Improving Human Decision-Making with Machine Learning

Hamsa Bastani[1] Osbert Bastani[1] Wichinpong Park Sinchaisri[2]

[1]University of Pennsylvania [2]University of California, Berkeley

Abstract

A key aspect of human intelligence is their ability to convey their knowledge to others in succinct forms. However, despite their predictive power, current machine learning models are largely blackboxes, making it difficult for humans to extract useful insights. Focusing on sequential decision-making, we design a novel machine learning algorithm that conveys its insights to humans in the form of interpretable "tips". Our algorithm selects the tip that best bridges the gap in performance between human users and the optimal policy. We evaluate our approach through a series of randomized controlled user studies where participants manage a virtual kitchen. Our experiments show that the tips generated by our algorithm can significantly improve human performance relative to intuitive baselines. In addition, we discuss a number of empirical insights that can help inform the design of algorithms intended for human-AI interfaces. For instance, we find evidence that participants do not simply blindly follow our tips; instead, they combine them with their own experience to discover additional strategies for improving performance.

1 Introduction

A hallmark of human intelligence is our ability to distill knowledge and expertise into clear and simple representations that can be communicated to others. This ability is critical for human progress, since even simple insights often require a great deal of costly experience and experimentation to discover—for instance, Newton's laws of motion span a few lines of text, yet represent the culmination of millennia of scientific thinking. As a more commonplace example, humans naturally pass on their knowledge to help others—for instance, doctors regularly communicate best practices to their less experienced peers to help them improve their performance [1]. While often succinct, these best practices typically require a great deal of trial and error to discover. By sharing our insights with others, we avoid repeatedly reinventing concepts and instead focus our efforts on expanding the sum of human knowledge.

While recent advances in deep learning have enabled machines to achieve human-level or super-human performance at many artificial intelligence tasks [2, 3], the knowledge behind their abilities is hidden within blackbox models such as deep neural networks. As a consequence, despite enormous progress, deep learning has remained a tool for pattern matching and prediction rather than one for augmenting human understanding. One of the holy grails of machine learning is to develop techniques for learning compact representations of knowledge that can be understood by humans [4]. Recent work on interpretable machine learning has strived to learn structured models where the computation is transparent to the user, such as decision trees [5, 6], rule lists [7], generalized additive models [8], sparse linear models [9], or interpretable representations [10]. Indeed, recent work suggests that there often exist interpretable models that achieve performance similar to their deep neural network counterparts [11]; instead, the challenge lies in discovering these high-performing models.

A natural question arises: can we leverage these techniques to learn representations of knowledge that are useful to humans? In particular, existing work on interpretable machine learning has largely focused on whether a human can understand the computation performed by a model [8, 12]; in contrast, our goal is to examine whether these models can convey useful insights to humans. To this end, we devise a novel algorithm for inferring simple tips that can be used to improve performance of a human user. We focus on settings where the human must make a sequence of decisions to optimize an outcome in a complex environment. These settings are particularly challenging for humans, since they must trade off short-term and long-term rewards in ways that are often highly counterintuitive. Our algorithm builds on the idea of using model distillation [13, 14] for interpretable reinforcement learning [15, 16], which involves first training a high-performance neural network decision-making policy using reinforcement learning [17], and then training an interpretable policy to approximate this neural network.

In our setting, we are interested in interpreting the discrepancy between the human policy and the neural network, not the neural network itself. To this end, we derive a novel objective encoding this discrepancy; then, our algorithm selects an interpretable tip that minimizes this discrepancy. Intuitively, this tip best bridges the gap between the actions taken by the human users and the ones taken by the neural network. Importantly, it does so in a way that accounts for which actions are consequential for achieving higher performance—i.e., it suggests actions that are expected to improve the long-term performance of the human rather than to simply mimic the neural network. While previous work has leveraged machine learning to predict when humans make mistakes compared to the optimal policy [18, 19, 20], they do so in an uninterpretable way that makes it difficult to convey their insights to humans.

Two criteria are needed for our approach to be effective. First, our algorithm must be able to identify sufficiently useful tips to improve performance. Second, humans must be able to understand our tip and furthermore comply with its suggested actions. To study these issues, we perform an extensive user study in a scenario where human users play a game managing a virtual kitchen; an illustration of this task is shown in Figure 1. In this game, users must

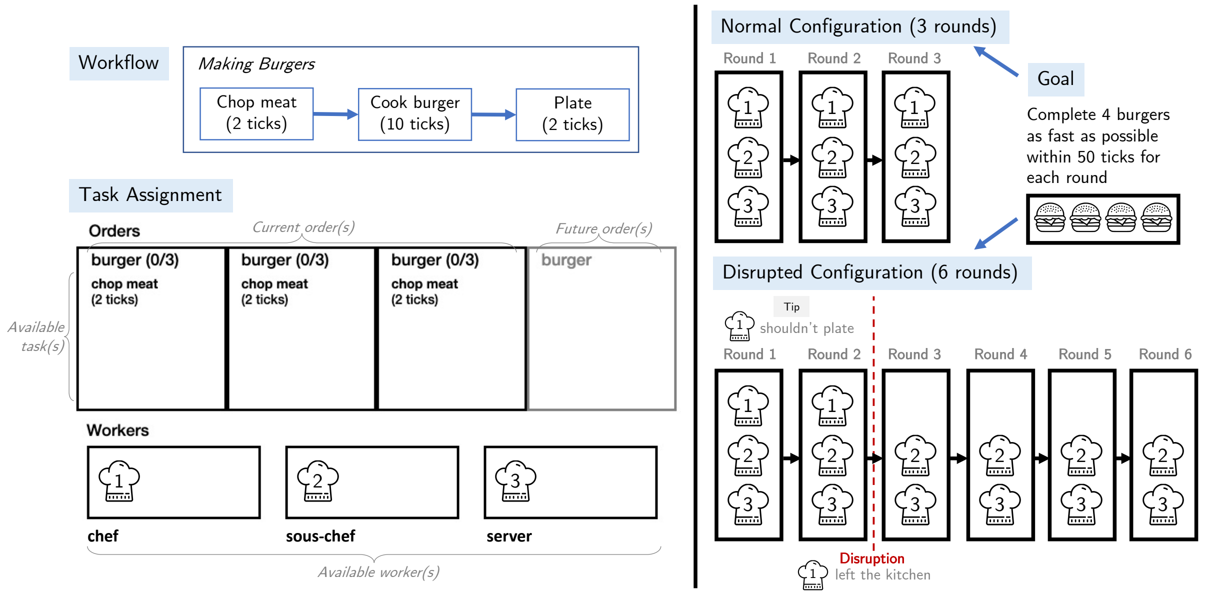


Figure 1: Overview of kitchen management game. The left panel depicts what participants see: (i) the workflow required to complete a burger order, and (ii) the game screen that allows available tasks to be dragged and dropped to one of 3 virtual workers. The right panel depicts the study design: in the normal configuration, participants play the same game for 3 rounds; in the disrupted configuration, participants play the same game for 2 rounds, face a disruption in the kitchen, and play the disrupted game for 4 rounds.

assign subtasks to virtual workers with varying capabilities in a way that optimizes the time it takes to complete a set of food orders. This game is challenging because human users must make tradeoffs such as deciding whether to greedily assign a worker to a subtask that they are slow to complete, or leave them idle in anticipation of a more suitable subtask.

Our results demonstrate that our algorithm can generate novel insights that enable human users to substantially improve their performance compared to counterparts that are not shown the tip or that are shown alternative tips derived from natural baselines. Interestingly, users do not merely adjust their actions by blindly following the tip. Instead, as they gain experience with the game, they increasingly understand the significance of the tip and improve their performance in ways beyond the surface-level meaning of the tip. Overall, our findings suggest that our algorithm infers interpretable and useful insights about the underlying task, and successfully conveys these insights to human participants.

Beyond demonstrating the effectiveness of our algorithm, our results also have implications for the design of effective human-AI interfaces. Examining the rate at which participants comply with various tips, we find evidence suggesting that, to achieve high compliance, the human must not only understand the actions suggested by the tip, but also how to *operationalize* the suggested actions. Thus, effective human-AI interfaces must convey not just the decisions suggested by the AI, but also how these decisions fit into a broader strategy.

2 Inferring Tips via Interpretable Reinforcement Learning

Consider a human making a sequence of decisions to achieve some desired outcome. We study settings where current decisions affect future outcomes—for instance, if the human decides to consume some resources at the current time step, they can no longer use these resources in the future. These settings are particularly challenging for decision-making due to the need to reason about how current actions affect future decisions, making them ideal targets for leveraging tips to improve human performance.

We begin by formalizing the tip inference problem. We model our setting as the human acting to maximize reward in an undiscounted Markov Decision Process (MDP) $\mathcal{M} = (S, A, R, P)$ over a finite time horizon T [21]; here, S is the state space, A is the action space, R is the reward function, and P is the transition function. Intuitively, a state $s \in S$ captures the current configuration of the system (e.g., available resources), and an action $a \in A$ is a decision that the human can make (e.g., consume some resources to produce an item). In particular, we represent the human as a decision-making policy π_H mapping states to (possibly random) actions. At each time step $t \in \{1, ..., T\}$, the human observes the current state s_t and selects an action a_t to take according to the probability distribution $p(a_t \mid s_t) = \pi_H(s_t, a_t)$. Then, they receive reward $r_t = R(s_t, a_t)$, and the system transitions to the next state s_{t+1}, which is a random variable with probability distribution $p(s_{t+1} \mid s_t, a_t) = P(s_t, a_t, s_{t+1})$, after which the process is repeated until $t = T$. A sequence of state-action-reward triples sampled according to this process is called a *rollout*, denoted $\zeta = ((s_1, a_1, r_1), ..., (s_T, a_T, r_T))$. The human's goal is to act according to a policy π_H that maximizes the cumulative expected reward $J(\pi_H)$, where

$$J(\pi) = \mathbb{E}_{\zeta \sim D^{(\pi)}} \left[\sum_{t=1}^{T} r_t \right],$$

and where $D^{(\pi)}$ is the distribution of rollouts induced by using policy π.

Now, given the MDP $\mathcal{M}$ along with the human policy π_H, our goal is to learn a tip ρ that most improves the cumulative expected reward. Formally, a tip indicates that in certain states s, the human should use action $\rho(s) \in A$ instead of following their own policy π_H. For simplicity, we assume that the human always follows the tip. While this assumption does not necessarily hold in practice, we find that it works sufficiently well to improve performance as long as the human can understand both the tip and its rationale (we give a detailed discussion on compliance in our Results section). Thus, we consider tips in the form of a single, interpretable rule:

$$\rho(s) = \text{if } \psi(s) \text{ then take action } a,$$

where $a \in A$ is an action and $\psi(s) \in \{\text{true}, \text{false}\}$ is a logical predicate over states $s \in S$—e.g., it might say that a sufficient quantity of a certain resource is currently available. Intuitively, a tip $\rho = (\psi, a)$ says that if the predicate ψ is satisfied, then the human should use action a; otherwise, they should use their own policy π_H. Assuming the human follows this tip exactly, then the resulting policy they use is $\pi_H \oplus \rho$, where

$$(\pi \oplus \rho)(s, a') = \begin{cases} \mathbb{1}(a' = a) & \text{if } \psi(s) \\ \pi(s, a') & \text{otherwise,} \end{cases}$$

where $\mathbb{1}$ is the indicator function; that is, the human takes action a with probability one if $\psi(s)$ holds, and follows their existing policy otherwise. Then, our goal is to compute the tip that most improves the human's performance—i.e.,

$$\rho^* = \arg\max_{\rho} J(\pi_H \oplus \rho). \tag{1}$$

This formulation ensures that the optimal tip is consequential to improving performance, rather than naïvely identifying state-action pairs that frequently differ between the human and optimal policies; we compare to such a baseline in our Results section.

