Supplementary Material for "Real-time 3D Target Inference via Biomechanical Simulation"

A NEURAL DENSITY ESTIMATION

In our inference model, we employed *RealNVP* as the normalizing flows architecture [1]. This architecture facilitates a bijective transformation between simple Gaussian-distributed parameters and the target distributions we aim to model, specifically, the selection target positions. By using normalizing flows, we achieve two key benefits: 1) we can efficiently sample from the inferred posterior distribution of the selection target positions, and 2) we can approximate the posterior probability for a given target position from observations.

Suppose the normalizing flows enable the bijective transformations between latent Gaussian variables $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \mathbb{I})$ and target positions $\boldsymbol{\theta}$. Specifically, these transformations are conditioned on the observation y (or a extracted feature representation $\hat{\boldsymbol{y}}$). The normalizing flows provide bidirectional transformations; that is, $\boldsymbol{\theta} = f_\phi(\boldsymbol{z}; \boldsymbol{y})$ and $\boldsymbol{z} = f_\phi^{-1}(\boldsymbol{\theta}; \boldsymbol{y})$, where f represents the normalizing flows parameterized by neural network parameters ϕ. Following prior work [8], the approximated posterior density of $\boldsymbol{\theta}$ given $\boldsymbol{y}$ can be calculated as follows:

$$p_\phi(\boldsymbol{\theta}|\boldsymbol{y}) = p(\boldsymbol{z} = f_\phi^{-1}(\boldsymbol{\theta}; \boldsymbol{y})) \left| \det \left(\frac{\partial f_\phi^{-1}(\boldsymbol{\theta}; \boldsymbol{y})}{\partial \boldsymbol{\theta}} \right) \right|. \tag{1}$$

Due to the Gaussian distribution of the variables $\boldsymbol{z}$, the term $p(\boldsymbol{z} = f_\phi^{-1}(\boldsymbol{\theta}; \boldsymbol{y}))$ can be straightforwardly computed. Additionally, the architectural design of normalizing flows simplifies the computation of the determinant of the Jacobian matrix for f_ϕ^{-1}.

The cornerstone of the RealNVP architecture is its utilization of *affine coupling layers* [1]. These layers serve a dual purpose: they facilitate bijective transformations and also enable the efficient computation of the determinant of the Jacobian matrix. To illustrate, consider a single affine coupling layer that transforms vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, conditioned on a observation $\tilde{\boldsymbol{y}}$. In the forward transformation $\boldsymbol{u} \rightarrow \boldsymbol{v}$, $\boldsymbol{u}$ is partitioned into two halves, $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$. The resulting vector $\boldsymbol{v}$ is then assembled from $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, calculated as:

$$\begin{aligned} \boldsymbol{v}_1 &= \boldsymbol{u}_1 \odot \exp(\mathrm{NN}_1(\boldsymbol{u}_2, \boldsymbol{y})) + \mathrm{NN}_2(\boldsymbol{u}_2, \boldsymbol{y}), \\ \boldsymbol{v}_2 &= \boldsymbol{u}_2 \odot \exp(\mathrm{NN}_3(\boldsymbol{v}_1, \boldsymbol{y})) + \mathrm{NN}_4(\boldsymbol{v}_1, \boldsymbol{y}). \end{aligned}$$

Here, $\odot$ denotes element-wise multiplication, and NN_1 to NN_4 represent nonlinear neural network mappings, the weights of which are encapsulated within the parameter set ϕ. The observation $\boldsymbol{y}$ acts as an external input to each of these neural networks, enabling the transformation to be conditioned on $\boldsymbol{y}$. The inverse transformation $\boldsymbol{v} \rightarrow \boldsymbol{u}$ can be similarly derived, and is likewise conditioned on $\boldsymbol{y}$:

$$\begin{aligned} \boldsymbol{u}_2 &= (\boldsymbol{v}_2 - \mathrm{NN}_4(\boldsymbol{v}_1, \boldsymbol{y})) \odot \exp(-\mathrm{NN}_3(\boldsymbol{v}_1, \boldsymbol{y})), \\ \boldsymbol{u}_1 &= (\boldsymbol{v}_1 - \mathrm{NN}_2(\boldsymbol{u}_2, \boldsymbol{y})) \odot \exp(-\mathrm{NN}_1(\boldsymbol{u}_2, \boldsymbol{y})). \end{aligned}$$

The Jacobian matrix corresponding to the affine transformation ($\boldsymbol{u} \rightarrow \boldsymbol{v}$) is formulated as follows:

$$\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{u}^T} = \begin{pmatrix} \mathrm{diag}(\exp(\mathrm{NN}_1(\boldsymbol{u}_2, \boldsymbol{y}))) & \frac{\partial \boldsymbol{v}_1}{\partial \boldsymbol{u}_2} \\ \mathbf{0} & \mathrm{diag}(\exp(\mathrm{NN}_3(\boldsymbol{v}_1, \boldsymbol{y}))) \end{pmatrix}$$

The structure of this Jacobian matrix ensures that it is either strictly upper or lower triangular. This architectural feature considerably simplifies the computation of its determinant, thereby facilitating the calculation of Eq. (1). To improve the expressiveness and performance of the coupling layers, random permutation matrices are introduced before each layer to promote enhanced mixing among the components of the variables $\boldsymbol{u}$ and $\boldsymbol{v}$. Additional implementation details can be found in the prior work [1, 6, 8].

