Adapting User Interfaces with Model-based Reinforcement Learning

Kashyap Todi
kashyap.todi@gmail.com
Aalto University
Finland

Gilles Bailly
gilles.bailly@sorbonne-universite.fr
Sorbonne Université, CNRS, ISIR
France

Luis A. Leiva
luis.leiva@uni.lu
University of Luxembourg
Luxembourg

Antti Oulasvirta
antti.oulasvirta@aalto.fi
Aalto University
Finland

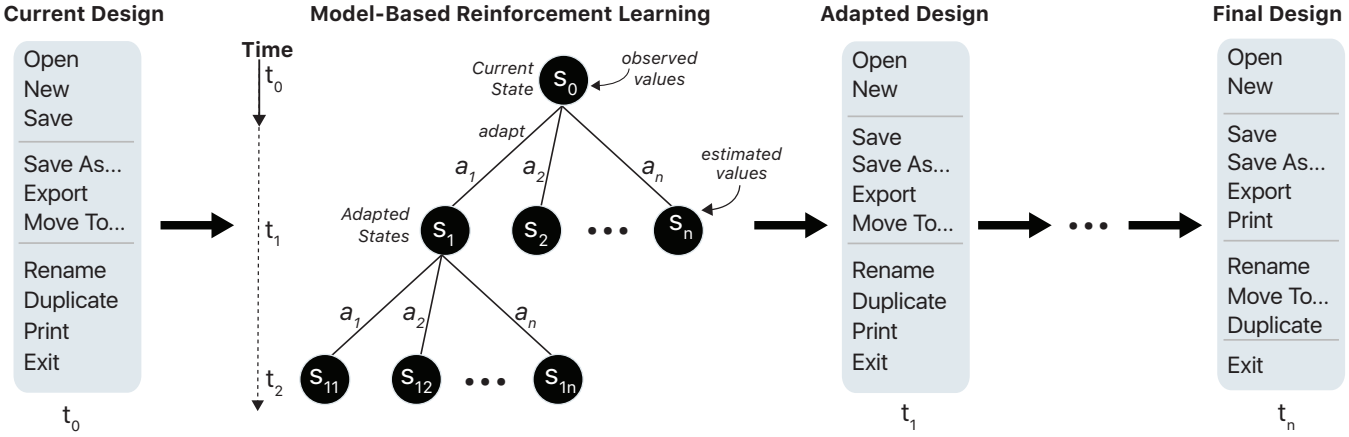

Figure 1: We present a model-based reinforcement learning approach for adaptive UIs that can improve usability while avoiding unexpected changes that surprise the user or require relearning. An interface is adapted by simulating several possible *sequences* of adaptations and evaluating them using predictive models in HCI. This approach avoids greedy, disadvantageous adaptations, and may anticipate possible user responses even with limited observation data.

ABSTRACT

Adapting an interface requires taking into account both the positive and negative effects that changes may have on the user. A carelessly picked adaptation may impose high costs to the user – for example, due to surprise or relearning effort – or "trap" the process to a suboptimal design immaturely. However, effects on users are hard to predict as they depend on factors that are latent and evolve over the course of interaction. We propose a novel approach for adaptive user interfaces that yields a conservative adaptation policy: It finds beneficial changes when there are such and avoids changes when there are none. Our model-based reinforcement learning method plans sequences of adaptations and consults predictive HCI models to estimate their effects. We present empirical and simulation results from the case of adaptive menus, showing that the method outperforms both a non-adaptive and a frequency-based policy.

CHI '21, May 8–13, 2021, Yokohama, Japan

ACM ISBN 978-1-4503-8096-6/21/05…$15.00
https://doi.org/10.1145/3411764.3445497

CCS CONCEPTS

• **Human-centered computing** → **Interactive systems and tools**.

KEYWORDS

Adaptive User Interfaces; Reinforcement Learning; Predictive Models; Monte Carlo Tree Search

ACM Reference Format:
Kashyap Todi, Gilles Bailly, Luis A. Leiva, and Antti Oulasvirta. 2021. Adapting User Interfaces with Model-based Reinforcement Learning. In *CHI Conference on Human Factors in Computing Systems (CHI '21), May 8–13, 2021, Yokohama, Japan.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3411764.3445497

1 INTRODUCTION

Adaptive user interfaces can autonomously change the content, layout, or style of an interface to improve their fit with the user's capabilities and interests. This paper looks at a foundational technical problem of adaptive interfaces that lies at the intersection of human–computer interaction and machine learning research: *How to select adaptations?* An adaptive system must decide *what* to adapt, and *when* – or *when not* – to make changes. Different computational approaches to this problem have been studied, among others, rule-based systems, heuristics, bandits, Bayesian optimisation, and supervised learning (see section 2). Although positive empirical results have been obtained (e.g., [4, 13, 17, 46, 49, 51, 52, 54]), known approaches have been criticised for being unpredictable and unreliable; they pick detrimental adaptations unacceptably often [16, 22, 33, 36, 56].

Estimation of utility is required for selecting an adaptation. Utility – in this case – refers to the usefulness of an adaptation to the user, or how it is perceived to benefit interaction when possible costs are taken into account. Picking an adaptation can be considered a hypothesis on how useful it is for user. Unfortunately, utility is very hard to estimate accurately – hard both at design time as well as interactively from the kind of data these systems have access to, such as clicks or viewing duration. In machine learning terms, utility is *latent*. Moreover, in adaptive interaction, utility is also *non-stationary*. That is, the skills and interests of the user evolve over time. A change that would make sense in the beginning when the user is novice with the design may be devastating for an experienced user.

We believe that adaptive systems could provide greater benefits by *planning sequences of adaptations* that gracefully lead a user through gradual changes. However, non-stationarity makes planning challenging: considering only a short period of time (i.e. a short horizon) can result in suboptimal designs. An adaptation that is overfit to a novice could be impossible to recover from later on when the user is more experienced. On the other hand, planning a long sequence of adaptations increases the size of search space, growing exponentially with the length of the planning horizon.

Model-based reinforcement learning is here developed as a principled and effective approach to these issues. We define the adaptation problem as a *stochastic sequential decision problem* [7], where the adaptive system should plan a sequence of adaptations over a long horizon. Reinforcement learning (RL) is a class of machine learning methods appropriate for this type of problems. Typically, a policy is learnt via trial-and-error that maximises future cumulative utility. Our RL approach is *model-based* as it uses predictive HCI models to estimate utility. These models simulate consequences – benefits and costs – of possible adaptation sequences without actually executing them [28, 50]. However, their application is conditioned on their accuracy: They should accurately predict short-term costs such as due to relearning as well as longer-term improvements to performance. Generally, such models are available at least in the areas of pointing, menu interaction, and graphical layouts. In comparison with an alternative, the *model-free* approach, the model-based approach requires less training and better generalises across conditions [28, 50]. However, finding the best adaption – by assessing the value of each sequence of adaptations with predictive models – is computationally costly, especially when considering sequences of changes over a long horizon. To solve this computational problem in an online setting, we use a combination of Monte Carlo Tree Search (MCTS) for planning, and deep learning to boost performance. To avoid extensive trial-and-error with users in the loop, our deep learning models are trained offline using HCI models.

Our general approach can be applied for various HCI applications, such as adaptive mobile homescreens, graphical layouts, and application menus. We demonstrate it specifically in the context of *adaptive menus*. The task is to adapt the interface by changing the arrangement and grouping of menu items (Figure 1), thus improving the menu's usability. We exploit and extend multiple menu search models from literature to estimate the upper and lower bounds of the value of an adaptation as well as their change as the user learns. This helps us form an adaptation policy that accounts for different user strategies and avoids adaptations that incur high costs to users. Our technical evaluation shows that our solution can tackle realistic problem sizes, and find favourable adaptations, on a commodity computer. Finally, we present results from an empirical evaluation where the approach compared favourably against a non-adaptive baseline design and a frequency-based adaptation policy.

To sum up, this paper makes three key contributions:

(1) A new formulation for adaptive interfaces that formalises them as a stochastic sequential decision-making problem;
(2) Development of model-based reinforcement learning for planning adaptations in the case where users learn;
(3) Application in adaptive menus with demonstrated benefits to usability.

2 PREVIOUS WORK: MACHINE LEARNING METHODS FOR ADAPTIVE INTERFACES

Our work contributes to methods for adaptive interfaces designed to operate autonomously, that is without explicit feedback or training samples from user. The core computational problem we review here is how to pick an adaptation; we do not cover issues like prior elicitation, explainability, nor the design space of intelligent interaction techniques.

2.1 Rules, heuristics, and logic

Early work on this topic studied rule systems, heuristics, and logic as the basis of deciding what to adapt (see, e.g., [44]). For example, in menu-based interaction, most systems still follow a heuristic approach, where adaptation is picked based on hand-written heuristics that exploit click frequency, visit duration, recency or other specific features that can be computed from observation data [21]. These approaches, in general, are feasible only when sensed input is highly predictive of the most appropriate adaptation. Another limitation is that writing a comprehensive and accurate rule system requires plenty of foresight. A rule system must be developed that, on the one hand, covers conceivable conditions the system can enter and, on the other, can graciously resolve conflicts when multiple rules apply.