Next, we describe our algorithm for solving this problem. To guide our algorithm, we first compute the optimal Q-function Q^* by (approximately) solving the Bellman equations

$$Q^*(s, a) = R(s, a) + \mathbb{E}_{a \sim \pi(s,\cdot), s' \sim P(s,a,\cdot)}[Q^*(s, a)]$$

using Q-learning [22], where we parameterize Q^* using a neural network with a single hidden layer. Then, we can rewrite the objective $J(\pi_H \oplus \rho)$ in (1) as follows [16]:

$$J(\pi_H \oplus \rho) = \mathbb{E}_{\zeta \sim D^{(\pi_H \oplus \rho)}} \left[\sum_{t=1}^{T} Q^*(s_t, a_t) \right].$$

Since we do not have access to samples $\zeta \sim D^{(\pi_H \oplus \rho)}$, we use the approximation using $D^{(\pi_H \oplus \rho)} \approx D^{(\pi_H)}$, which is good as long as the tip ρ does not drastically change the human's decisions. Furthermore, we approximate the expectation in our objective using rollouts $\zeta_1, ..., \zeta_k \sim D^{(\pi_H)}$ from the human policy π_H, where $\zeta_i = ((s_{i,1}, a_{i,1}, r_{i,1}), ..., (s_{i,T}, a_{i,T}, r_{i,T}))$. Thus, our algorithm computes the tip

$$\hat{\rho} = \arg\max_{\rho} \frac{1}{k} \sum_{i=1}^{k} \sum_{t=1}^{T} Q^*(s_{i,t}, (a_{i,t} \oplus \rho)(s_{i,t})), \tag{2}$$

where for a given tip $\rho = (\psi, a)$ and action a', we define

$$(a' \oplus \rho)(s) = \begin{cases} a & \text{if } \psi(s) = 1 \\ a' & \text{otherwise.} \end{cases}$$

We optimize (2) by enumerating through candidate tips ρ, evaluating the objective, and selecting the tip $\hat{\rho}$ with the highest objective value.

3 Case Study: Virtual Kitchen-Management Game

Inspired by a recent multi-agent reinforcement learning benchmark [23], we consider a sequential decision-making task in the form of a virtual kitchen-management game that can be played by individual human users (see Figure 1).[1] In this game, the user takes the role of a manager of several virtual workers—namely, chef, sous-chef, and server—serving burgers in a virtual kitchen. Each burger consists of a fixed set of subtasks that must be completed in order—namely, chopping meat, cooking the burger, and plating the burger.The game consists of discrete time steps; on each step, the user must decide which (if any) subtask to assign to each idle worker. The worker then completes the subtask across a fixed number of subsequent time steps, and then becomes idle again. A burger is completed once all its subtasks are completed, and the user completes the game once four burger orders are completed. The user's goal is to complete the game in as few steps as possible.

There are two key aspects of the game that make it challenging. First, the subtasks have dependencies—i.e., a subtask can only be assigned once previous subtasks of the same order have already been completed. For example, the "plate burger" task can only be assigned once the "cook burger" task is completed. Second, the virtual workers have heterogeneous skills—i.e., different workers take different numbers of steps to complete different subtasks. For example, the chef is skilled at chopping/cooking but performs poorly at plating, while the server is the opposite, and the sous-chef has average skill on all subtasks; see Table 1 in the Appendix for details. Ideally, one would match workers to tasks that they are skilled at to reduce completion time. Thus, the user faces the following dilemma. When a worker becomes available but is not skilled at any of the currently available subtasks, then the user must decide between (i) assigning a suboptimal subtask to that worker, potentially creating a bottleneck, or (ii) leaving the worker idle until a more suitable subtask becomes available. For instance, if the server is idle but all available subtasks are "cook burger", then the user must either (i) assign cooking to the unskilled server, thereby slowing down completion of that burger and eliminating the possibility of assigning plating to the server for the near future, or (ii) leave the server idle until a "plate burger" subtask becomes available. Furthermore, users are not shown the number of steps a worker takes to complete a subtask until they assign the subtask to that worker; instead, they must experiment to learn this information.

We consider two scenarios of the game, differing only in terms of worker availability. In the first scenario, the kitchen is *fully-staffed*, where the human user has access to all three virtual workers (chef, sous-chef, and server). In the second scenario, the human user faces a disruption and the kitchen becomes *understaffed*, with only two virtual workers (sous-chef and server).

[1] We deliberately choose a task where we can compute the optimal policy (see the Appendix for a description of this policy), which enables us to evaluate human suboptimality.

4 Experimental Design

We investigate how humans interpret and follow the tips inferred by our algorithm in pre-registered behavioral experiments involving Amazon Mechanical Turk (AMT) users.[2] Participants are compensated a flat rate for completion of the study, and a relatively larger performance-based bonus determined by how quickly they complete each round of the game; see Experimental Design Details in the Appendix for payment details and game screenshots.

First and foremost, we study whether participants shown our tip (the "algorithm" treatment condition) outperform participants not shown any tips (the control group), participants shown a tip suggested by previous human participants who played the same scenario multiple times (the "human" condition), and participants shown a tip derived by a baseline algorithm that identifies the state-action pair where human participants and the optimal policy most frequently differ (the "baseline" condition). Second, we examine participant behaviors in response to different tips, particularly their compliance and how they learn beyond the provided tips.

Our experiments proceed in two phases: in Phase I, we collect trace and survey data for learning tips (without showing tips to any participants); in Phase II, we randomize participants to different conditions (where they are shown varying tips) and evaluate their performance. In both phases, each participant plays a sequence of three to six rounds of the game, which we call a *configuration*; this approach enables us to study both how performance varies with the tip they are shown, as well as how it evolves across games as participants gain experience.

We consider two configurations. In the *normal* configuration, each participant simply plays three rounds of the fully-staffed scenario. In the *disrupted* configuration, each participant plays two rounds of the fully-staffed scenario, followed by four rounds of the understaffed scenario. Intuitively, the normal configuration studies whether tips can help human participants fine-tune their performance. In contrast, the disrupted configuration is designed to show how tips can help participants adapt to novel situations where the optimal strategy substantially changes. The set of participants across all four configuration-phase pairs is mutually exclusive—i.e., no participant has prior experience with any version of the game.

We provide an overview of each phase here; see Figure 2 for a summary and the Appendix for additional details. In Phase I, we have a small sample of human participants (normal: $N =$ 183 participants; disrupted: $N = 172$ participants) play each configuration without tips, and collect data on the actions they take. Then, we use our tip inference algorithm in conjunction with this data to generate our tip. Each participant is also shown a comprehensive list of candidate tips at the end of Phase I, and is asked to select the tip they believe would most improve the performance of future players; we take the most commonly chosen tip as the "human tip" baseline. Our second baseline computes the best tip that would match the human user's actions in Phase I with those of the optimal policy, without regard to which

[2]The full pre-registration document for our study is available at `https://aspredicted.org/blind.php?x=8ye5cb`

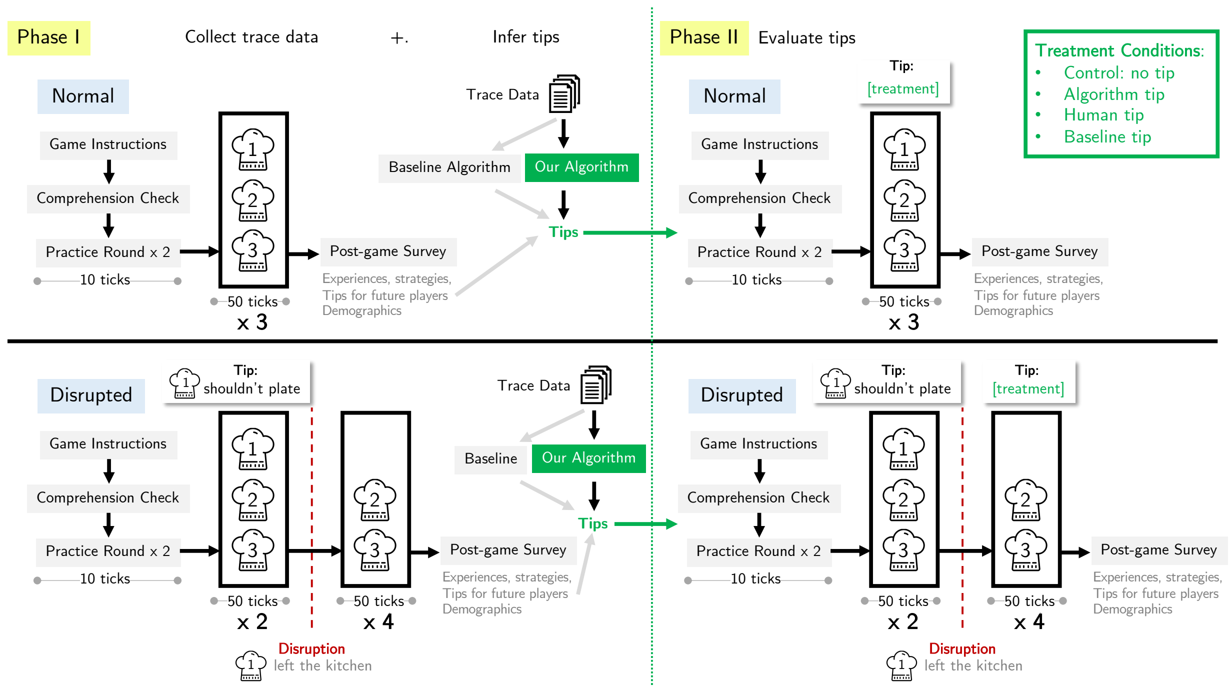


Figure 2: Overview of experimental flow. The top two panels depict Phase I (left) and II (right) for the normal configuration, where each participant plays 3 fully-staffed scenarios. The bottom two panels depict Phase I (left) and II (right) for the disrupted configuration, where each participant plays 2 fully-staffed and 4 understaffed scenarios. Participants in Phase II are randomly assigned to one of 4 conditions (control, algorithm, human, and baseline). The set of participants across all 4 configuration-phase pairs is mutually exclusive.

actions are consequential for improving performance. In particular, given rollouts $\hat{D}$ from the optimal policy π^*, it computes the frequency $C^*(s,a)$ of state-action pairs in $\hat{D}$, and then selects

$$\hat{\rho}_{\text{bl}} = \arg\max_{\rho} \frac{1}{k} \sum_{i=1}^{k} \sum_{t=1}^{T} C^*(s_{i,t}, (a_{i,t} \oplus \rho)(s_{i,t})). \tag{3}$$

Intuitively, this objective ignores the structure of the MDP implicitly encoded in the Q-function, and instead directly tries to imitate the optimal policy.

Then, in Phase II, we randomly assign a large number of new participants (normal: $N = 1,317$ participants; disrupted: $N = 1,011$ participants) into one of four conditions: control (no tip), algorithm (shown the tip $\hat{\rho}$ inferred by our algorithm in Eq. (2) based on Phase I data), human (shown the tip most frequently chosen by Phase I participants), or baseline (shown the tip $\hat{\rho}_{\text{bl}}$ inferred by the baseline algorithm in Eq. (3) based on Phase I data). In the normal configuration, Phase II participants are shown the tip designated by their condition for the fully-staffed scenario on all rounds. In the disrupted configuration, they are shown the tip designated by their condition for the understaffed scenario (the last 4 rounds). Our goal in the first 2 rounds of the disrupted configuration is to quickly acclimate participants to the fully-staffed scenario in a way that is consistent across conditions. Thus, we show our algorithm tip "Chef should never plate" across all conditions (including control) for the first two rounds—as shown in our Results section, this tip most quickly improves performance.

