncelli. 2006. Noise characteristics and prior expectations in human visual speed perception. *Nature Neuroscience* 9, 4 (2006), 578–585.

[72] Andreas Stuhlmüller, Jacob Taylor, and Noah Goodman. 2013. Learning stochastic inverses. *Advances in Neural Information Processing Systems* 26 (2013).

[73] Kashyap Todi, Gilles Bailly, Luis Leiva, and Antti Oulasvirta. 2021. Adapting user interfaces with model-based reinforcement learning. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–13.

[74] Kashyap Todi, Jussi Jokinen, Kris Luyten, and Antti Oulasvirta. 2019. Individualising graphical layouts with predictive visual search models. *ACM Transactions on Interactive Intelligent Systems* 10, 1 (2019), 1–24.

[75] Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. 2018. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 969–977.

[76] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* 30 (2017).

[77] Martin J Wainwright and Michael I Jordan. 2008. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning* 1, 1–2 (2008), 1–305.

[78] Jacob O Wobbrock, Edward Cutrell, Susumu Harada, and I Scott MacKenzie. 2008. An error model for pointing based on Fitts' law. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1613–1622.