B TRAINING OF THE USER SIMULATOR

B.1 Policy Networks

For our simulator's action policy, we employed both actor and critic neural networks, which are designed to process the task observations received from the simulated agent. These observations consist of visual and proprioceptive feedback. To extract relevant features from each type of observation, we utilize specialized encoders that are shared across the actor and critic networks. For visual feedback, we employ a three-layer Convolutional Neural Network (CNN). The first layer has 8 channels and uses a 5×5 kernel with 1×1 padding and a 2×2 stride, while the second and third layers each have 16 channels and employ a 3×3 kernel. Following the CNN, the visual observation passes through a max-pooling layer with a 2×2 kernel and is then flattened. A subsequent fully-connected (FC) layer with 128 output neurons is applied, followed by a Leaky Rectified Linear Unit (Leaky ReLU) activation function. For proprioceptive feedback, we use a single FC layer with 128 output neurons, also followed by a Leaky ReLU activation function.

Both the actor and critic networks in our simulator are equipped with fully-connected (FC) layers that directly follow the specialized encoders designed for task observation processing. The actor network is structured with three FC layers of dimensions $(256, 256, 8)$. The first two layers employ a tanh activation function. The 8 output neurons of the the final layer serves to represent the dimensions of the action space. Conversely, the critic network features three separate FC layers with dimensions $(256, 256, 1)$, accompanied by the same activation functions. The lone output neuron of the critic network is tasked with estimating the value of the action proposed by the actor network. To integrate user-specific free parameters (s_{limb}, σ_{sho}, σ_{elb}, σ_{wri}, and w_{fail}), both networks are engineered to accept these parameters in addition to the observation variables [4–6]. Specifically, these free parameters are concatenated to the input neurons for each FC layer and jointly processed with the features extracted from observations.

B.2 Implementation Details

To address the challenges posed by sparse reward settings in our simulator's target task, we implemented three key strategies: 1) the separation of policies for trial initiation and target selection, 2) distance-based reward shaping, and 3) curriculum learning.

The target selection task in our simulator consists of two distinct steps: 1) initiating the trial by interacting with the *starting object*, and 2) selecting the *selection target* via a point-and-click action. Accordingly, we trained two separate policies to handle each of these steps. The policy for trial initiation aims to guide the ray such that it passes through the starting object. During training, the initial position and orientation of the upper limb are randomly sampled for each episode, and the episode is terminated as successful if the ray passes through the starting object. Likewise, the policy for target selection was trained to point-and-click the selection target from randomized initial limb positions and orientations. Post-training, the two policies are integrated in the following manner: during simulation, the trial initiation policy is first employed until the agent's ray direct the starting object (i.e., the selection target appears). After this, a Gaussian-distributed time noise (mean=200 ms, SD=50 ms) is introduced to simulate human perception time.

Table B1. Hyperparameters for PPO training for the Simulator.

Parameter Name	Value
Gamma	0.999
Entropy Coefficient	0.005
Value Function Coefficient	0.5
Optimizer	Adam
Learning Rate	0.00005
Gradient Clipping	0.2
Curriculum Completion Step	40,000,000
Total Training Steps	100,000,000

Subsequently, the target selection policy is activated. Upon the "click" action determined by this policy, an additional Gaussian-distributed time noise (mean=200 ms, SD=50 ms) is introduced to mimic human reaction time for executing the click [7].

In order to facilitate the learning process for agents to direct the end effector toward the selection target, we employed distance-based reward shaping in conjunction with curriculum learning. During the early phases of training, the agents were rewarded according to a specially designed shaped-reward formulation. Rather than applying a uniform time-based penalty, denoted as w_{time}, at each timestep, we employed a distance-based penalty of $e^{-2d} - 1/10$ to guide the agent's learning, drawing inspiration from prior work [3]. In this formulation, d denotes the Euclidean distance between the end effector and the correct target. As the training process progresses, we incrementally shift from utilizing the distance-based penalty to the consistent time-based penalty. To manage this transition, we employ a curriculum variable that linearly scales between 0 and 1 over the course of training iterations. When the curriculum variable is 0, the penalty is solely distance-based; when it reaches 1, the penalty becomes exclusively time-based. This strategic transition ensures that the final, converged policy is influenced solely by the reward framework outlined in Section 4.1.2.