Finally, we assess performance in each condition both through the overall completion time in the final round, as well as the fraction of participants who learn the optimal policy (20 and 34 time steps for the fully-staffed and understaffed scenarios, respectively).

5 Results

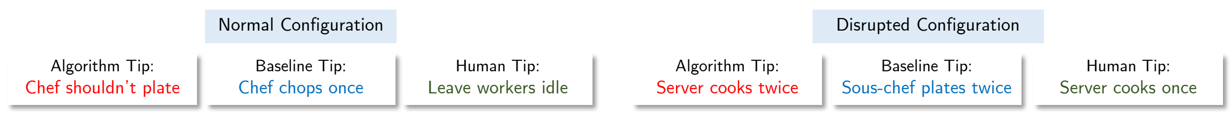
Despite their simplicity and conciseness, we find that our tips can significantly improve participant performance since they capture strategies that are hard for participants to learn; in contrast, alternative tips have varying empirical shortcomings that reduce their effectiveness. We also describe how participant compliance varies across tips; in particular, participants must understand how to operationalize the tip effectively for them to comply. Finally, we find evidence that participants do not blindly follow our tips, but combine them with their own experience to discover additional strategies beyond our tips. Figure 3a shows the tips inferred in each condition for each configuration using trace and survey data from Phase I.

Performance: Figure 3 shows performance results across all four conditions and both configurations. Figure 3b & 3c show participant performance in the final round of our game, Figure 3d & 3e show how performance improves across rounds, and Figure 3f & 3g show the fraction of users achieving optimal performance across rounds. We discuss these results below.

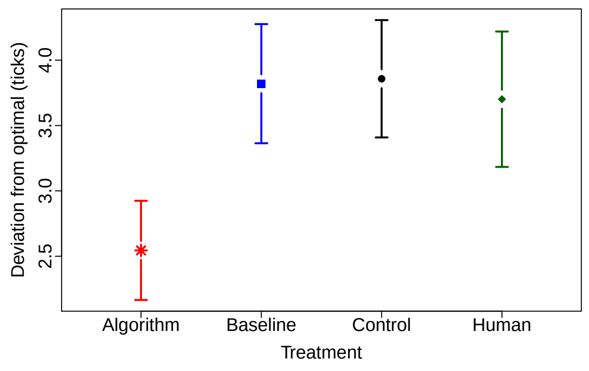
The normal configuration is relatively easy—a substantial fraction (24%) discover the optimal policy by the final round without the aid of tips (control group). As shown in Figure 3b, participants shown our tip completed the final round in 22.5 steps on average, significantly outperforming participants in the control group ($p < 10^{-4}$), those shown the human-suggested tip ($p = 2\times10^{-4}$), and those shown the tip from the baseline algorithm ($p < 10^{-4}$).[3] Our tip speeds up learning by at least one round compared to the other conditions—i.e., the performance of participants given our tip on round k was similar to or better than the performance of participants in other conditions on round $k+1$ (Figure 3d). Our tip also helped more participants (35%) achieve optimal performance (20 steps) in the final round, compared to 24-29% in other conditions.

The disrupted configuration is substantially harder, since participants must adapt to the more counterintuitive understaffed scenario. Perhaps as a consequence, participants benefit much more from tips: those in the control group took four rounds to achieve the same level of performance as those shown our tip on the first round. Participants shown our tip

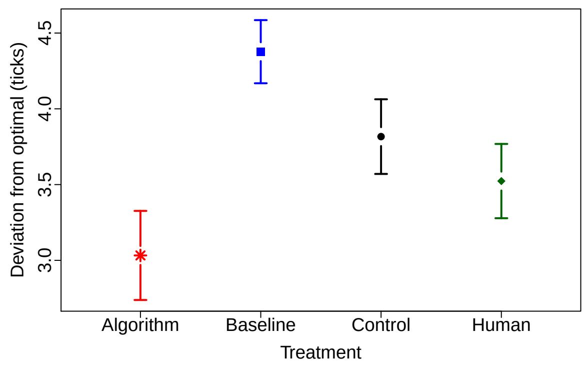
[3]Results remain highly statistically significant under a Bonferroni correction for multiple hypothesis testing.


Normal Configuration
Algorithm Tip: Chef shouldn't plate
Baseline Tip: Chef chops once
Human Tip: Leave workers idle
Disrupted Configuration
Algorithm Tip: Server cooks twice
Baseline Tip: Sous-chef plates twice
Human Tip: Server cooks once

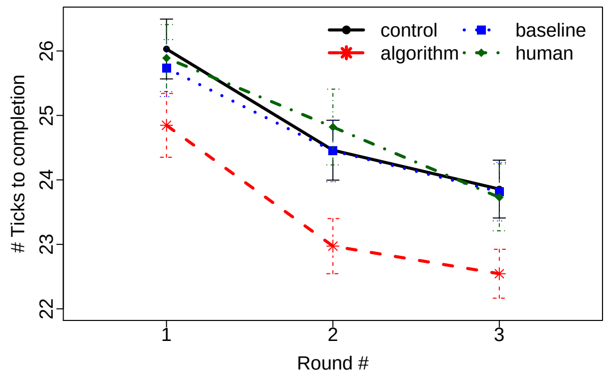
(a) Tips for each condition and configuration



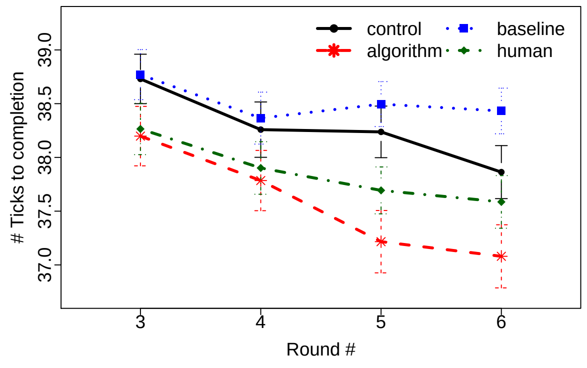
(b) Final Round Performance (Normal)



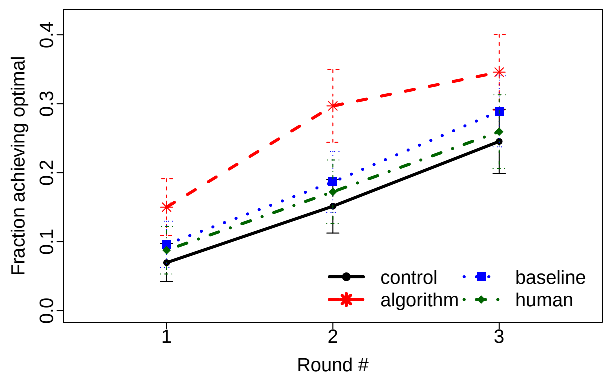
(c) Final Round Performance (Disrupted)



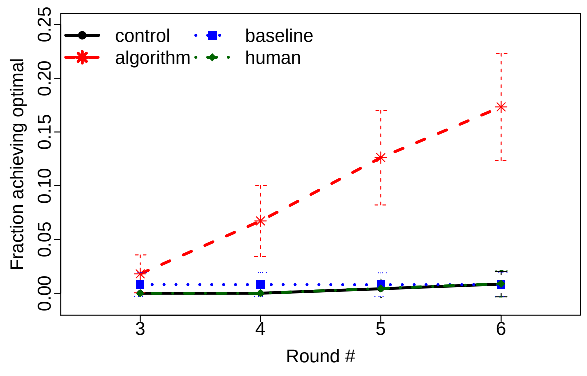
(d) Performance Over Time (Normal)



(e) Performance Over Time (Disrupted)



(f) Fraction Achieving Optimal (Normal)



(g) Fraction Achieving Optimal (Disrupted)

Figure 3: Phase II Participant Performance. The top row shows the tips derived for each condition and configuration based on Phase I data. Remaining rows depict various views of participant performance across conditions in the normal (left) and disrupted (right) configurations. The top row shows performance in the last round of the configuration, the second row shows how participant performance improves over time, and the third row shows the fraction of participants who execute an optimal policy over time.

completed the final round in 37.1 steps, again significantly outperforming participants in the control group ($p < 10^{-4}$), those shown the human-suggested tip ($p = 6 \times 10^{-3}$), and those shown the tip from the baseline algorithm ($p < 10^{-4}$). In the disrupted configuration, the baseline tip actually reduces participant performance, likely by misleading participants. More starkly, 19% of participants shown our tip achieved optimal performance (34 steps) in the final round, compared to less than 1% in all other conditions—i.e., our tip uniquely helps participants learn to play optimally.[4]

Shortcomings of alternative tips: Next, we discuss potential reasons for the success of our tips compared to the others. First, the baseline tips likely perform poorly since it blindly tries to mimic the optimal policy rather than focusing on consequential actions. Specifically, in the disrupted configuration, the baseline tip "Sous-chef should plate twice" suggests actions that occur at the end of the game, after it is too late for participants to significantly improve their performance. In contrast, our algorithm focuses on mimicking decisions made by the optimal policy that have long-term benefits—e.g., our tip "Server should cook twice" frees the sous-chef to plate later in the game.

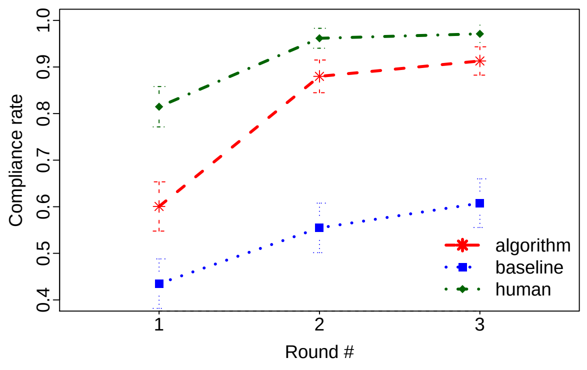
While the human-suggested tips consistently improve performance compared to the control group, they can be overly general or incorrect. In the normal configuration, Phase I participants were unable to translate their strategy into a specific tip—i.e., their suggested tip "Strategically leave some workers idle" captures a strategy needed to perform better but does convey details needed to ascertain the optimal strategy. Alternatively, in the disrupted configuration, Phase I participants provided an incorrect tip, suggesting "Server should cook once", whereas the optimal policy actually assigns the server to cook twice (as suggested by our tip)—i.e., participants identified the correct direction of change, but at an insufficient magnitude.

Compliance: As discussed earlier, the effectiveness of a tip critically depends on whether humans are able to understand it and implement it effectively. This involves both complying with the tip's suggested actions as well as modifying other portions of their strategy to make full use of the tip. Here, we examine compliance with the tips; we examine learning strategies beyond the tips below. In particular, Figure 4 shows the fraction of participants that complied with the tip they were offered in each condition.[5] As can be seen, participants increasingly comply with the tips shown over time in all conditions.

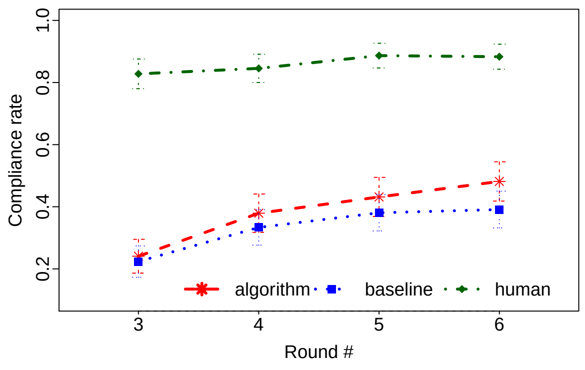
Compliance with the baseline tip was relatively low in both configurations, suggesting that participants did not find it as useful. Alternatively, compliance with the human-suggested tip was higher than compliance with our tip, particularly in the disrupted configuration; most

[4]There were no significant differences in performance across conditions when playing the two fully-staffed rounds in the disrupted configuration. Therefore, the relatively worse performance under other conditions reflect the informativeness of alternative tips.