2.2 Machine learning

The prevailing understanding is that *learning* is a key capability for adaptive systems [32]. Two learning capabilities are needed: *(1) inference*, the capability to update assumptions about the user based on observations; and *(2) decision-making*, the capability to choose appropriate adaptation in the light of assumptions about the user [41]. The two challenges can be relaxed, for example if user state is trivially known, or if the state is highly predictive of appropriate adaptation. In the latter case, the problem can be approached as a *supervised learning* problem, where a mapping is learned between user data and suitable adaptations. While this approach has been successful for input techniques, such as gesture recognition [57], it is an open question if this scales up to adaptive interfaces, which must not only learn user state but flexibly decide how to intervene in the user interface. A practical obstacle is how to obtain a dataset that describes the consequences of possible adaptations on possible users.

In the rest of the section, we focus on the general case, where both inference and decision-making are required and non-trivial.

2.3 Bandit systems and Bayesian optimisation

Bandit systems are one of the most successful probabilistic approaches to this problem, not only for recommendation systems but also for interface design and adaptation [38]. Each adaptation is modelled as an 'arm' associated with a distribution describing expected gains. Given prior data and new evidence on the measured success of an adaptation, bandits use Bayes theorem to update expectation. Importantly, a principled solution is offered to the exploration/exploitation problem. Methods like Thompson sampling can optimally balance between exploring actions, to learn about which actions work, and exploitation, to converge to good designs.

Bayesian optimisation generalises bandit systems to the case of multiple interrelated decision variables [47]. It is a global optimisation method that tries to find optimal adaptation by probing to a black box function; here, the user. It is a robust and sample-efficient and well-suited for noisy, expensive-to-evaluate functions. The method uses a *surrogate model* for approximating the model fit across the parameter space and quantify uncertainty. This is necessary for the acquisition rule to address the exploration–exploitation trade-off. This way, it is possible to learn adaptive responses via trial and error. Applications have been shown in human-in-the-loop design of interface features [15] and adaptation of low-dimensional design features [30]. However, while bandits and BO have been successful in simpler adaptation problems, like recommendations and calibration of interface parameters, their intrinsic limitation is that they are myopic; that is, they do not plan over a series of changes – a capability we need in adaptive interaction.

2.4 Reinforcement learning

Unlike bandits, reinforcement learning permits learning policies for *sequences* of actions where rewards are not immediately achievable. While applications have been shown for example in crowdsourcing [14], dialogue systems [58], and gaze-based interactions [20], a known limitation with the prevailing model-free RL approach is still the extensive amount of poor attempts (or trials) that are required to learn a good policy [50] This makes it poorly suitable for situations with very large state-action spaces.

Model-based RL uses a predictive model to simulate possibilities without first trying them out, which is useful for adaptive interfaces, because it significantly improves the efficiency of finding good solutions [28]. A policy can be determined with much fewer trials, and if the model is good, at times directly. Outside of user interfaces, we find applications of model-based RL, for example in board games, robots, video games [29], as well as behaviour-change applications, such as in generating behavioural instructions for people with dementia [23]. A grand obstacle for applications in adaptive systems is the model: where to get a good one? A related problem is that of *drift*: prediction errors can have a compounding, cumulative effect on planning performance. In HCI, although previous work has explored the use of model-free RL (e.g., see [35]), model-based RL has not been explored in adaptive interfaces at large, as far as we know. Also, while predictive models have been used for one-shot design generation [42], design space exploration [53], and for selecting a single action in a myopic manner [27, 51], they have not been used for simulation-based planning in an adaptive system.

In this paper, we approach the fundamental problem of selecting user interface adaptations by applying model-based RL, and exploiting predictive HCI models, to simulate and plan adaptations.

3 PROBLEM FORMULATION

We formulate the problem of adaptation as a *stochastic sequential decision-making problem* [7]. The adaptive system must *decide* what to adapt, if anything, given its observations. It should pick a *sequence* of adaptations in order to maximise their expected value to user over a longer window of interactions. For example, in our application example later on, we optimise for performance improvements in menu selection tasks achievable over multiple sessions of interactions. Further, in a *stochastic* problem, the world is neither fully known nor under the control of the system. In our case, while the system can change the interface, it cannot change the human, which has its own latent processes. For example, humans learn and change interests. This complicates the problem: Any greedily chosen adaptation may lead to irreversibly poor interactions later on. Thus, adaptations must be picked with a horizon of such developments in mind.

In the following, we formulate this problem as a *Markov decision process* (MDP). The benefit of an MDP formulation is that it offers a rigorous and actionable understanding of the problem. In particular, it (1) illuminates the decision problem, (2) links it to a body of theoretical results and practical approaches in AI and ML research, and (3) points toward appropriate algorithmic solutions.

The problem of adaptive interfaces is that of maximising expected cumulative discounted rewards $r(s_t, a_t)$ from acting according to an optimal policy π^* (see e.g. [26]):

$$\pi^* = \underset{\substack{\pi \\ p(s_{t+1}|s_t,a_t)}}{\operatorname{argmax}} \; \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^{\,t} r(s_t, a_t) \right] \tag{1}$$

where,

- $s \in S$ is a state of interaction; This consists of both the interface design (d) and the user (u)

- $a \in A$ is an adaptation; i.e. a change that can be carried out on the interface.
- p is a transition function; it provides the probability of transitioning from state s to state s_{t+1} after performing adaptation a; i.e., $p(s_{t+1}|s_t, a_t)$.
- r is a reward collected for making adaptation a in state s.
- γ is a discount factor controlling for how much to favour immediate (small γ) vs. long-term (large γ) reward.

Consider the case of adapting the homescreen layout of smartphones, consisting of a grid of application icons. Here, a state s of the system consists of both the homescreen design d and latent state of the user u who is interacting with the device. More specifically, the design d can encapsulate factors such as the arrangement of icons, their grouping or relationship to other icons, and other relevant features. With regard to the user u, aspects such as expertise, interests, and abilities, can be considered.

Given an initial homescreen design, with which the user has interacted, an adaptation a would result in a new design by, for example, changing the layout or ordering of icons. Upon adaptation, the transition function p specifies how the internal state of the user (e.g. their expertise) changes along with the external design state. The reward r then signifies the benefit an adaptation provides to the user by improving future interactions, for example, by reducing the time required to select an icon. Finally, the discount factor γ indicates to the adaptive system how immediate benefits and long-term improvements contribute towards the reward. Given this setting, the goal of the system is to find a suitable policy τ that can be used to select adaptations that maximise the estimated cumulative reward.

Finding a policy to select adaptations can be challenging for adaptive interfaces. While the true state of the design is fully observable, the system does not have access to the true state of the user. We can only estimate it through HCI models, where predictions provide us a *belief* about the user status (for the theoretical implications of this, see [2]). Further, due to the lack of user feedback, computing reward is not straightforward. Instead, predictive HCI models can be used to build objective functions related to performance and re-learning costs. Note that this formulation does not take a stance on what is the 'right' objective function. Finally, at any given state, there is a large number of possible adaptations that can change the design. When considering a sequence of adaptations over a long horizon, the state space grows exponentially. In the following, we describe how these challenges are addressed via model-based RL.

4 METHOD: DEEP MODEL-BASED REINFORCEMENT LEARNING

The core of this approach considers planning: the selection of a sequence of adaptation with the goal of maximising utility to user. Planning algorithms such as minimax and A-star, among others, utilise a tree representation of the search space. They have found several applications (e.g. game-playing [8], circuit-routing [11], etc.). The tree consists of nodes, connected by branches, representing valid states and transitions between them. However, classic tree search algorithms often require expansion of the entire tree. This is computationally expensive given the large number of possible adaptations (breadth) and long horizons (depth) in adaptive interfaces.

Monte Carlo Tree Search (MCTS) has been successfully employed in various game-playing applications to plan a sequence of moves efficiently (see [9]). MCTS can operate under uncertainty by analysing the most promising moves, and expanding tree nodes using random sampling based approaches. A well-known, inspiring application of MCTS is in AlphaGo [48], the computer program capable of playing the game of Go competitively against human players. A key insight here has been to incorporate neural networks to help predict which branches have highest expected value, and thereby deal with larger problem instances. In our work, we incorporate a value neural network to compute longer sequence cases faster.

4.1 Planning with MCTS

In contrast to game-playing applications, where a win/loss defines the terminal state, our case does not have a well-defined horizon. The user could keep interacting for years. We thus need to estimate a cumulative reward over a horizon of reasonable length. In other parts, our solution follows standard implementations of MCTS (Figure 2):

1. Selection: When a user concludes an interaction session, the root state s_0 is given by the current design d_0 and user observations u_0. A state $s_j \in S$ is selected via Upper Confidence Trees (UCT), a widely used estimator in model-based planning [31]. A key feature is that it has a coefficient C to balance between exploration and exploitation when evaluating several possible UI adaptations:

$$\text{UCT} = \frac{r_j}{n_j} + C\sqrt{\frac{\ln n_i}{n_j}} \tag{2}$$

where r_j is the total reward for child state s_j, n_j is the number of times s_j has been visited, and n_i is the number of times parent state s_i has been visited. Exploration constant C in our application is set to $1/\sqrt{2}$ following convention. If all adaptations from the selected state s_j have been previously explored, then selection is repeated until a leaf state is selected with unexplored adaptations.