The curriculum includes a couple of additional mechanisms besides shifting from distance-based to time-based penalties. Firstly, we gradually increased the maximum levels of postural deviations, such as torso tilt and eye position, in each episode based on the curriculum variable. This approach allows the agent to initially confront simpler target selection tasks, incrementally introducing more challenging scenarios that involve greater postural deviations as training progresses. Secondly, the penalty associated with missed selections is gradually scaled from zero to its maximum value over the training period. This design encourages the agents to explore the utility of clicks early in training without being penalized, thereby accelerating quicker success in trials.

The entire training was done by the PPO algorithm [9], with the hyperparameters listed in Table B1.

C TRAINING OF THE INFERENCE MODEL

We employed the same neural network architecture for both Human-data-based Neural Inference and Simulation-based Neural Inference (our method). The observation input to the inference model comprises two components: 1) the configuration of objects and 2) the end-effector trajectory. The object configuration includes a 1-D target size and 3-D positions of potential selection targets (25 targets for the *Dense* grid and 26 targets for the *Wide* grid). The end-effector trajectory is represented by the last 30 steps of sequential 3-D positions, equating to the last 1.5 seconds of the trajectory.

Table C1. Hyperparameters for the inference model training.

Parameter Name	Value
Batch Size	2048
Optimizer	Adam
Gradient Clipping	0.5
Learning Rate Schedule	Cosine Annealing with Warm Restarts [2]
Max. Learning Rate	0.0001
Learning Rate Gamma	0.9
No. of Training Steps per Annealing Cycle	20,000
Total Training Steps	200,000

To process these observations, we used an encoder network consisting of a simple MLP with two FC layers, each having 128 neurons. The end-effector trajectory and object configurations were flattened and fed into the MLP, resulting in a 166-D input for the *Dense* grid and a 169-D input for the *Wide* grid. Tanh activation and batch normalization were applied between layers. We opted for a simple MLP over more specialized time-series architectures like LSTMs or Transformers for two reasons: 1) ease of conversion to ONNX format compatible with Unity's Barracuda engine, and 2) insignificant performance differences between the MLP and other structures found in our internal evaluation. For the normalizing flows, we implemented five RealNVP steps [1], each involving an affine coupling layer block. These blocks are composed of four neural-network-based mappings, denoted NN_1 to NN_4 in Section A. Each mapping was an MLP with three FC layers and neuron sizes of (32, 32, 3). The total number of trainable parameters for each inference model was 474K.

C.1 Human-data-based Neural Inference

For evaluating the human-data-based inference model, we employed a 5-fold cross-validation approach, training five distinct inference models using different sets of human data. In each training, data from 16 users were utilized, with each user performing 250 target-selection trials. The hyperparameters for training are detailed in Table C1.

C.2 Simulation-based Neural Inference

We utilized ~65,000 simulated trials as the training dataset for our simulation-based inference method. During the training, we randomly sampled user-specific parameters (Table 1) for each trial. The neural network architecture and training hyperparameters were consistent with those described in Section C and Table C1, respectively.

D RESULTS ANALYSIS OF STUDY 3

D.1 Effect of Target Size on Assistance Performance

For each results of Study 3A (*Wide* target grid) and Study 3B (*Dense*), we evaluated participants' task performance using a two-way repeated-measures ANOVA (*Inference Type* × *Target Size*) with Greenhouse–Geisser correction. In Study 3A, the results revealed significant effects of Target Size: smaller targets corresponded with prolonged completion times ($F_{1,19} = 146.38$, $p < 0.001$) and increased error rates ($F_{1,19} = 89.67$, $p < 0.001$). There were also notable interactions between Inference Type and Target Size for both completion time ($F_{3,57} = 65.62$, $p < 0.001$) and error rate ($F_{3,57} = 20.95$, $p < 0.001$). In Study 3B, similar trends were observed. There were significant effects of Target Size: smaller targets