[5]Importantly, participants were not informed of the source of the tip (i.e., algorithm or human suggestions), so any variation in compliance is due to the content of the tip rather than algorithmic aversion [24].



(a) Normal configuration



(b) Disrupted configuration

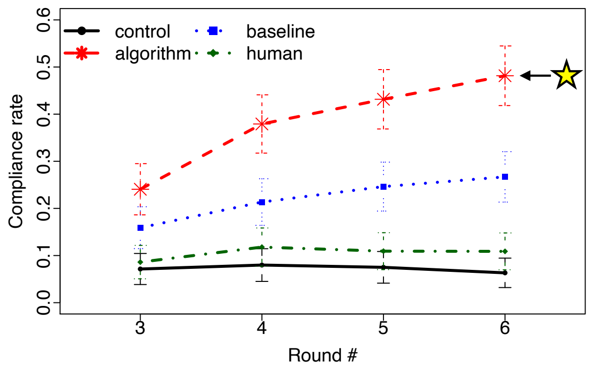
Figure 4: Compliance with Tips. Participant compliance in Phase II with the respective tip they were shown in each condition for the normal (left) and disrupted (right) configurations over time.

likely, the human-suggested tip is more intuitive since it was devised by humans. In contrast, even if our tip is correct, it can be highly counterintuitive—in the disrupted scenario, our tip "Server should cook twice" is counterintuitive since the server is very slow at cooking, yet this strategy is the only way to achieve optimal performance.

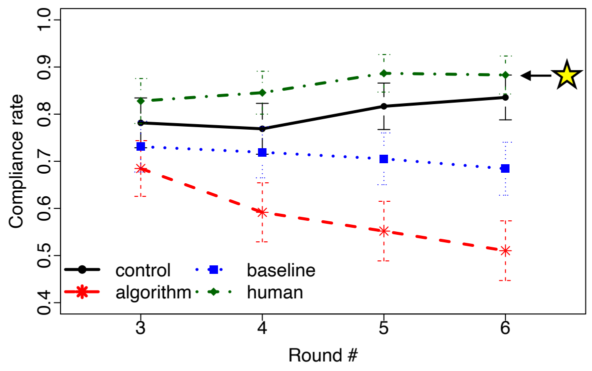
Our results suggest that participants do not blindly follow tips; instead, they only follow them if they understand why the suggested strategy is effective. Thus, compliance is not only a function of the syntactic structure of the tip (which is unchanged across conditions), but also its semantic content. When the optimal strategy is counterintuitive, we observe an intrinsic tradeoff between the optimality of the tip and compliance to the tip. In the disrupted configuration, our tip succeeds despite relatively lower compliance since it suggests a highly effective strategy; as seen in Figure 3g, participants that understand this strategy can achieve optimal performance. We also found qualitative evidence that participants ignored tips they could not understand based on their post-game survey responses, which we discuss in the Appendix.

Learning beyond tips: Next, we examine how humans learned a strategy that goes beyond the tips they were shown. To this end, we examine *cross-compliance*, which is the compliance of the participant to tips other than the one they were shown. Naïvely, there is no reason to expect participants to cross-comply with a tip that we did not show them beyond the cross-compliance exhibited by the control group (assuming that it does not overlap with the tip they were shown). Thus, any cross-compliance beyond that of the control group measures how a tip enables participants to learn strategies beyond the stated tip. We focus on the disrupted configuration since it is more challenging for participants, leading to more complex cross-compliance patterns across conditions. Figure 5 shows the cross-compliance of participants in each condition with the different tips (algorithm, baseline, human), as well as a new rule ("Server chops once") that is taken by the optimal policy (but not shown to any participants).

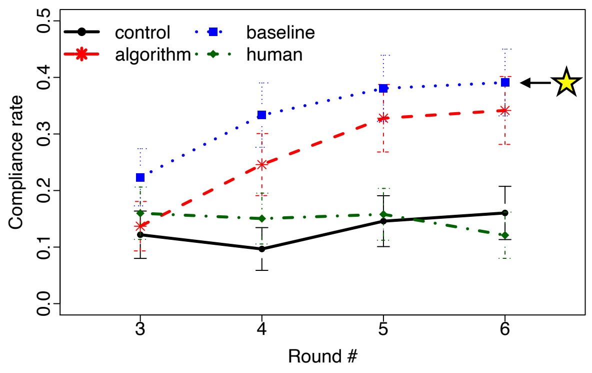
Participants in the human and control groups only comply with the human tip; indeed, the human-suggested tip actually contradicts the optimal policy, thereby preventing participants from learning the other tips which are part of the optimal policy. Similarly, participants shown the baseline tip only have high compliance with this tip, indicating that the baseline tip could not help participants uncover the optimal policy. In contrast, participants who received our tip have high cross-compliance with *all* parts of the optimal policy (i.e., the baseline and unshown tips); furthermore, our algorithm is the only condition where cross-compliance with the suboptimal human tip decreases over time. That is, our tip uniquely enables participants to combine the tip with their own experience to discover useful strategies beyond what is stated in the tip.



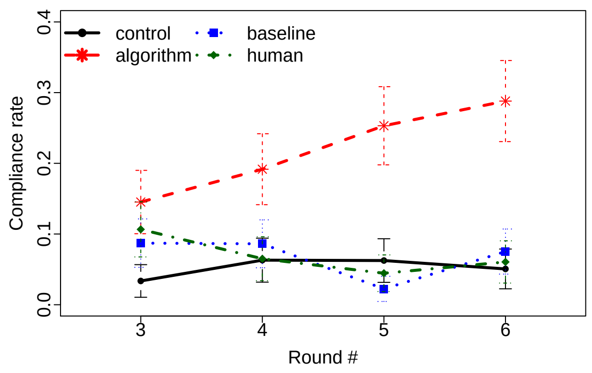
(a) Algorithm Tip: "Server cooks twice"



(b) Human Tip: "Server cooks once"



(c) Baseline Tip: "Sous-chef plates twice"



(d) Unshown Tip: "Server chops once"

Figure 5: Learning beyond Tips. Panels (a)-(c) show the rate at which participants in each condition cross-comply with each offered tip over time in the disrupted configuration; results from participants in the same condition are marked with yellow stars. Panel (d) shows analogous results for a rule that is part of the optimal policy but was not shown as a tip in any condition.

6 Discussion

We have proposed a novel machine learning algorithm for automatically identifying interpretable tips designed to help improve human decision-making. Our large-scale behavioral study demonstrates that the tips inferred by our algorithm can successfully improve human performance at challenging sequential decision-making tasks, speeding up learning by up to three rounds of in-game experience. Furthermore, we find evidence that participants combine our tips with their own experience to discover additional strategies beyond those stated in the tip. In other words, our machine learning algorithm is capable of identifying concise insights and communicating them to humans in a way that expands and improves their knowledge. To the best of our knowledge, our work is the first to empirically demonstrate that machine learning can be used to improve human decision-making.

Furthermore, we provide a number of insights that can aid the design of human-AI interfaces. First, a significant factor in the performance of a tip is whether humans comply with that tip. Prior work has studied compliance from the perspective of *algorithm aversion* (i.e., whether humans trust other humans more than algorithms) [25, 24, 26], as well as interpretability (i.e., whether the human understands the tip) [12, 27, 11]. Our results suggest that human compliance additionally depends on whether humans understand how to operationalize the tip to improve performance, which brings a novel dimension to the design of human-AI interfaces. Second, it takes time for humans to correctly operationalize and adopt the tip—humans need experience to understand why the tip is correct and to discover complementary strategies that further improve their performance. Thus, there is an opportunity for human-AI interfaces to help humans gradually adapt their behavior to improve performance. Third, even tips that are part of the optimal policy can hurt participant performance if they focus on actions that are not consequential; avoiding such tips is important since it can cause participants to lose trust in the AI. We anticipate that human-AI interfaces will become increasingly prevalent as machine learning algorithms are deployed in real-world settings to help humans make consequential decisions, and a better understanding of how to design trustworthy interfaces will be critical to ensuring that these interfaces ultimately improve human performance.

References

[1] Hummy Song, Anita L Tucker, Karen L Murrell, and David R Vinson. Closing the productivity gap: Improving worker productivity through public relative performance feedback and validation of best practices. *Management Science*, 64(6):2628–2649, 2017.

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

[3] 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. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[4] Jake M Hofman, Duncan J Watts, Susan Athey, Filiz Garip, Thomas L Griffiths, Jon Kleinberg, Helen Margetts, Sendhil Mullainathan, Matthew J Salganik, Simine Vazire, et al. Integrating explanation and prediction in computational social science. *Nature*, pages 1–8, 2021.

[5] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[6] Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, 2017.

[7] Benjamin Letham, Cynthia Rudin, Tyler H McCormick, David Madigan, et al. Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model. *The Annals of Applied Statistics*, 9(3):1350–1371, 2015.

[8] Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1721–1730, 2015.

[9] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[10] Sophie Hilgard, Nir Rosenfeld, Mahzarin R Banaji, Jack Cao, and David Parkes. Learning representations by humans, for humans. In *International Conference on Machine Learning*, pages 4227–4238. PMLR, 2021.

[11] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

[12] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

[13] Cristian Bucilu, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, 2006.

[14] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[15] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*, pages 5045–5054. PMLR, 2018.

[16] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *Advances in neural information processing systems*, pages 2494–2504, 2018.

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

[18] Reid McIlroy-Young, Siddhartha Sen, Jon Kleinberg, and Ashton Anderson. Aligning superhuman ai with human behavior: Chess as a model system. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1677–1687, 2020.

[19] Drew Fudenberg, Jon Kleinberg, Annie Liang, and Sendhil Mullainathan. Measuring the completeness of economic models. Technical report, 2021.

[20] Drew Fudenberg and Annie Liang. Predicting and understanding initial play. *American Economic Review*, 109(12):4112–41, 2019.

[21] Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

[22] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[23] Sarah A Wu, Rose E Wang, James A Evans, Joshua B Tenenbaum, David C Parkes, and Max Kleiman-Weiner. Too many cooks: Bayesian inference for coordinating multi-agent collaboration. *Topics in Cognitive Science*, 13(2):414–432, 2021.

[24] Berkeley J Dietvorst, Joseph P Simmons, and Cade Massey. Algorithm aversion: People erroneously avoid algorithms after seeing them err. *Journal of Experimental Psychology: General*, 144(1):114, 2015.

[25] Joseph Eastwood, Brent Snook, and Kirk Luther. What people want from their professionals: Attitudes toward decision-making strategies. *Journal of Behavioral Decision Making*, 25(5):458–468, 2012.

[26] Berkeley J Dietvorst, Joseph P Simmons, and Cade Massey. Overcoming algorithm aversion: People will use imperfect algorithms if they can (even slightly) modify them. *Management Science*, 64(3):1155–1170, 2018.

[27] Isaac Lage, Andrew Slavin Ross, Been Kim, Samuel J Gershman, and Finale Doshi-Velez. Human-in-the-loop interpretability prior. *Advances in neural information processing systems*, 31, 2018.