2. Expansion: The selected node s_j is next expanded by picking an adaptation $a \in A$ that results in a new state s_{j+1}. At this point, $n_{j+1}, r_{j+1} = 0$. In our application, we further assume that the expanded state is either visible or invisible to the user. Invisibility can be exploited to plan multiple adaptations in a single turn.

3. Roll-outs and simulations: During one *roll-out*, as the tree has no value estimates to inform the selection of consequential states, adaptations $\{a_0, \ldots, a_N\} \in A$ are chosen at random, and rewards are estimated using predictive HCI models. All models simulate what would happen with that adaptation sequence and return an estimate of values over the whole window. This is repeated for a fixed number of steps, given by the horizon H, and cumulative rewards are computed for each predictive model:

$$r_{j+1} = \sum_{k=j+2}^{H} r_k \tag{3}$$

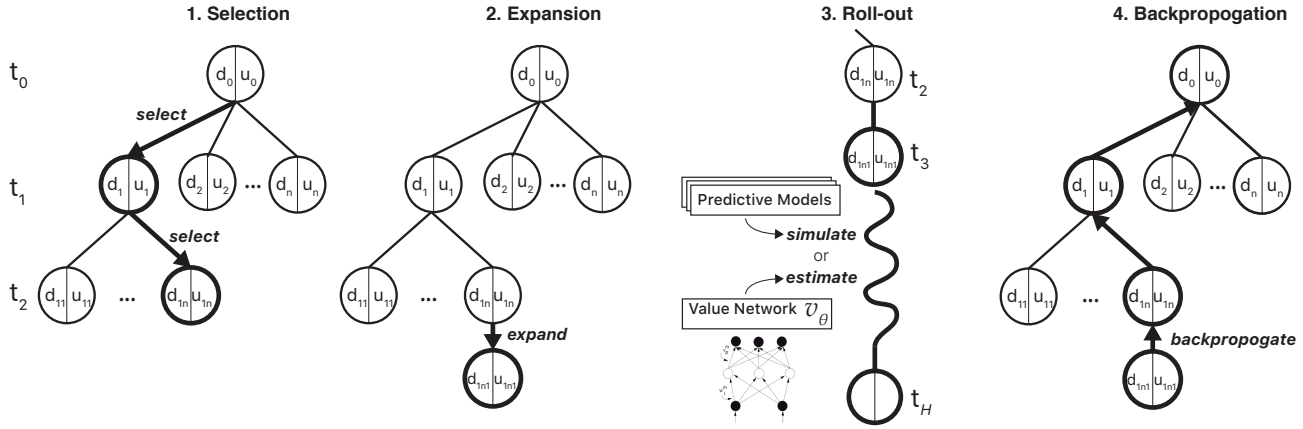

Figure 2: Model-based planning of adaptations using Monte-Carlo Tree Search (MCTS). Adaptations are selected using upper-confidence trees (UCT). After expanding a new node (adaptation), reward estimates are obtained through roll-outs, and back-propogated to the root node. The child with the highest value is picked as the next adaptation.

4. Backpropagation: At the end of a simulation, the cumulative reward r_{j+1} is backpropagated from the newly-expanded state s_{j+1} to the initial state s_0, and values (rewards r and visits n) are updated.

The above steps (1–4) are repeated several times to obtain value estimates for each adapted state.

Selecting the next adaptation: Given these value estimates, the system can now choose the best adaptation (by setting $C = 0$ in Equation 2) to maximise expected utility for the user while avoiding costly changes.

We support several ways for selecting adaptation that allows controlling for the trade-off between risk and gain. Our approach assumes that there are multiple predictive HCI models that bound the true behaviour, for example by offering best-case and (realistic) worst-case estimates. Alternatively, predictive models can be included to address multiple objectives, such as task completion time, cognitive load, and disruption [25]. Combining value estimations from all models, we can implement different objective functions, such as:

(1) *Average:* The mean of rewards from each model gives total reward r, thus accounting for varying user strategies.
(2) *Optimistic:* The system assumes optimal user strategy, and selects the model that maximises rewards.
(3) *Conservative:* The system ensures that no user strategy is harmed by selecting the lowest-possible reward, and penalising negative rewards.

4.2 Value Estimation with Deep Neural Networks

To address large problem sizes in online settings, where repeating a sufficient number of MCTS simulations to attain robust estimates is infeasible, we develop a deep neural network architecture that can efficiently provide predictions in real-time. In place of roll-outs, where simulations can be costly for longer horizons, a pre-trained *value* network is used to directly obtain value estimates for unexplored states.

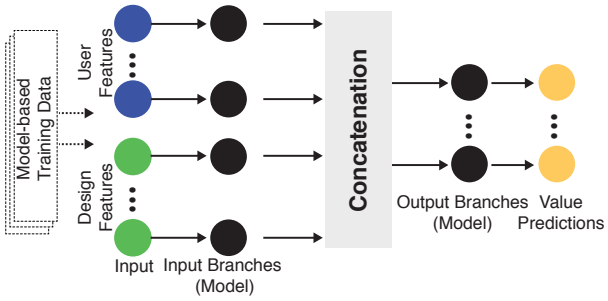

Figure 3: Neural network architecture for obtaining value estimates. Training data is generated using HCI models.

As illustrated in Figure 3, we propose an m-headed n-tailed architecture that is trained end-to-end with backpropagation. Each of the m input parameters is treated as an independent model branch (head) that is eventually concatenated and passed to n independent model branches (tails). During offline training, model-based data is generated using MCTS roll-outs from randomly sampled initial states. Value estimates, along with state (design and user) information, are given as input samples to a deep neural network model. Elementary design and user features are parameterised with the neural network model. This is then used to predict value estimates for any given state in an online setting. Note that we decouple value estimations from objective functions (see above), leaving it up to the adaptive system to decide how to use information from multiple models. The main advantages of using neural networks for estimation are that they have high learning capacity, and can evaluate thousands of states in real-time without running expensive simulations.

4.3 Applications in Adaptive Interfaces

With certain conditions we outline here, the approach we outline is broadly useful across applications of adaptive interfaces. For example, it can be used to adapt the structure of webpages layouts,

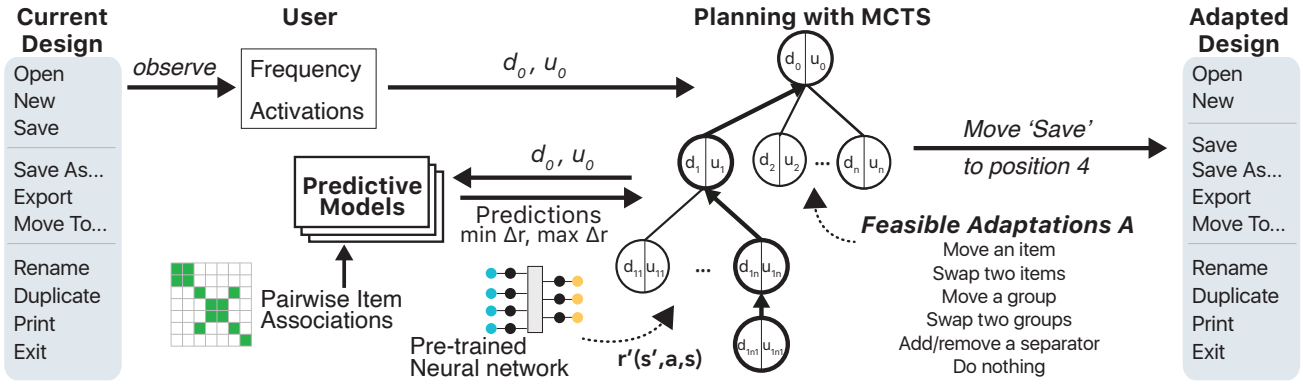

Figure 4: Menu adaptation with deep model-based RL. Given a current design and user observations, MCTS-based planning is used to select the next adaptation. Reward estimates are obtained either using predictive models or from a neural network.

arrange icons on mobile homescreens, or reorganise application menus. Depending on the application, the goal or objective can differ and might include minimising selection time, increasing saliency, reducing cognitive load, increasing engagement, or a combination of them. The approach can handle different types of adaptations including the presentation of the graphical elements (position, size, colour, etc.) or their behaviour (number of elements, animations, etc.).

For any application case, a core aspect of our approach relies on HCI models [42] to sufficiently accurately capture the effects of these adaptations on the chosen goals. For instance, Fitts' law [39] can be used to adapt the location and the size of an elements to minimise pointing time. Other practical limitations include the size of state–action space. In the following section, we demonstrate our approach with an application in adaptive menus, and present a novel HCI model to predict selection time in menus.