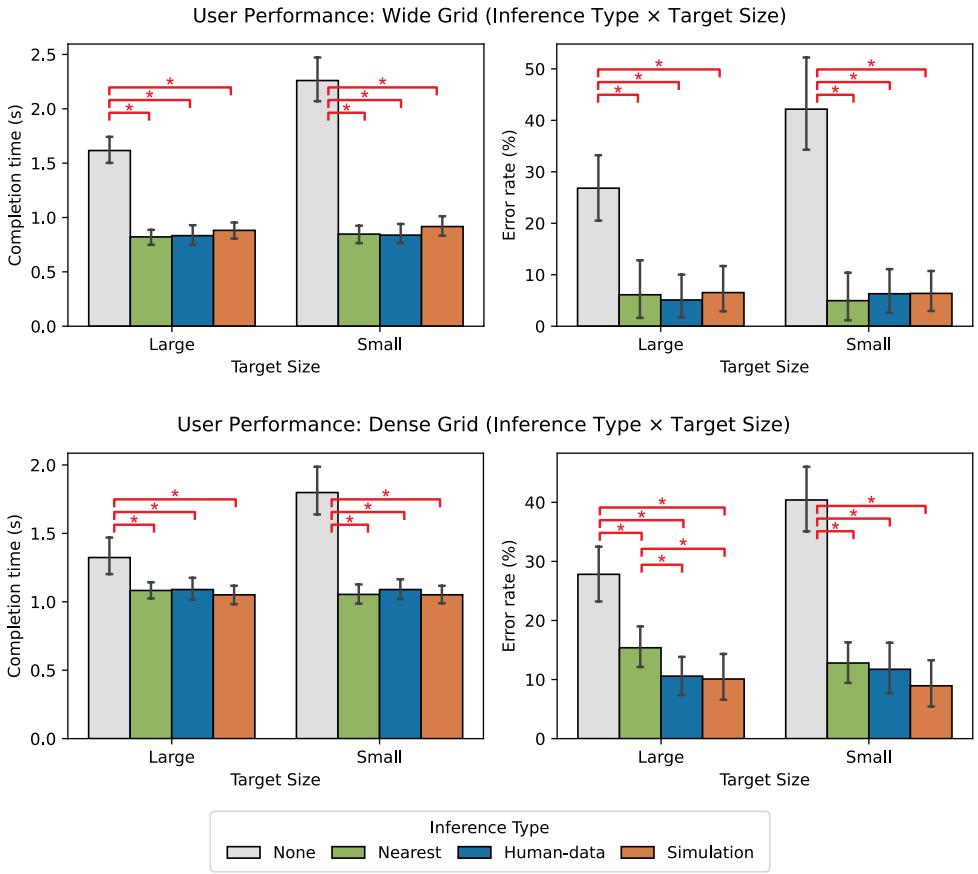

Fig. D1. Study 3: Participants' target-selection performance across different inference types and target sizes. An asterisk (*) indicates the statistically significant difference with $p < 0.05$ after adjustments using Bonferroni correction. Error bars denote 95% confidence intervals.

corresponded with prolonged completion times ($F_{1,19} = 21.95$, $p < 0.001$) and increased error rates ($F_{1,19} = 5.35$, $p = 0.032$). Significant interactions were found between Inference Type and Target Size for both completion time ($F_{3,57} = 30.65$, $p < 0.001$) and error rate ($F_{3,57} = 26.75$, $p < 0.001$). Subsequent post-hoc analyses with Bonferroni correction indicated significant differences across all combinations of Inference Type and Target Size. Figure D1 visualizes user performance across these variables, highlighting the combinations where differences were statistically significant at $p < 0.05$.

D.2 Effect of Time-based Auto-Click

In our extended examination of the auto-click mechanism, we carried out a two-way repeated-measures ANOVA (*With-or-without Auto-Click* × *Inference Type*) with Greenhouse–Geisser correction. The results highlighted significant effects from incorporating the auto-click mechanism, affecting both completion time ($F_{1,19} = 24.28$, $p < 0.001$) and error rate ($F_{1,19} = 41.69$, $p < 0.001$). Furthermore, significant interactions emerged between the employment of the auto-click mechanism and the inference type ($F_{3,57} = 11.28$, $p < 0.001$ for completion time; $F_{3,7} = 169.13$, $p < 0.001$

for error rate). Post-hoc pairwise comparisons, adjusted using the Bonferroni correction, confirmed the auto-click mechanism's marked enhancements in performance. Specifically, in situations without any inference (*None*), utilizing the time-based auto-click reduced both error rate and completion times ($p = 0.006$ for completion time; $p < 0.001$ for error rate). For Nearest Neighbor inference, its associated time-based auto-click substantially improved the error rate ($p < 0.001$), though it did not show significant change in completion time. A notable tradeoff emerged when comparing the confidence-driven auto-click performance of our Simulation-based Neural Inference to the that of time-based auto-click methods. Our inference method resulted in significantly quicker completion times than both None and Nearest Neighbor (all $p < 0.001$), while it resulted in significantly higher error rate than None ($p < 0.001$).

REFERENCES

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

[2] Manuel Glöckler, Michael Deistler, and Jakob H Macke. 2017. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*.

[3] Aleksi Ikkala, Florian Fischer, Markus Klar, Miroslav Bachinski, Arthur Fleig, Andrew Howes, Perttu Hämäläinen, Jörg Müller, Roderick Murray-Smith, and Antti Oulasvirta. 2022. Breathing life into biomechanical user models. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*. 1–14.

[4] 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.

[5] 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.

[6] 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*. 1–20.

[7] Eunji Park and Byungjoo Lee. 2020. An intermittent click planning model. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.

[8] 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).

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