[28] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[29] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Appendix

The supplementary material is organized as follows. The first section describes our tip inference procedure in more detail, the second section describes additional details on the design of our behavioral study and payments, and the last section provides additional results.

A Tip Inference for Virtual Kitchen-Management Game

MDP formulation: At a high level, the states encode progress towards completing all orders, the actions encode the currently available assignments of subtasks to workers, and the rewards encode the number of ticks taken to complete all orders. Specifically, the states encode (i) which subtasks have been completed so far across all orders, and (ii) which subtask has been assigned to each virtual worker (if any), as well as how many steps remain to complete this subtask. Next, the actions consist of all possible assignments of available subtasks (i.e., have not yet been assigned) to available virtual workers (i.e., not currently working on any subtask). Finally, the reward is -1 at each step, until all orders are completed; thus, the total number of steps taken to complete all orders is the negative reward.

Optimal policies: We now summarize the optimal policy for each scenario. Note that the optimal policy for the understaffed case is more complex and counter-intuitive than the optimal policy for the fully-staffed scenario.

Fully-staffed scenario: In this scenario, the participant has access to all three virtual workers. The optimal number of steps needed to complete this scenario is 20 ticks. The key insights to achieving optimal performance are: (i) all three workers should be assigned to chopping in the first time step, (ii) the chef must cook three of the burgers and the sous-chef must cook one (i.e., the second burger), (iii) the server should never cook and must be kept idle when the third burger becomes available for cooking; they should instead wait to be assigned to plating the first cooked burger, (iv) the chef should never plate, (v) the sous-chef must plate exactly one of the burgers, and (vi) none of the three workers should be left idle except in the previous cases.

Understaffed scenario: In this scenario, the participant has access to only two virtual workers—namely, the sous-chef and the server. The optimal number of steps needed to complete this scenario is 34 ticks. The keys insights to achieving the optimal performance are: (i) both workers should be assigned to chopping in the first time step, (ii) the sous-chef and the server must cook two burgers each, even though the server is very slow at cooking, (iii) the sous-chef must choose chopping over cooking after finishing her first chopping task, (iv) the server's first three tasks must be chopping, cooking, and cooking, in that order, (v) the sous-chef must chop three of the four burgers and the server must chop one, (vi) both workers must plate two burgers each, even though the sous-chef is slower at plating than the server, (vii) the second cooked burger must not be served until the third and fourth burgers are cooked, and (viii) both workers must be kept busy at all times.

Search space of tips: Each tip is actually composed of a set of rules inferred by our algorithm. Recall that our algorithm considers tips in the form of an if-then-else statement that says to take a certain action in a certain state. One challenge is the combinatorial nature of our action space—there can be as many as $k!/(k-m)!$ actions, where m is the number of workers and $k = \sum_{j=1}^{n} k_j$ is the total number of subtasks. The large number of actions can make the tips very specific—e.g., simultaneously assigning three distinct subtasks to three of the virtual workers. Instead, we decompose the action space and consider assigning a single subtask to a single virtual worker. More precisely, we include three features in the predicate ϕ: (i) the subtask being considered, (ii) the order to which the subtask belongs, and (iii) the virtual worker in consideration. Then, our algorithm considers tips of the form

$$\text{if (order} = o \ \wedge \ \text{subtask} = s \ \wedge \ \text{virtual worker} = w) \text{ then (assign } (o, s) \text{ to } w),$$

where o is an order, s is a subtask, and w is a virtual worker.

Even with this action decomposition, we found that these tips are still too complicated for human users to internalize. Thus, we post-process the tips inferred by our algorithm by aggregating over tuples (o, s, w) that have the same s and w.[6] For example, instead of considering two separate tips

$$\begin{aligned}
&\text{if (order} = \text{burger}_1 \ \wedge \ \text{subtask} = \text{cooking} \ \wedge \ \text{virtual worker} = \text{chef)} \\
&\qquad \text{then (assign (burger}_1, \text{cooking) to chef)} \\
&\text{if (order} = \text{burger}_2 \ \wedge \ \text{subtask} = \text{cooking} \ \wedge \ \text{virtual worker} = \text{chef)} \\
&\qquad \text{then (assign (burger}_2, \text{cooking) to chef),}
\end{aligned}$$

we merge them into a tip

$$\text{assign cooking to chef 2 times.}$$

In other words, a tip is a combination of tips $\rho = (\rho_1, ..., \rho_k)$. Finally, the score our algorithm assigns to such a tip is $J(\rho) = \sum_{i=1}^{k} J(\rho_i)$; then, it chooses the tip with the highest score.

Tip inference procedure: Next, we describe how our algorithm computes optimal tips for the kitchen game MDP. In principle, we could use dynamic programming to solve for the optimal value function V^*, and then compute the optimal Q-function based on V^*. However, while our state space is finite, it is still too large for dynamic programming to be tractable. Instead, we use the policy gradient algorithm (which is widely used for model-free

[6] We experimented with *combinations* of tips in exploratory pilots, and found that AMT workers were unable to operationalize and comply with such complex tips even though they might be part of an optimal strategy.

reinforcement learning) as a heuristic to learn an expert policy π_* for our MDP [28]. At a high level, the policy gradient algorithm searches over a family of policies π_θ parameterized by $\theta \in \Theta \subseteq \mathbb{R}^{d_\Theta}$; typically, π_θ is a neural network, and θ is the corresponding vector of neural network parameters. This approach requires featurizing the states in the MDP—i.e., constructing a feature mapping $\phi : S \to \{0,1\}^d$. Then, the neural network policy π_θ takes as input the featurized state $\phi(s)$, and outputs an action $\pi_*(\phi(s)) \in A$ to take in state s.[7] Then, the policy gradient algorithm performs stochastic gradient descent on the objective $J(\pi_\theta)$, and outputs the best policy $\pi_* = \pi_{\theta^*}$. In general, $J(\pi_\theta)$ is nonconvex, so this algorithm is susceptible to local minima, but it has been shown to perform well in practice. For the kitchen game MDP, we use state features including whether each subtask of each order is available, the current status of each worker, and the current time step. We take π_θ to be a neural network with 50 hidden units; to optimize $J(\pi_\theta)$, we take $10,000$ stochastic gradient steps with a learning rate of 0.001.

Once we have computed π_*, we use our tip inference algorithm to learn an estimate $\hat{Q}$ of the Q-function $Q^{(\pi_*)}$ for π_*. We choose $\hat{Q}$ to be a random forest [29]. It operates over the same featurized states as the neural network policy—i.e., it has the form $\hat{Q}(\phi(s), a) \approx Q^{(\pi_*)}(s, a)$. Finally, we apply our algorithm to inferring tips on state-action pairs collected from observing human users playing our game. Since our goal is to help human users improve their performance, we restrict the training dataset to the bottom 25% performing human users. In addition, we apply two post-processing steps to the set of candidate tips. First, we eliminate tips that apply in less than 10% of the (featurized) states that occur in the human dataset. This step eliminates high-variance tips that may have large benefit, but are useful only a small fraction of the time; we omit such tips since our estimates of their quality tend to have very high variance. Second, we eliminate tips that disagree with the expert policy more than 50% of the time—i.e., for a tip (ψ, a), we have $\psi(s) = 1$ and $a \neq \pi^*(s)$ for more than 50% of state-action pairs in the human dataset. This step eliminates tips that have large benefits on average, but frequently offer incorrect advice that can confuse the human user or cause them to distrust our tips.

B Experimental Design Details

We perform separate experiments for each of the two configurations of our game. The high-level structure of our experimental design for each configuration is the same; they differ in terms of when we show tips to the participant and which tips we show. Before starting our game, each participant is shown a set of game instructions and comprehension checks; then, they play a practice scenario twice (with an option to skip the second one). The practice scenario is meant to familiarize participants with the game mechanics and the user interface. In this scenario, they manage three identical chefs to make a single, simple food order. This

[7] To be precise, $\pi_*(\phi(s))$ outputs a probability $\pi_*(a \mid \phi(s))$ for each action $a \in A$ of taking a in state s. Once the neural network has been trained, we always take the action a with the highest probability.

	Chopping meat	Cooking burger	Plating burger
Chef	1	4	6
Souf-chef	2	8	2
Server	3	12	1

Table 1: Skill matrix. Shows the (deterministic) number of time steps each virtual worker requires to complete a given subtask. Participants can learn these values by making assignments in the game.

food order is significantly different than the burger order used in the main game. Then, they proceed to play the scenarios for the current configuration. Table 1 exhibits the number of time steps needed for each of the virtual workers to complete each of the subtasks required to complete a single burger order.

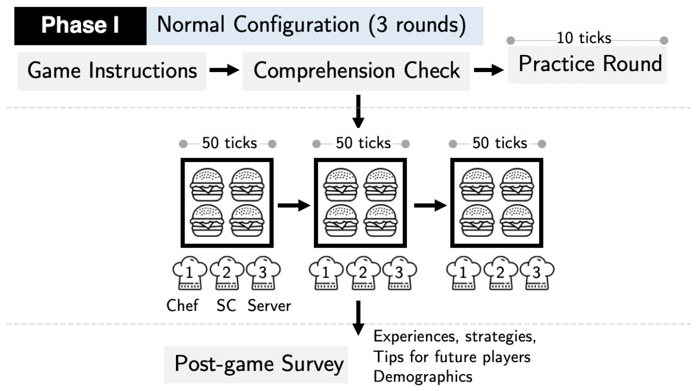
Finally, after completing all scenarios, we give each participant a post-game survey that includes several questions regarding their experience with the game. Each participant receives a participation fee of $0.10 for each round they complete; they also receive a performance-based bonus based on the number of time steps taken to complete each round. The bonus ranges from $0.15 to $0.75 per round. Participants provided informed consent, and all study procedures were approved by our institution's Institutional Review Board.

Phase I: For each configuration, we recruited 200 participants via Amazon Mechanical Turk to play the game. Figure 6 illustrates the study flow for Phase I. As part of the post-game survey, we ask the participants to suggest a tip for future players. In particular, we show each participant a comprehensive list of candidate tips and ask them to select the one they believe would most improve the performance of future players. This list of tips is constructed by merging three types of tips: (i) all possible tips in the search space considered by our algorithm (e.g., "Chef shouldn't plate."), (ii) generic tips that arise frequently in our exploratory user studies (e.g., "Keep everyone busy at all time."), (iii) a small number of manually constructed tips obtained by studying the optimal policy (e.g., "Chef should chop as long as there is no cooking task"). Importantly, this list always contains the top tip inferred using our algorithm.

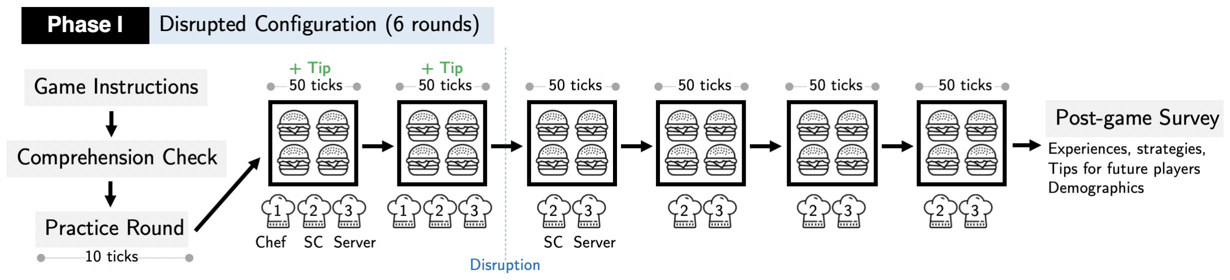
Inferred tips: Next, we use the data from the final round played by the participants to infer tips in three ways: (i) use our tip inference algorithm in conjunction with the data from Phase I, (ii) do the same with the baseline algorithm, and (iii) rank the candidate tips in the post-game survey based on the number of votes by the participants.