5 APPLICATION: ADAPTIVE MENUS

To demonstrate the applicability of our approach, we tackle a challenging and open question in the field of adaptive interfaces: *adaptive menus*. Menus have received extensive attention in HCI research because they are widely used, and adaptation has potential to improve usability [4, 55]. It is known that unexpected changes in menus can introduce a temporary performance drop, increase cognitive load, and potentially lead to the rejection of the adaption/techniques [4, 55]. No general solution has been proposed for autonomous adaptation that could not only move items to the top, but handle reorganisations more comprehensively. We provide a general solution here considering linear menus with up to 20 items, which covers a wide number of menus typically found in common operating systems, applications, and websites [3, 4, 6]. As menu adaptations, we only consider those modifying the position and grouping of items in the menus, leaving other presentation adaptations, such as highlighting and split menus, for future work. While we consider linear menus with textual labels, our solution can be extended to address the problem of adaptive homescreens (introduced in section 3) by extending the menu search model to consider two-dimensional grids and graphical icons.

5.1 Problem Definition

Following the general formulation in section 3, we first define the adaptive menu problem. Figure 4 presents an exemplary illustration.

State (S): A state $s \in S$ gives information about the menu design and the user.

We define a design $d \in D$ as a pair $< L, M >$ where L is a non-hierarchical linear menu [40] containing an ordered list of items. An item is either a word or a separator (used to create semantic groups). M is an association matrix defining the *semantic relatedness* between menu items. In our implementation, M is given by the designer as binary relationships by specifying lists of related items. For common words, it can be inferred using word embedding models [43].

The system observes user clicks on menu items to approximate a user's *expertise* and *interest*. We use a simplified version of the learning component from ACT-R [1] to compute *user expertise* for each menu item i_l in a menu with n items:

$$B(i_l) = \sum_{j=1}^{n} (T - T_{j,i_l})^{-\rho} \tag{4}$$

where $B(i_l)$ is the level of activation of item i at location l in the memory, T is the current time, T_{j,i_l} is the time of the j^{th} selection of i_l, and ρ is a decay parameter equal to 0.5. *User interest* (or prediction scheme [55]) is given by the frequency distribution of commands selected during the previous interaction session, containing N clicks. Additional statistical models of user interest and expertise [18, 19, 21, 51], can be plugged into our architecture.

Feasible Adaptations (A): The set of possible adaptations A, through which a menu can be reorganised, includes (1) moving a menu item to a certain position, (2) swapping two items, (3) adding or removing a separator, (4) moving an entire group, (5) swapping two groups, and (6) not making any changes.

Transition function (p): We use MCTS, where the probability of making an adaptation from state s_0 to s_1 is given by UCT (Equation 2). During planning, we balance between exploiting high-reward adaptations and exploring others. When selecting an adaptation a to make on the current design d_0, resulting in a new design d_1, a greedy strategy is chosen:

$$\underset{a}{\text{argmax}} \ \frac{r_j}{n_j} \tag{5}$$

Reward (r): We extend predictive HCI models of menu use to obtain reward estimates. A key feature of these models is to take into account the implicit cost of adaptations. Given a pair of menu designs and estimates of user expertise and interest, these models predict selection time for items for varying user strategies. For each model, the reward r then is the difference in average selection time, weighted by user interest. Rewards from multiple models can be combined using any of the objective functions given in section 4.1.

When an adapted state is assumed to be displayed (visible) to the user, we simulate an interaction session (new clicks) based on user interest, and update expertise accordingly. Conversely, the system can use invisible states to withhold presentation and combine multiple adaptations, such that the state is used only as a pathway to another state without being displayed. Here, no user updates are applied.

5.2 Models for Simulating Menu Search

Several predictive models explain how users search within linear menus [5, 10, 12, 24, 40]. We build on these models to define three search strategies, and use these to evaluate the utility of a menu design by simulating search tasks, illustrated in Figure 5:

(1) *Serial search*: The user searches serially, from top to bottom, until the desired item is found in the menu.
(2) *Foraging search*: Grouping of items is exploited such that the user only searches for the target within relevant groups.
(3) *Recall search*: The user relies on memory to search for the desired item at an expected location in the menu.

Our choice of the three models was made with the hypothesis that they would provide bounds for best-case and worst-case performance. In the beginning, best-case performance would be governed by foraging and serial search, but as experience accumulates, the (rational) user would shift to foraging based and recall-based strategies. But making the assumption that a user *might* – for reasons like lack of effort or interest – use a poor strategy allows us to define a conservative policy for adaptation that is unlikely to annoy them. We note that it is possible to plug in additional search strategies (e.g. Random search [40]) without modifying the general architecture of the algorithm.

Table 1: Key notations used in this section.

Notation	Description
i_l	Target item i at location l
$T_{\text{model}}(i)$	Search time for an item i with a given model
δ	Constant for cautious inspection cost of an item
T_c	Constant surprise penalty when an item is not found as expected
T_{trail}	Constant pointing time when the cursor trails eye gaze
$M(i_k, i_l)$	Boolean relationship between items i_k and i_l, from association matrix M
$B(i_l)$	Activation level for i at l

5.2.1 ***Serial search.*** When searching for a menu item i_e at an expected position e, serial search [5, 13, 40] consists of a top-to-bottom inspection of the menu, until the item is reached. Inspection (or reading) cost for any item i_l is given by:

$$T_{\text{read}}(i_l) = \frac{\delta}{1 + B(i_l)} \tag{6}$$

where $B(i_l)$ is the activation of i at l, and δ is the cautious inspection cost constant when there is no activation. The total serial search time for a target at expected location e is thus given by:

$$T_{\text{serial}}(i_e) = \sum_{j=1}^{e} T_{\text{read}}(i_j) + T_{\text{trail}} \tag{7}$$

Where T_{trail} is a constant pointing time assuming that the mouse cursor trails the eye-gaze (tracking strategy) during serial search [5, 10].

In an adapted menu, if the item's new location a is **before** the expected location e ($a \leqslant e$), the search time reduces following Equation 7. However, if a is **after** e ($a \geqslant e$), *surprise penalty* T_c is imposed upon not finding the item as expected. Following this, the user cautiously inspects the remaining items at a slower rate δ. The total search time is:

$$T_{\text{serial}}(i_a) = T_{\text{serial}}(i_e) + T_c + (a - e) \cdot \delta + T_{\text{pointing}} \tag{8}$$

To support serial search, it is advantageous for an adaptive system to move frequently-used items towards the top.

5.2.2 ***Foraging search.*** Here, semantic structure (grouping) is exploited to avoid wasting time inspecting groups that most likely do *not* contain the target item [4]. The *anchor*, or first element of a group, "signals" what is in the group. If the anchor is unrelated to the target, the user skips the group. If the anchor is related, the user performs a serial search within this group.

Consider a menu where n_g is the number of groups, $n_g(i_e)$ is the location of the group that contains the target (i_e), and e the expected target location within the group. $M(g_j, i_e) \in [0, 1]$ specifies if i_e is associated to an anchor g_j. In a well-organised menu, where the anchor of one of the groups is related to the target item, foraging search time is:

$$T_{\text{forage}}(i_e) = \sum_{j=1}^{n_g(i_e)} \left(T_{\text{read}}(g_j) + M(g_j, i_e) \cdot \sum_{k=1}^{\min(e(g_j), s(g_j))} T_{\text{read}}(i_k) \right) + T_{\text{trail}} \tag{9}$$

where $s(g_j)$ is the size of the group, and $e(g_j)$ is the location of the target item t_e within group g_j if it is located within the group, ∞ otherwise. Finally, T_{trail} remains the constant pointing time when the mouse cursor trails the eye gaze. Thus, items within related groups are inspected serially until the target is successfully found.

In a poorly organised menu, where the target item is not located within the expected group(s), a user first attempts foraging search by inspecting all anchors, and all items within related anchors (given by Equation 9). Upon not finding the item, a surprise penalty T_c is incurred, and the user reverts to serial search under caution from the top of the menu.

To support foraging search, it is desirable for an adaptive system to create groups of related items, or eliminate groups where items have no associations.

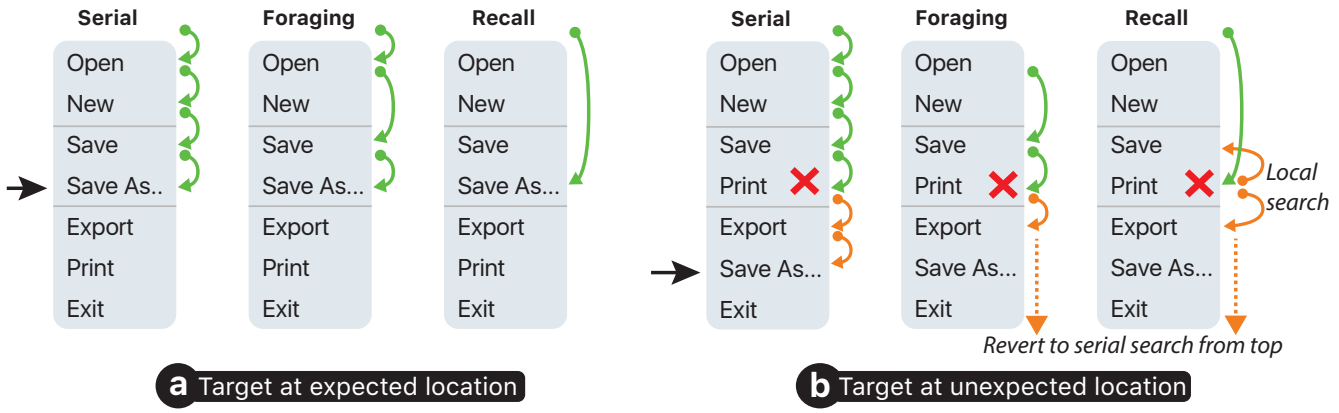

Figure 5: Model-based simulation of search for a target item (*'Save As...'*). (a) when the target is at an expected location, search proceeds as expected; (b) when the target is at an unexpected location, a penalty is imposed upon not finding the target at its expected location (indicated by ×), and search reverts to slower strategies.