For the normal configuration, 183 participants[8] successfully completed the game. The top three tips inferred from each of the sources are reported in Table 2. For the algorithm tip, "Chef should never plate" is selected as it is expected to be the most effective at shortening

[8]They are 34.6 years old on average, 57.38% are female, and 67.73% have at least a two-year degree.



(a) Normal configuration



(b) Disrupted configuration

Figure 6: Study flow for Phase I.

completion time (2.43 steps). For the baseline tip, our naïve algorithm selects "Chef should chop once" as it is the most frequently observed state-action pair in the data. Finally, for the human tip, "Strategically leave some workers idle" received the most votes among the participants (28.42%). It is worth noting that all of the tips most voted by past players are in line with the optimal strategy. The first tip captures the key strategy that some virtual workers should be left idle rather than assigned to a time-consuming task. However, it is less specific than other tips. The second and third tips reflect the information participants could learn from assigning different tasks to different workers during the game: the server spends the most time cooking while the chef spends the most time plating.

Normal	**Tip #1**	Tip #2	Tip #3
Algorithm	**Chef should never plate**	Server plates three times	Server should skip chopping once
Baseline	**Chef should chop once**	Server should plate three times	Sous-chef should plate twice
Human (% voted)	**Strategically leave some workers idle** (28.42%)	Server should never cook (21.31%)	Chef should never plate (13.11%)

Table 2: Top three tips inferred from different sources for the normal configuration.

Disrupted	**Tip #1**	Tip #2	Tip #3
Algorithm	**Server should cook twice**	Sous-chef should plate once	Server should chop once
Baseline	**Sous-chef should plate twice**	Sous-chef should chop three times	Server should cook twice
Human (% voted)	**Server should cook once** (28.48%)	Server should never cook (23.84%)	Keep everyone busy (16.86%)

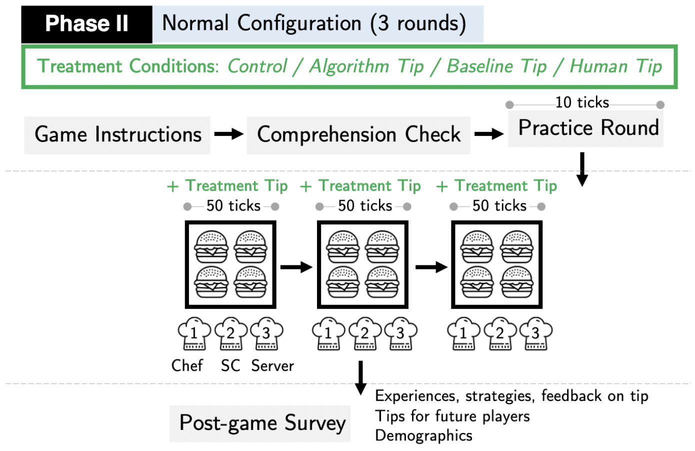
Table 3: Top three tips inferred from different sources for the disrupted configuration.

For the disrupted configuration, 172 participants[9] successfully completed the game. Table 3 reports the top three tips inferred from each of the sources. The best algorithm tip is "Server should cook twice" with the expected completion time reduction of 2.32 steps. The baseline algorithm chooses "Sous-chef should plate twice" and the human tip "Server should cook once" (equivalently "Sous-chef should cook three times") got the most votes. Unlike the normal configuration, the top two human tips are not part of the optimal policy. In the optimal policy, sous-chef and server should each cook twice. The third human tip does align with the optimal policy; however, it is much less specific than the other tips. This highlights the increased difficulty for humans to identify the optimal strategy in the disrupted configuration compared to the normal configuration.

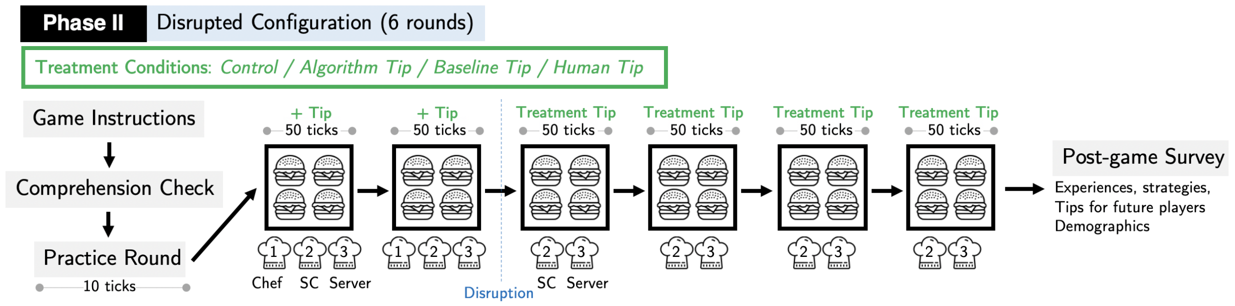
Phase II: Next, we evaluate the effectiveness of each of the inferred tips. In particular, Figure 7 illustrates the study flow for Phase II. In this phase, participants are randomly assigned to one of 4 conditions (control, baseline, algorithm, human). We recruited 350 AMT users to play each condition in each configuration, totaling to 2,800 users. The specific tips we show in each round depends not just on the condition, but also varies from round to round depending on the configuration. For the normal configuration, we show the tip for the current condition in all three rounds. However, for the disrupted configuration, the tip for the current condition is specific to the understaffed scenario. Thus, we only show the tip for the current condition in rounds 3-6; in all conditions, for rounds 1 and 2, we show the tip inferred by our algorithm for the fully-staffed scenario from the normal configuration. By doing so, we ensure that the tip shown during the fully-staffed scenario does not bias our evaluation of the tip for the understaffed scenario.

Pay schemes: *Normal configuration.* In Phase I, participants received \$0.30 as a base pay for their participation. In addition, they could earn a performance-based bonus for each of the three rounds of the game. The optimal (e.g., shortest possible) completion time is 20 time steps and the maximum time allowed is 50 time steps. The bonus is as follows: \$0.75 if completing the round in exactly 20 time steps, \$0.35 if completing the round in 21 to 22 time steps, \$0.15 if completing the round in 23 to 26 time steps, or no bonus otherwise. The total pay ranges from \$0.30 to \$2.55, with a mean of \$1.00, a median of \$0.95, and a standard deviation of \$0.56. In Phase II, which was conducted well into the COVID-19 pandemic,

[9]They are 36.4 years old on average, 61.63% are female, and 77.91% have at least a two-year degree.



(a) Normal configuration



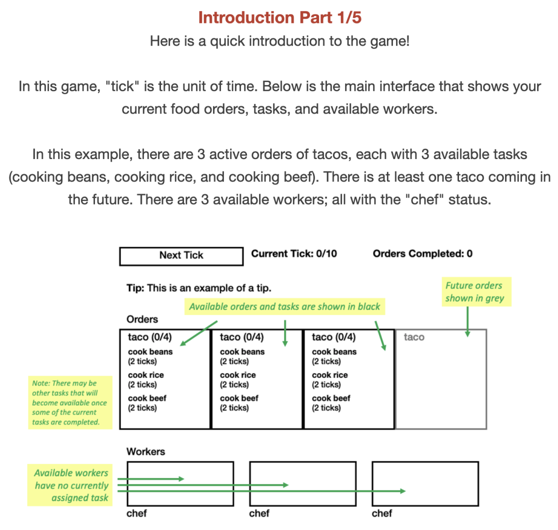
(b) Disrupted configuration

Figure 7: Study flow for Phase II.

we kept the same base pay but slightly increased the tiered bonus: \$1.25 if completing the round in exactly 20 time steps, \$0.60 if completing the round in 21 to 22 time steps, \$0.25 if completing the round in 23 to 26 time steps, or no bonus otherwise. The total pay ranges from \$0.30 to \$4.05, with a mean of \$1.63, a median of \$1.40, and a standard deviation of \$1.03.

Disrupted configuration. In both phases, participants received \$0.60 as a base pay for their participation. In addition, they could earn a performance-based bonus for each of the six rounds of the game. For the first two rounds, in which they managed a fully-staffed kitchen, the bonus scheme is the same as that of Phase I of the normal configuration. For the last four rounds, in which they managed an understaffed kitchen (optimal completion time is 34 time steps), the bonus is as follows: \$0.75 if completing the round in exactly 34 time steps, \$0.35 if completing the round in 35 to 36 time steps, \$0.15 if completing the round in 37 to 38 time steps, or no bonus otherwise. In Phase I, the total pay ranges from \$0.60 to \$3.30, with a mean of \$1.63, a median of \$1.55, and a standard deviation of \$0.60. In Phase II, the total pay ranges from \$0.60 to \$4.50, with a mean of \$1.81, a median of \$1.75, and a standard deviation of \$0.68.

Selected screenshots: Finally, we provide screenshots to illustrate our experimental design. Figures 8 & 9 show the introduction to the task shown to participants explaining various concepts in the game. Figures 10 & 11 show instructions for the fully-staffed and understaffed scenarios, respectively, shown to participants. Finally, Figure 12 shows the payment information shown to the participants.



Introduction Part 1/5

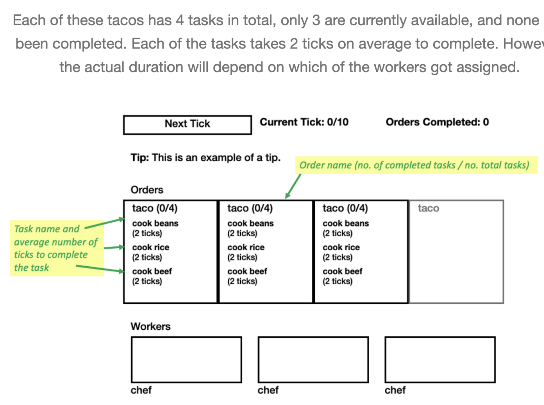
Here is a quick introduction to the game!

In this game, "tick" is the unit of time. Below is the main interface that shows your current food orders, tasks, and available workers.

In this example, there are 3 active orders of tacos, each with 3 available tasks (cooking beans, cooking rice, and cooking beef). There is at least one taco coming in the future. There are 3 available workers; all with the "chef" status.

Next Tick
Current Tick: 0/10
Orders Completed: 0
Tip: This is an example of a tip.
Future orders shown in grey
Available orders and tasks are shown in black
Orders
taco (0/4)
cook beans (2 ticks)
cook rice (2 ticks)
cook beef (2 ticks)
taco
Note: There may be other tasks that will become available once some of the current tasks are completed.
Workers
Available workers have no currently assigned task
chef

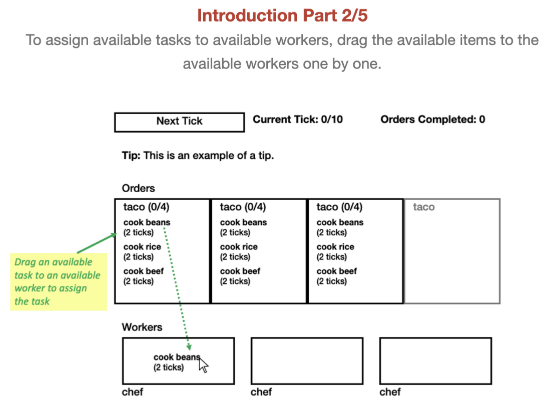
(a) Introduction to the interface



Each of these tacos has 4 tasks in total, only 3 are currently available, and none has been completed. Each of the tasks takes 2 ticks on average to complete. However, the actual duration will depend on which of the workers got assigned.