5.2.3 ***Recall search***. Recall (direct) search [5] relies on user's memory, given by activations B, to directly glance at items without inspecting the entire menu. For a target item i, if there are no activations B_i above a threshold θ (we use 0.5), the user reverts to serial search (Equation 7). When $B(i_l) \geqslant \theta$ for a target i_l at location l, the user attempts recall search by inspecting the item at l (Equation 6).

If found at l, the user then performs a pointing task. Here, the visual search and pointing task are performed sequentially as eye movement is faster than mouse movement [5]. We use Fitts' law to estimate pointing time in menus:

$$T_{\text{pointing}}(i_l) = a_p + b_p \cdot \log(1 + i_l) \tag{10}$$

where $a_p = 10.3$ and $b_p = 4.8$ according to [5].

If not found at l, after incurring surprise penalty T_c, the user attempts *local search*, by randomly inspecting N_{local} items in the vicinity of location l.

$$T_{\text{local}}(i_l) = T_c + \delta \cdot N_{\text{local}} + T_{\text{trail}} \tag{11}$$

In non-ordered menus, N_{local} is equal to 2 times the number of items in the Fovea. In semantic menus, N_{local} is the number of items in the group.

In an adaptive menu, the target item i might have been encountered at several locations. Here, the user attempts recall search at all locations $l \in L$ where $B(i_l) \geqslant \theta$, until the target is found. The total recall search time for target i_a at adapted location a is given by:

$$T_{\text{recall}}(i_a) = T_{\text{pointing}}(i_a) + \sum_{l \in L: B(i_l) \geqslant \theta}^{l \pm N/2} T_{\text{read}}(i_l) + T_{\text{local}}(i_l) \tag{12}$$

If recall search fails ($a \notin L$ or $B(t, a) < \theta$), the user eventually reverts to serial search under caution (Equation 8).

To support recall, it is advantageous for an adaptive system to place frequently-encountered items at locations where they have been seen before.

5.3 Neural Network for Rewards Estimation

The above search models enable us to predict selection times for varying user strategies. By simulating consequences of adaptations during roll-outs, we can estimate implications of design changes on user performance. However, given the varying length of menu sizes, running simulations with long horizons can be infeasible for online settings. For example, for a menu with 15 items, and up to 8 separators, there are over 500 feasible adaptations. To address large problem sizes, we instantiate our general network architecture (Figure 3) for adaptive menus. The key idea is to anticipate the rewards for a given menu adaptation taking into account the previous menu design and the user behaviour. The model inputs are: (1) design head: adapted menu design, association matrices of the current and adapted menu; (2) user head: previous and current clicks distribution. The model outputs reward predictions for each of the three search models: serial, foraging, and recall. Figure 6 illustrates the model architecture.

Each input is treated as an independent model branch (head) that is eventually concatenated and passed to three independent model branches (tails), one for each output reward. Each item in the adapted menu is converted to a one-hot encoded vector, flattened, and then passed to a fully connected layer. The association matrices are diffed and passed to a long short-term memory (LSTM) layer, then passed to a fully connected layer. Finally, the click history is passed to an LSTM layer, which models sequential data and is

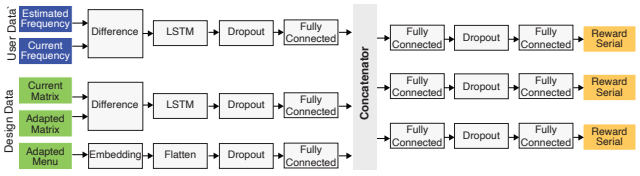

Figure 6: Our value network architecture takes menu design and user features as input to provide individual reward predictions.

designed to handle long-term dependencies, and is then passed to a fully connected layer.

The concatenated inputs are passed to each network tail, which comprises two stacked fully connected layers. At the end of each tail, we use linear activation to predict each reward, since they are not bounded. For regularisation purposes, our architecture uses Dropout layers with drop rate 0.5 before each of the fully connected layers. This prevents overfitting the model to the training data, improving generalisability to unseen data. The loss function for all model tails is the mean squared error (MSE), which is computed as the average of the squared differences between the predicted and the actual values, which penalises large errors. We use the RMSProp optimiser, a popular stochastic gradient descent algorithm with momentums. We use learning rate $\eta = 0.001$ and decay factor $\beta = 0.9$ for the optimiser. We train the model for 200 epochs at most, using early stopping (10 epochs patience) to retain the best model weights, and monitor its performance on a validation set comprising 20% of training data. After training, our model achieved remarkable performance: $\text{MSE}_{serial} = 0.149$, $\text{MSE}_{forage} = 0.408$, $\text{MSE}_{recall} = 0.431$.

6 EVALUATION

We validate our method, applied to adaptive menus, through technical and empirical evaluations.

6.1 Technical Evaluation

We conducted a technical evaluation with realistic and challenging scenarios, where the adaptive system must adapt menus for simulated users. The two main questions we seek to answer are:

(1) *Can a model-based planning approach successfully and consistently improve predicted usability?*
(2) *Does our neural network based solution scale it up to address larger problem sizes?*

6.1.1 Tasks.

Menu Designs and User Interest: We considered 3 *menu sizes* — 5, 10, and 15 items — to address varying cases, from short contextual menus to longer application menus. In addition, up to 8 separators were allowed for grouping, resulting in menus with up to 23 items. We picked common labels for menu items, where categories specified pairwise associations (e.g. animals, furniture, vegetables, clothing). For each menu size, we created 4 starting menu designs by randomly assigning labels to item positions. We used a Zipfian distribution to reasonably model menu usage [13, 37]. We sampled 8 different click histories by randomly assigning frequency to item labels. This resulted in 3×8×4 = 96 configurations, each assigned to a different simulated user.

Reward Estimation: We compare two methods of estimating rewards: *model-based simulations* and *value neural network predictions*. With model-based simulations, predictive models are used during roll-outs to estimate rewards for each state. With value network predictions, our pre-trained network models are used to predict value estimates for each state.

6.1.2 Implementation.

Our MCTS-based planning algorithm, and the predictive menu models, are implemented in Python 3.7. The value neural network is implemented with TensorFlow 2.0.0. We used a GNU/Linux server with an Intel Xeon Gold CPU @ 2.10 GHz (64 bit processor) for simulations. The execution of the study was automated such that trials were conducted sequentially, to avoid variations in computational resource usage.

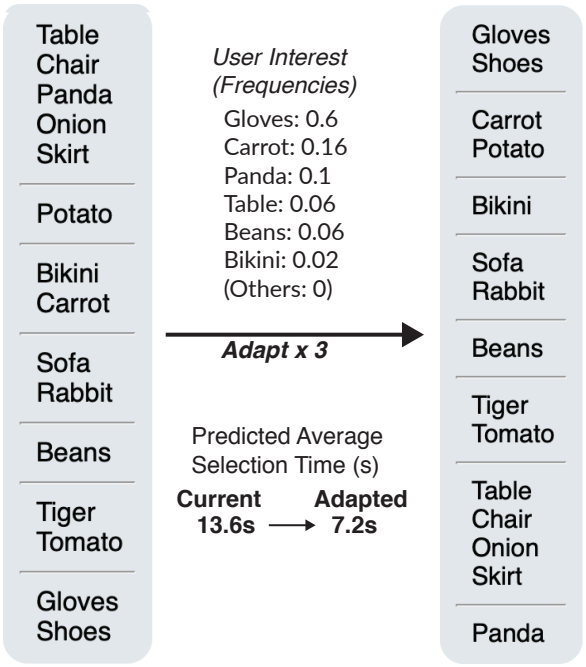

Figure 7: A sample result from the simulation study, for a 15-item menu design. In 3 steps, the menu was adapted to better suit the given user's interests ('Gloves' group moved to the top), and improved some grouping ('Carrot' with 'Potato') as well.

During each trial, a combination of {menu size × user history × menu design × objective function × reward source} was selected, and given as input to the system. In a constrained setting, the MCTS algorithm was allowed 400 iterations, and a shallow roll-out horizon of 4 steps, to build the search tree and find suitable adaptations.

6.1.3 Result: Success Rate.

With the above setup, we first evaluated whether our approach, and implementation, could successfully identify promising adaptations. As dependent variable, we measured *success rate* of finding beneficial adaptations. We define a successful trial as one where the predicted selection time is improved by adaptation. Figure 7 shows an example result for a challenging case with a 15-item menu.

The overall success rate with model-based simulation was 92.7%, indicating that in most cases an improvement was found. Similarly, with the value network, success rate was 89.6%. These results support our approach towards planning menu adaptations that can improve user performance.

6.1.4 Result: Scalability.

To assess the scalability of our solution, we compared computation time for model-based simulations vs. value network predictions. In addition to the horizon of 4 steps used to evaluate success rate, we evaluated 3 other search depths –

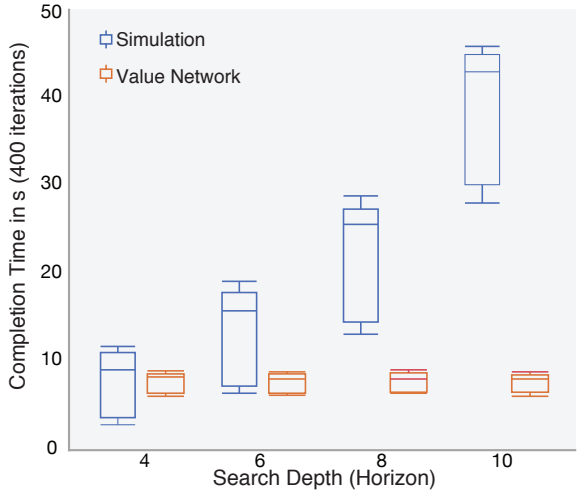

Figure 8: Computation time for planning adaptations via model-based simulations vs. value network for varying search depths. Our value network exhibited consistent performance for longer horizons.

6, 8, and 10 – depicting a range of planning horizons, from short sequences to longer sequences.

Figure 8 illustrates computation time results for each depth level (for 400 MCTS iterations). We observe that for depths ≤ 4, the value network does not provide much benefit. However, as search depth increases, while the computation time with our value network remains constant (mean M = 7.77s, SD = 1.0), it drastically increases with simulations (from M = 7.9s, SD = 3.5s at depth 4 to M = 39.0s, SD = 7.4 at depth 10).

6.2 Empirical Evaluation

The primary goal of this evaluation is to test whether our planning approach (henceforth MCTS) applied to linear menus improves performance in comparison with static menus (Static), and with the well-known frequency-based adaptive approach (Frequency) as a baseline (e.g. as in [34]). In MCTS, the menu adapts after each block by planning adaptations; in Static, the menu does not adapt over time; in Frequency, the menu adapts based on the frequency of clicks on menu items. To this end, we conducted a lab study where participants completed selection tasks in a *within-subject design* with three conditions (Static, Frequency, MCTS).

6.2.1 Materials. For the experiment, linear menus with 15 item labels were randomly generated. Items labels were selected from common categories (e.g. animals, fruits, countries, etc.) to avoid prior biases. For each participant, two menus were generated for each of the three conditions, resulting in six unique menus. To avoid confusion, there were no overlaps in item labels or categories between the menus for a participant.

For each menu within a condition, a Zipfian distribution, known to accurately capture real-world command selections [13, 37], with shape s = 1.5 was used to control the frequency distribution of target items. The same frequency distribution was used for all three conditions within a participant. Unique frequency distributions were generated for every participant, to consider variance in user interests. These frequency distributions were then used to generate sequences of target items, to be presented as stimulus during the trials.

6.2.2 Participants. 18 participants (10 masculine, 8 feminine, 0 others), aged 18 to 38 (mean 27.2), with varying educational backgrounds, were opportunistically recruited. All participants reported frequent desktop or mobile, and web usage. Participation was compensated with a movie ticket voucher (approx. €12.00).

6.2.3 Apparatus. The experiment was conducted on a Macbook Pro, with a 15" Retina display. An Apple Magic Mouse with default tracking speed was used for selection tasks. The study interface was implemented using HTML and Javascript, and was displayed in a browser window. Timestamped cursor movements and clicks on menu items were recorded.

6.2.4 Stimulus and Task. The target item name was displayed at the top of the browser window. Participants began a trial by clicking on a confirm button, upon which the menu was displayed directly below. Errors were logged, and participants had to select the target item to finish the trial. Upon clicking the target item, the menu was hidden, and a short break was provided.

6.2.5 Procedure and design. The experiment began with an introductory briefing and participant consent. In a within-subject experimental design, each participant tested the three conditions (Static, Frequency, MCTS) sequentially. Condition order was counterbalanced between participants using a 3×3 Latin square.

During each condition, the participant interacted with two different menus during 3 blocks. Within a block, the two menus appeared in an alternating order, separated by short breaks (3 seconds). For each menu, 20 selection tasks (trials) were completed. We introduced this design to reflect the fact that (1) users perform several sessions of work in the real life (one session == 1 block), (2) users regularly switch between applications within a session, and thus use different menus [45]: we wanted to avoid undesirable effects due to repetitive selection within a single menu, and (3) each menu has a different selection frequency, given by two Zipfian distributions.

Participants took mandatory breaks (1 minute) between two consecutive blocks, and longer breaks (5 minutes) between conditions where they answered open-ended interview questions. In summary, the design is: 18 participants × 3 conditions × 3 blocks × 2 menus × 20 selections = 6480 trials.

6.2.6 Quantitative Results.

Average Selection Time: We created a mixed-effect model for repeated measures analysis of variance (one-way ANOVA) with selection time (in ms) as dependent variable, conditions (Static, Frequency, MCTS) as fixed independent variable, and participant ID and menu design as random variables.

Condition had a statistically significant effect on selection time $F(2, 17) = 5.47$, $p < 0.05$, with grand means Static = 2283 ms, Frequency = 2298 ms, and MCTS = 2162 ms (Figure 9a). Post-hoc test using Tukey HSD revealed that MCTS (2162 ms) was significantly faster (lower selection time) than both Frequency (2298 ms) and Static (2283 ms); the difference between Static and Frequency was not statistically significant.

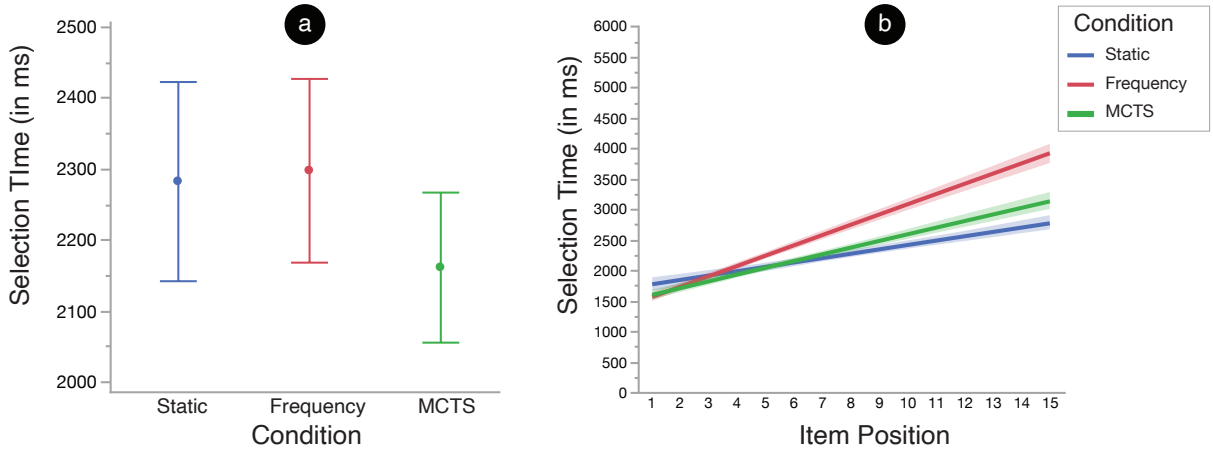

Figure 9: (a) The MCTS condition was associated with significantly lower selection time as compared to the two baselines. Vertical bars indicate 95% confidence intervals. (b) For items positioned lower in the menu, selection time in FREQUENCY increased more drastically than other conditions.

Target Item Position: Given the menu selection scenario, items near the top of the menu are typically faster to select than items that are near the bottom. However, this selection time depends not only on the cursor movement distance, but also the user's ability to search for items in the menu. To get a better understanding of how the different conditions influenced performance, we further looked at how target item positions in the menu influence selection time. Figure 9b illustrates the linear increase in selection time with target position for the three conditions. It can be observed that while selection time for the top-most items (lower target positions) is quite similar for the three conditions, with MCTS being the fastest, it increases more drastically for the FREQUENCY condition, as compared to STATIC and MCTS. When we exclude the top-three target items, the difference in selection time between MCTS (mean = 2454 ms) and FREQUENCY (mean = 2799 ms) is 344 ms (i.e. FREQUENCY is about 15% slower).