Next Tick
Current Tick: 0/10
Orders Completed: 0
Tip: This is an example of a tip.
Order name (no. of completed tasks / no. total tasks)
Orders
taco (0/4)
cook beans (2 ticks)
cook rice (2 ticks)
cook beef (2 ticks)
taco
Task name and average number of ticks to complete the task
Workers
chef

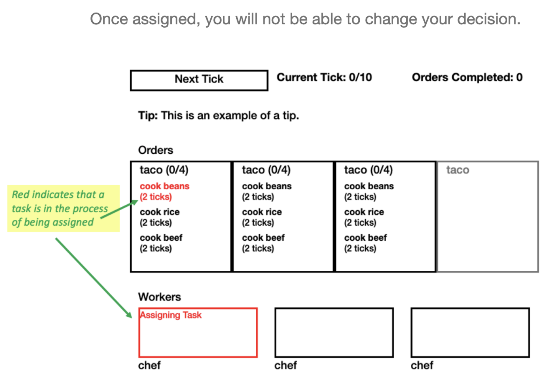
(b) Introduction to the subtasks



Introduction Part 2/5

To assign available tasks to available workers, drag the available items to the available workers one by one.

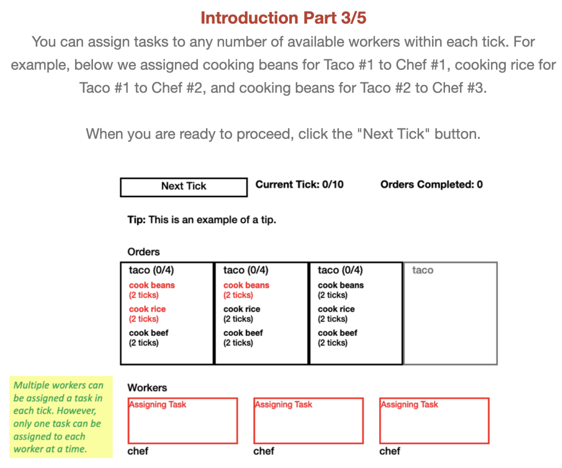
(c) Introduction to task assignment



Once assigned, you will not be able to change your decision.

(d) Introduction to task assignment (cont.)

Figure 8: Screenshots of the game introduction.



Introduction Part 3/5

You can assign tasks to any number of available workers within each tick. For example, below we assigned cooking beans for Taco #1 to Chef #1, cooking rice for Taco #1 to Chef #2, and cooking beans for Taco #2 to Chef #3.

When you are ready to proceed, click the "Next Tick" button.

Next Tick | **Current Tick: 0/10** | **Orders Completed: 0**

Tip: This is an example of a tip.

Orders

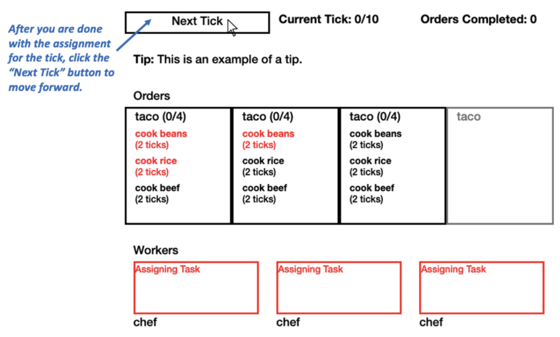
taco (0/4)	taco (0/4)	taco (0/4)	taco
cook beans (2 ticks) cook rice (2 ticks) cook beef (2 ticks)	cook beans (2 ticks) cook rice (2 ticks) cook beef (2 ticks)	cook beans (2 ticks) cook rice (2 ticks) cook beef (2 ticks)	

Multiple workers can be assigned a task in each tick. However, only one task can be assigned to each worker at a time.

Workers

Assigning Task	Assigning Task	Assigning Task
chef	chef	chef

(a) Introduction to task assignment (cont.)



After you are done with the assignment for the tick, click the "Next Tick" button to move forward.

Next Tick | **Current Tick: 0/10** | **Orders Completed: 0**

Tip: This is an example of a tip.

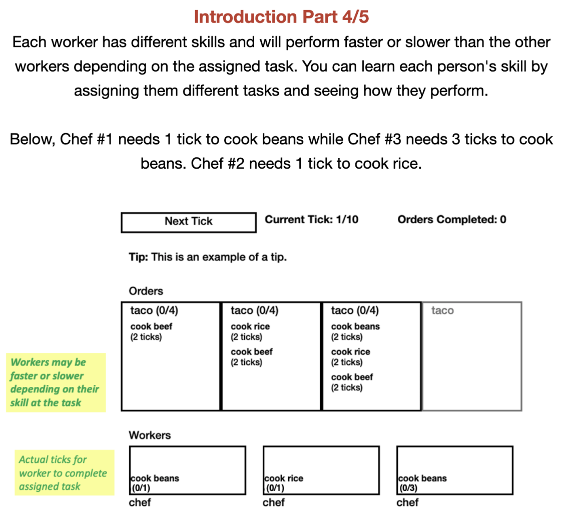
Orders

taco (0/4)	taco (0/4)	taco (0/4)	taco
cook beans (2 ticks) cook rice (2 ticks) cook beef (2 ticks)	cook beans (2 ticks) cook rice (2 ticks) cook beef (2 ticks)	cook beans (2 ticks) cook rice (2 ticks) cook beef (2 ticks)	

Workers

Assigning Task	Assigning Task	Assigning Task
chef	chef	chef

(b) Introduction to task assignment (cont.)



Introduction Part 4/5

Each worker has different skills and will perform faster or slower than the other workers depending on the assigned task. You can learn each person's skill by assigning them different tasks and seeing how they perform.

Below, Chef #1 needs 1 tick to cook beans while Chef #3 needs 3 ticks to cook beans. Chef #2 needs 1 tick to cook rice.

Next Tick | **Current Tick: 1/10** | **Orders Completed: 0**

Tip: This is an example of a tip.

Orders

taco (0/4)	taco (0/4)	taco (0/4)	taco
cook beef (2 ticks)	cook rice (2 ticks) cook beef (2 ticks)	cook beans (2 ticks) cook rice (2 ticks) cook beef (2 ticks)	

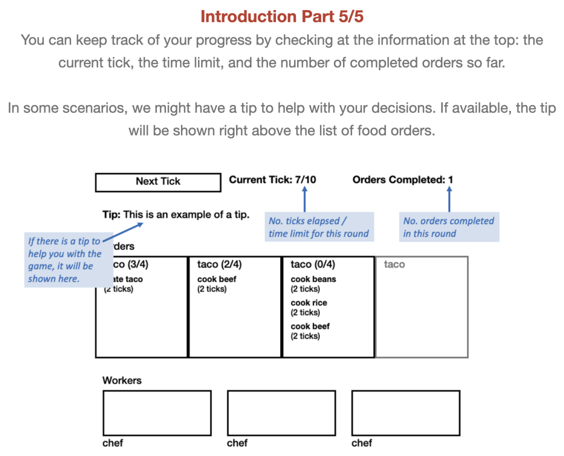
Workers may be faster or slower depending on their skill at the task

Workers

Actual ticks for worker to complete assigned task

cook beans (0/1)	cook rice (0/1)	cook beans (0/3)
chef	chef	chef

(c) Introduction to workers' skill levels



Introduction Part 5/5

You can keep track of your progress by checking at the information at the top: the current tick, the time limit, and the number of completed orders so far.

In some scenarios, we might have a tip to help with your decisions. If available, the tip will be shown right above the list of food orders.

Next Tick | **Current Tick: 7/10** | **Orders Completed: 1**

Tip: This is an example of a tip.

No. ticks elapsed / time limit for this round

No. orders completed in this round

If there is a tip to help you with the game, it will be shown here.

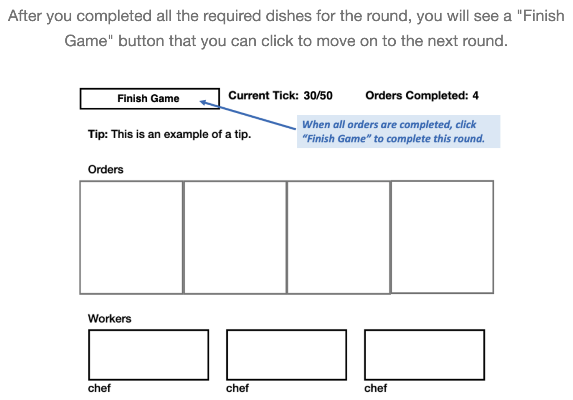
Orders

taco (3/4)	taco (2/4)	taco (0/4)	taco
plate taco (2 ticks)	cook beef (2 ticks)	cook beans (2 ticks) cook rice (2 ticks) cook beef (2 ticks)	

Workers

chef	chef	chef

(d) Introduction to the tip



After you completed all the required dishes for the round, you will see a "Finish Game" button that you can click to move on to the next round.

Finish Game | **Current Tick: 30/50** | **Orders Completed: 4**

Tip: This is an example of a tip.

When all orders are completed, click "Finish Game" to complete this round.

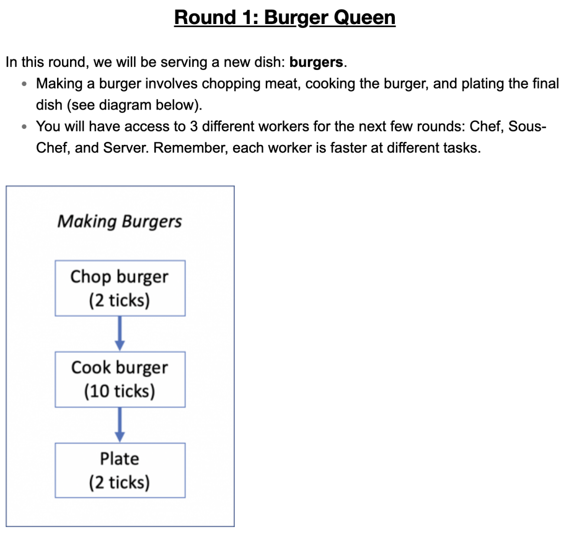
Orders

Workers

chef	chef	chef

(e) Introduction to round completion

Figure 9: Screenshots of the game introduction (continued).

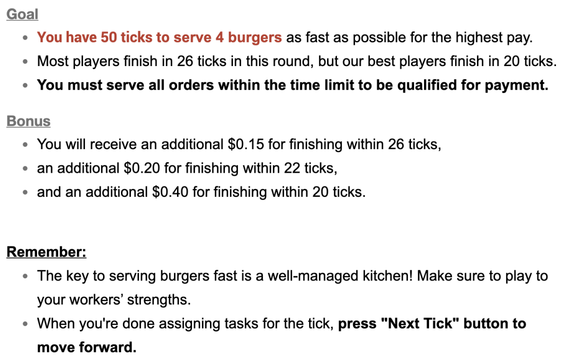


Round 1: Burger Queen

In this round, we will be serving a new dish: **burgers**.

- Making a burger involves chopping meat, cooking the burger, and plating the final dish (see diagram below).
- You will have access to 3 different workers for the next few rounds: Chef, Sous-Chef, and Server. Remember, each worker is faster at different tasks.

(a) Burger's subtasks and available workers



Goal

- You have 50 ticks to serve 4 burgers as fast as possible for the highest pay.
- Most players finish in 26 ticks in this round, but our best players finish in 20 ticks.
- **You must serve all orders within the time limit to be qualified for payment.**

Bonus

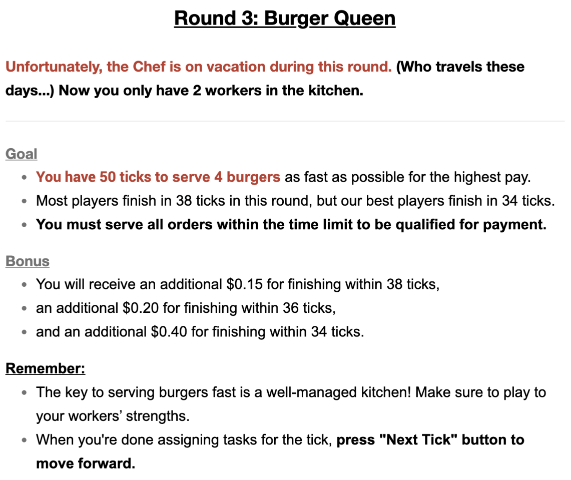
- You will receive an additional $0.15 for finishing within 26 ticks,
- an additional $0.20 for finishing within 22 ticks,
- and an additional $0.40 for finishing within 20 ticks.

Remember:

- The key to serving burgers fast is a well-managed kitchen! Make sure to play to your workers' strengths.
- When you're done assigning tasks for the tick, **press "Next Tick" button to move forward.**

(b) Goal, incentives, and reminder

Figure 10: Screenshots of the instructions for the fully-staffed scenario.



Round 3: Burger Queen

Unfortunately, the Chef is on vacation during this round. **(Who travels these days...) Now you only have 2 workers in the kitchen.**

Goal

- You have 50 ticks to serve 4 burgers as fast as possible for the highest pay.
- Most players finish in 38 ticks in this round, but our best players finish in 34 ticks.
- **You must serve all orders within the time limit to be qualified for payment.**

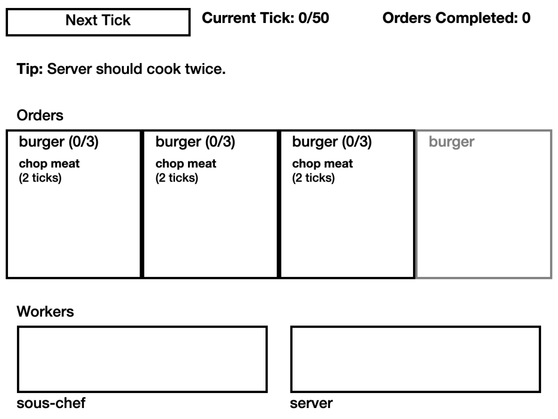
Bonus

- You will receive an additional $0.15 for finishing within 38 ticks,
- an additional $0.20 for finishing within 36 ticks,
- and an additional $0.40 for finishing within 34 ticks.

Remember:

- The key to serving burgers fast is a well-managed kitchen! Make sure to play to your workers' strengths.
- When you're done assigning tasks for the tick, **press "Next Tick" button to move forward.**

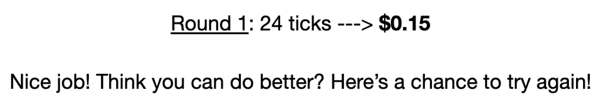
(a) Updated instructions following the in-game disruption



Tip: Server should cook twice.

(b) Game interface (with the algorithm tip)

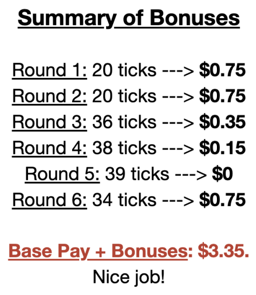
Figure 11: Screenshots of the instructions for the understaffed scenario.



Round 1: 24 ticks ---> **$0.15**

Nice job! Think you can do better? Here's a chance to try again!

(a) Individual round pay information



Summary of Bonuses

Round 1: 20 ticks ---> **$0.75**
Round 2: 20 ticks ---> **$0.75**
Round 3: 36 ticks ---> **$0.35**
Round 4: 38 ticks ---> **$0.15**
Round 5: 39 ticks ---> **$0**
Round 6: 34 ticks ---> **$0.75**

Base Pay + Bonuses: $3.35.
Nice job!

(b) Summary of total pay

Figure 12: Screenshots of the pay information.

C Experimental Results Details

Table 4 exhibits the demographic and gameplay information of the participants across our studies. The four groups of participants are not significantly different from one another, except that those playing the disrupted configuration spent slightly longer time to complete the game and found the game to be slightly more difficult, compared to the normal configuration. Tables 5 and 6 show the average performance within each round across two phases and treatment conditions for normal and disrupted configurations, respectively.

	Phase I: Normal	Phase II: Normal	Phase I: Disrupted	Phase II: Disrupted
Total	183	1,317	172	1,011
Mean age [range]	34.6 [18, 76]	33.3 [18, 74]	34 [19, 76]	34.9 [16, 84]
Female	57.38%	51.03%	61.63%	60.14%
$\geq$ 2-year degree	73.22%	67.73%	77.91%	70.43%
Median duration	18.82 minutes	20.50 min	27.80 min	26.80 min
Found the game difficult	60.66%	50.04%	70.93%	64.99%
Never played similar games	45.36%	43.82%	46.51%	43.52%

Table 4: Participants' demographic and gameplay information.

	Phase I	Phase II: Control	Phase II: Algorithm	Phase II: Baseline	Phase II: Human
Round 1	25.73	26.03	25.04	26.01	26.16
Round 2	25.02	24.46	23.29	24.71	25.06
Round 3	23.74	23.86	22.99	24.04	24.06

Table 5: Average performance by treatment condition and round (normal configuration).

	Phase I	Phase II: Control	Phase II: Algorithm	Phase II: Baseline	Phase II: Human
Round 1	24.35	24.26	24.18	24.77	24.64
Round 2	22.87	22.38	22.95	23.08	22.69
Round 3	38.75	38.73	38.19	38.74	38.26
Round 4	38.39	38.21	37.77	38.38	37.85
Round 5	38.25	38.17	37.25	38.42	37.62
Round 6	37.96	37.82	37.14	38.37	37.62

Table 6: Average performance by treatment condition and round (disrupted configuration).

Participant comments on the provided tips: The effectiveness of a tip depends in part on whether the participants follow it. In general, we observe that the compliance to more intuitive tips is significantly higher than compliance to counter-intuitive tips. In particular, for the disrupted configuration, the tips inferred from our algorithm are counter-intuitive since they ask the server to cook twice despite the fact that the server is slow at cooking. Here, we further discuss comments the participants provided regarding the tips they were shown. In the post-game survey, we asked the participants the following question: "What did you think about the tip for these last [three/four] rounds and how did you incorporate it in your strategy?" We manually code these responses; Tables 7 and 8 exhibit the breakdown of participants in each condition based on the coded responses for normal and disrupted configurations, respectively. Note that participants were not informed of the source of the tips, so variation in compliance is entirely due to the content of the tips. We also note that we verified that the participants saw each tip by asking them to write the content of the tips in the survey.

Normal	Algorithm	Baseline	Human
	"Chef shouldn't plate"	*"Chef chops once"*	*"Leave some idle"*
(N1) Positive	25.87%	16.33%	29.23%
(N2) Negative	4.20%	5.44%	1.92%
(N3) Neutral	53.85%	51.70%	48.08%

Table 7: Participants' coded feedback on the provided tips (normal configuration).

We first observe that for both configurations, a substantially larger fraction of participants in the human condition responded positively to the tips compared to the other conditions. Similarly, a substantially smaller fraction of those in the human condition viewed the tips negatively. These results suggest that human participants were effective at selecting a tip that would be also accepted by other humans, potentially due to such tip matching humans' intuition in general. On the other hand, counter-intuitive tips such as algorithm and baseline tips in the disrupted configuration received substantially more negative feedback. Both tips contradict the optimal strategy from the fully-staffed scenario in which the server who is the slowest at cooking should not be assigned to cook; instead, the server should be assigned

Disrupted	Algorithm *"Server cooks twice"*	Baseline *"Sous-chef plates twice"*	Human *"Server cooks once"*
(D1) Positive	23.10%	10.19%	25.87%
(D2) Negative	33.10%	37.58%	16.78%
(D3) Neutral	32.76%	42.99%	47.90%

Table 8: Participants' coded feedback on the provided tips (disrupted configuration).

most of the plating tasks. The negative sentiment towards these two tips could explain the much lower compliance rates we observed in Figure 4b. The improvement in performance over time implied that it took the participants some time and effort to correctly incorporate these tips into their workflow. To better understand these trends, we next present selected excerpts from the participant comments.

(D1-A) Positive comments on the algorithm tip in the disrupted configuration:

- "It was very helpful. It made me focus on making sure the server cooked more even if that was not his obvious strength."
- "I ignored the tip at first, but later I used the tip and it helped me complete the tasks quickly."
- "At first I didn't follow it because it seemed counter intuitive since they're slow. But then I had trouble, so I tried it and came out ahead."
- "I did not listen to it at first because I didn't believe that it would actually help but it did."
- "The tip was helpful. Without it, I think I would have tried to complete the task without the Server cooking, which would have left someone idle for a long time."

(D2-A) Negative comments on the algorithm tip in the disrupted configuration:

- "I think it was a bad tip. I couldn't figure out how to incorporate it successfully."
- "Seemed counterintuitive."
- "It did not help me. I did not use it for round 1, I used it for round 2 and it made me do worse, so round 3 I tried it again and was still unable to do well, so the last round I ignored the tip."
- "I don't think it helped. I thought having the sous chef cook 3 times would take too long and the point at which I tried it, I decided last minute to have the server cook twice. So I don't think it told me anything useful."
- "It was not needed since the server took so much longer to cook."

(D1-H) Positive comments on the human tip in the disrupted configuration:

- "It seemed pretty much essential to have server cook once."

- “I thought it was smart and I used it exclusively.”
- “It was accurate, and I implemented the tip.”
- “I felt that tip was valid, as the server primarily is useful plating/chopping. I only had him cook once.”
- “It helped because she could cook one burger but any more than that and your ticks would be too high.”

(D2-H) Negative comments on the human tip in the disrupted configuration:

- “I used the tip but I don’t think it was helpful. The server took long to cook.”
- “I don’t agree with this tip.”
- “It stunk honestly. The server takes forever to cook.”

As can be seen, many users felt the human tip to be valid and accurate since it was closer to their intuition while disagreeing with tips that they found counter-intuitive. To some degree, the human tip was also counter-intuitive since it asks the server to cook once instead of not at all. As a consequence, compliance to the tip suffered. As discussed earlier, because the participants do not know the source of each tip, these differences are purely due to the content of the tips. As a consequence, to achieve high compliance, it is not sufficient for the participant to understand the action suggested by the tip; instead, they have to understand why the suggested action helps improve performance and how to correctly incorporate it into their workflow.

Hypothetical disruption: In the post-game survey of both phases of the normal configuration, participants were asked to imagine a hypothetical understaffed scenario where the chef was no longer available in the kitchen and select the best tip that they believed would most help improve performance in such disruption. Note that these participants did not experience a disruption during their gameplay. The list of tip presented to them is the same as the one offered to the participants in the disruption configuration. Consistently in both phases, the tip that received the most votes is “Server shouldn’t cook”. Again, this is likely due to the fact that, after three rounds of managing the virtual kitchen under the fully-staffed scenario, the participants potentially learned the optimal policy that the server should not be assigned to cook any burger. Without the actual experience of managing the disruption, they appeared to be biased towards their strategy learned in the fully-staffed scenario, which felt more intuitive to them. This observation highlights one of the key insights of our study that humans’ intuition could be far away from the optimal policy, making them less likely to comply to the counter-intuitive tip inferred from our algorithm.