6.2.7 Qualitative Results. During the study, participants were not informed about adaptations (if any) in advance. After each block, we asked them whether they noticed changes to menus during use, and their opinions about these changes (if any). 15 participants commented that they noticed changes in the FREQUENCY condition, but only 2 participants noticed *how* these changes were occurring. Participants commented that the reordering was confusing, and prevented them from remembering item locations: *"I might skip down instead of checking the top, and then go back to the top."* (P3). In the STATIC condition, participants could use their memory to directly access some items, but commented on the lack of proper grouping: *"the items were not consistent in their grouping, and they were not intuitively grouped"* (P11). In the MCTS condition, participants noticed that the categorisation of items into groups improved upon adaptation, and helped them in searching for related items: *"the items were organised under categories often – that helped select items"* (P18).

6.3 Summary

Results from our simulation-based evaluation offer evidence for our approach and technical solutions. First, MCTS-based planning consistently proposes adaptations that could improve predicted performance (Figure 9a). Second, as search depth increases, our results indicate that value network is more efficient for estimating reward predictions. Further, results from our user study highlight benefits over a static and an adaptive baseline. Through model-based planning, we can adapt menus that improve overall performance, as given by reduced selection time. A common pitfall of the frequency-based approach is that, although it can improve performance for commonly-used items or commands, it prevents recall and makes selection of other items increasingly difficult. In contrast, we observed (Figure 9b) that adaptations made through our approach could provide these benefits while avoiding costly changes that require relearning and cause annoyance.

7 CONCLUSION

We have presented a model-based reinforcement learning method suitable for adaptive interactions. While recent successes of reinforcement learning have created renewed enthusiasm toward this

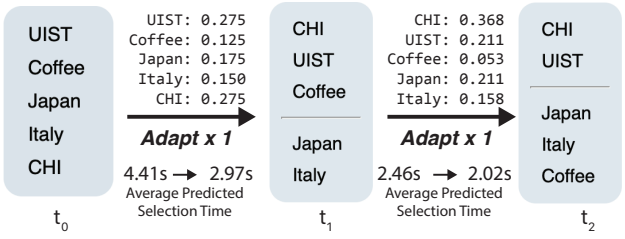

Figure 10: Sequential adaptation of a 5-item menu. The system avoids greedily moving 'Coffee' (lowest frequency) at the first step. In light of new observations, this change is justified by the reduced user interest.

approach in HCI, applications to adaptive user interfaces have remained scarce. To apply this class of machine learning methods for selecting adaptations, we have proposed the use of predictive models in HCI for value estimation during planning. We have presented solutions to several consequent technical challenges, most notably:

- How to model the decision problem in adaptive interfaces for model-based RL;
- How to estimate MCTS roll-outs using HCI models;
- How to design deep neural networks to boost planning.

To study and demonstrate the viability of the approach, we have applied it to the challenging case of adaptive menus by extending predictive models. Our simulated and empirical evaluations suggest significant and practically valuable improvements to usability. Importantly, the adaptive system does not require explicit user input, but is still able to perform conservatively without disadvantageous or annoying changes (Figure 10). Our empirical evaluation reveals that this approach can work even when starting from poor designs that would be hard to recover with approaches that do not consider planning. Future adaptive applications can benefit from our general approach by exploiting and extending predictive models of interaction.

Limitations and Future Work

We see several exciting topics for future research on model-based RL and its applications in HCI. First, one limitation to the applicability of the approach is the requirement for accurate models of short- and long-term consequences of adaptations. So far we have assumed that such models are expressed as step-by-step computer simulations or mathematical models. However, there is no reason why data-driven models (e.g. [59]) – trained on larger datasets of user data – could not be used for this purpose, significantly expanding the scope of possible applications. Second, algorithm engineering is needed to deploy this approach to larger applications. In particular, presently, with our computing resources, problem sizes of up to 20 items are still within reach in the case of menu systems. To improve performance beyond that, techniques for GPU computation and more efficient training will need attention. Finally, while our work successfully used a value network, further improvements can be expected by implementing a policy network [48].

To conclude, we hope our work can be broadly appealing, and invite contributions from both the HCI and the machine learning community. At the core of model-based RL is an understanding of how users behave and what makes a good design in given conditions. We believe that future applications can benefit from this approach to improve interactions.

8 OPEN SCIENCE

We support adoption and further research efforts by providing an open code repository, with examples and instructions, on our project page: https://userinterfaces.aalto.fi/adaptive.

ACKNOWLEDGMENTS

We thank all study participants for their time, and colleagues and reviewers for their helpful comments. This work was funded by the Department of Communications and Networking (Comnet), the Finnish Center for Artificial Intelligence (FCAI), Academy of Finland projects 'Human Automata' and 'BAD', Agence Nationale de la Recherche (grant number ANR-16-CE33-0023), and HumaneAI Net (H2020 ICT 48 Network of Centers of Excellence).

REFERENCES

[1] John R Anderson and Christian J Lebiere. 2014. *The atomic components of thought.* Psychology Press. https://doi.org/10.4324/9781315805696
[2] Mauricio Araya, Olivier Buffet, Vincent Thomas, and Françcois Charpillet. 2010. A POMDP Extension with Belief-dependent Rewards. In *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta (Eds.). Curran Associates, Inc., 64–72. https://doi.org/10.1109/TCIAIG.2012.2186810
[3] Gilles Bailly, Eric Lecolinet, and Laurence Nigay. 2008. Flower Menus: A New Type of Marking Menu with Large Menu Breadth, within Groups and Efficient Expert Mode Memorization. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '08)*. 15–22. https://doi.org/10.1145/1385569.1385575
[4] Gilles Bailly, Eric Lecolinet, and Laurence Nigay. 2016. Visual Menu Techniques. *ACM Comput. Surv.* 49, 4, Article 60 (Dec. 2016), 41 pages. https://doi.org/10.1145/3002171
[5] 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 (CHI '14)*. 3865–3874. https://doi.org/10.1145/2556288.2557093
[6] Gilles Bailly, Antti Oulasvirta, Timo Kötzing, and Sabrina Hoppe. 2013. MenuOptimizer: Interactive Optimization of Menu Systems. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology (UIST '13)*. 331–342. https://doi.org/10.1145/2501988.2502024
[7] Richard E Bellman and Lotfi Asker Zadeh. 1970. Decision-making in a fuzzy environment. *Management science* 17, 4 (1970), B–141. https://doi.org/10.1287/mnsc.17.4.B141
[8] Mark G Brockington. 1996. A taxonomy of parallel game-tree search algorithms. *ICGA Journal* 19, 3 (1996), 162–174.
[9] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4, 1 (2012), 1–43. https://doi.org/10.1109/TCIAIG.2012.2186810
[10] Michael D. Byrne. 2001. ACT-R/PM and menu selection: applying a cognitive architecture to HCI. *International Journal of Human-Computer Studies* 55, 1 (2001), 41 – 84. https://doi.org/10.1006/ijhc.2001.0469
[11] Yuan Cai and Xiang Ji. 2018. ASA-routing: A-Star Adaptive Routing Algorithm for Network-on-Chips. In *Algorithms and Architectures for Parallel Processing*, Jaideep Vaidya and Jin Li (Eds.). Springer International Publishing, Cham, 187–198.
[12] Andy Cockburn and Carl Gutwin. 2009. A Predictive Model of Human Performance With Scrolling and Hierarchical Lists. *Human–Computer Interaction* 24, 3 (2009), 273–314. https://doi.org/10.1080/07370020902990402
[13] Andy Cockburn, Carl Gutwin, and Saul Greenberg. 2007. A Predictive Model of Menu Performance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07)*. 627–636. https://doi.org/10.1145/1240624.1240723
[14] Peng Dai, Christopher H Lin, Daniel S Weld, et al. 2013. POMDP-based control of workflows for crowdsourcing. *Artificial Intelligence* 202 (2013), 52–85.
[15] John J. Dudley, Jason T. Jacques, and Per Ola Kristensson. 2019. Crowdsourcing Interface Feature Design with Bayesian Optimization. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3290605.3300482
[16] Leah Findlater and Joanna McGrenere. 2004. A Comparison of Static, Adaptive, and Adaptable Menus. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vienna, Austria) *(CHI '04)*. Association for Computing Machinery, New York, NY, USA, 89–96. https://doi.org/10.1145/985692.985704
[17] Leah Findlater, Karyn Moffatt, Joanna McGrenere, and Jessica Dawson. 2009. Ephemeral Adaptation: The Use of Gradual Onset to Improve Menu Selection Performance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Boston, MA, USA) *(CHI '09)*. Association for Computing Machinery, New York, NY, USA, 1655–1664. https://doi.org/10.1145/1518701.1518956
[18] Stephen Fitchett and Andy Cockburn. 2012. AccessRank: Predicting What Users Will Do Next. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Austin, Texas, USA) *(CHI '12)*. Association for Computing Machinery, New York, NY, USA, 2239–2242. https://doi.org/10.1145/2207676.2208380
[19] Peter A Frensch. 1994. Composition during serial learning: A serial position effect. *J. Exp. Psychol. Learn. Mem. Cogn.* 20 (1994), 423–423.
[20] Christoph Gebhardt, Brian Hecox, Bas van Opheusden, Daniel Wigdor, James Hillis, Otmar Hilliges, and Hrvoje Benko. 2019. Learning Cooperative Personalized Policies from Gaze Data. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology* (New Orleans, LA, USA)

(UIST '19). Association for Computing Machinery, New York, NY, USA, 197–208. https://doi.org/10.1145/3332165.3347933

[21] Camille Gobert, Kashyap Todi, Gilles Bailly, and Antti Oulasvirta. 2019. SAM: A Modular Framework for Self-Adapting Web Menus. In *Proceedings of the 24th International Conference on Intelligent User Interfaces* (Marina del Ray, California) *(IUI '19)*. Association for Computing Machinery, New York, NY, USA, 481–484. https://doi.org/10.1145/3301275.3302314

[22] Saul Greenberg and Ian H. Witten. 1985. Adaptive personalized interfaces—A question of viability. *Behaviour & Information Technology* 4, 1 (1985), 31–45. https://doi.org/10.1080/01449298508901785

[23] Jesse Hoey, Axel Von Bertoldi, Pascal Poupart, and Alex Mihailidis. 2007. Assisting persons with dementia during handwashing using a partially observable Markov decision process.. In *International Conference on Computer Vision Systems: Proceedings (2007)*.

[24] Anthony J. Hornof and David E. Kieras. 1997. Cognitive Modeling Reveals Menu Search in Both Random and Systematic. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI '97)*. 107–114. https://doi.org/10.1145/258549.258621

[25] Bowen Hui, Grant Partridge, and Craig Boutilier. 2009. A Probabilistic Mental Model for Estimating Disruption. In *Proceedings of the 14th International Conference on Intelligent User Interfaces* (Sanibel Island, Florida, USA) *(IUI '09)*. Association for Computing Machinery, New York, NY, USA, 287–296. https://doi.org/10.1145/1502650.1502691

[26] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. 2019. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*. 12498–12509.

[27] Jussi P. P. 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* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 4203–4215. https://doi.org/10.1145/3025453.3025580

[28] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. 1996. Reinforcement learning: A survey. *Journal of artificial intelligence research* 4 (1996), 237–285.

[29] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. 2019. Model-based reinforcement learning for atari. *arXiv:1903.00374* (2019).

[30] Sunjun Kim, Byungjoo Lee, Thomas van Gemert, and Antti Oulasvirta. 2020. Optimal Sensor Position for a Computer Mouse. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376735

[31] Levente Kocsis and Csaba Szepesvári. 2006. Bandit Based Monte-Carlo Planning. In *Proceedings of the 17th European Conference on Machine Learning* (Berlin, Germany) *(ECML'06)*. Springer-Verlag, Berlin, Heidelberg, 282–293. https://doi.org/10.1007/11871842_29

[32] Pat Langley. 1997. Machine learning for adaptive user interfaces. In *Annual Conference on Artificial Intelligence (KI 1997)*. 53–62. https://doi.org/10.1007/3540634932_3

[33] Talia Lavie and Joachim Meyer. 2010-08. Benefits and Costs of Adaptive User Interfaces. *Int. J. Hum.-Comput. Stud.* 68, 8 (2010-08), 508–524. https://doi.org/10.1016/j.ijhcs.2010.01.004

[34] Dong-Seok Lee and Wan Chul Yoon. 2004. Quantitative results assessing design issues of selection-supportive menus. *International Journal of Industrial Ergonomics* 33, 1 (2004), 41–52.

[35] Katri Leino, Kashyap Todi, Antti Oulasvirta, and Mikko Kurimo. 2019. Computer-Supported Form Design Using Keystroke-Level Modeling with Reinforcement Learning. In *Proceedings of the 24th International Conference on Intelligent User Interfaces: Companion (IUI '19)*. 85–86. https://doi.org/10.1145/3308557.3308704

[36] Luis Leiva. 2012. Interaction-Based User Interface Redesign. In *Proceedings of the 2012 ACM International Conference on Intelligent User Interfaces* (Lisbon, Portugal) *(IUI '12)*. Association for Computing Machinery, New York, NY, USA, 311–312. https://doi.org/10.1145/2166966.2167028

[37] Wanyu Liu, Gilles Bailly, and Andrew Howes. 2017. Effects of Frequency Distribution on Linear Menu Performance. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 1307–1312. https://doi.org/10.1145/3025453.3025707

[38] J. Derek Lomas, Jodi Forlizzi, Nikhil Poonwala, Nirmal Patel, Sharan Shodhan, Kishan Patel, Ken Koedinger, and Emma Brunskill. 2016. Interface Design Optimization as a Multi-Armed Bandit Problem. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 4142–4153. https://doi.org/10.1145/2858036.2858425

[39] I. Scott MacKenzie. 1992. Fitts' Law as a Research and Design Tool in Human-Computer Interaction. *Hum.-Comput. Interact.* 7, 1 (March 1992), 91–139. https://doi.org/10.1207/s15327051hci0701_3

[40] Kent L Norman. 1991. *The psychology of menu selection: Designing cognitive control at the human/computer interface*. Intellect Books.

[41] Antti Oulasvirta. 2018. *Computational interaction*. Oxford University Press. https://doi.org/10.1093/oso/9780198799603.001.0001

[42] Antti Oulasvirta, Niraj Ramesh Dayama, Morteza Shiripour, Maximilian John, and Andreas Karrenbauer. 2020. Combinatorial Optimization of Graphical User Interface Designs. *Proc. IEEE* 108, 3 (2020), 434–464. https://doi.org/10.1109/JPROC.2020.2969687

[43] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[44] Angel R. Puerta, Henrik Eriksson, John H. Gennari, and Mark A. Musen. 1994. Model-Based Automated Generation of User Interfaces. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 1)* (Seattle, Washington, USA) *(AAAI '94)*. American Association for Artificial Intelligence, USA, 471–477.

[45] Reyhaneh Raissi, Evanthia Dimara, Jacquelyn H. Berry, Wayne D. Gray, and Gilles Bailly. 2020. Retroactive Transfer Phenomena in Alternating User Interfaces. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3313831.3376538

[46] Andrew Sears and Ben Shneiderman. 1994. Split Menus: Effectively Using Selection Frequency to Organize Menus. *ACM Trans. Comput.-Hum. Interact.* 1, 1 (March 1994), 27–51. https://doi.org/10.1145/174630.174632

[47] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. 2015. Taking the human out of the loop: A review of Bayesian optimization. *Proc. IEEE* 104, 1 (2015), 148–175.

[48] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016). https://doi.org/10.1038/nature16961

[49] Harold Soh, Scott Sanner, Madeleine White, and Greg Jamieson. 2017. Deep Sequential Recommendation for Personalized Adaptive User Interfaces. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces* (Limassol, Cyprus) *(IUI '17)*. Association for Computing Machinery, New York, NY, USA, 589–593. https://doi.org/10.1145/3025171.3025207

[50] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[51] Kashyap Todi, Jussi Jokinen, Kris Luyten, and Antti Oulasvirta. 2018. Familiarisation: Restructuring Layouts with Visual Learning Models. In *23rd International Conference on Intelligent User Interfaces (IUI '18)*. 547–558. https://doi.org/10.1145/3172944.3172949

[52] Kashyap Todi, Jussi Jokinen, Kris Luyten, and Antti Oulasvirta. 2019. Individualising Graphical Layouts with Predictive Visual Search Models. *ACM Trans. Interact. Intell. Syst.* 10, 1, Article 9 (Aug. 2019), 24 pages. https://doi.org/10.1145/3241381

[53] Kashyap Todi, Daryl Weir, and Antti Oulasvirta. 2016. Sketchplore: Sketch and Explore with a Layout Optimiser. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems* (Brisbane, QLD, Australia) *(DIS '16)*. Association for Computing Machinery, New York, NY, USA, 543–555. https://doi.org/10.1145/2901790.2901817

[54] Theophanis Tsandilas and m. c. schraefel. 2007. Bubbling Menus: A Selective Mechanism for Accessing Hierarchical Drop-down Menus. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '07)*. Association for Computing Machinery, New York, NY, USA, 1195–1204. https://doi.org/10.1145/1240624.1240806

[55] Jean Vanderdonckt, Sara Bouzit, Gaëlle Calvary, and Denis Chêne. 2019. Exploring a Design Space of Graphical Adaptive Menus: Normal vs. Small Screens. *ACM Trans. Interact. Intell. Syst.* 10, 1, Article 2 (July 2019), 40 pages. https://doi.org/10.1145/3237190

[56] Roel Vertegaal. 2003. Attentive User Interfaces. *Commun. ACM* 46, 3 (March 2003), 30–33. https://doi.org/10.1145/636772.636794

[57] Ying Wu and Thomas S. Huang. 1999. Vision-Based Gesture Recognition: A Review. In *Gesture-Based Communication in Human-Computer Interaction*, Annelies Braffort, Rachid Gherbi, Sylvie Gibet, Daniel Teil, and James Richardson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 103–115.

[58] Steve Young, Milica Gašić, Blaise Thomson, and Jason D Williams. 2013. POMDP-based statistical spoken dialog systems: A review. *Proc. IEEE* 101, 5 (2013), 1160–1179.

[59] Arianna Yuan and Yang Li. 2020. Modeling Human Visual Search Performance on Realistic Webpages Using Analytical and Deep Learning Methods. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3313831.3376870