# AssistanceZero: Scalably Solving Assistance Games

**Anonymous Authors**[1]

## Abstract

Assistance games are a promising alternative to reinforcement learning from human feedback (RLHF) for training AI assistants. Assistance games resolve key drawbacks of RLHF, like incentives for deceptive behavior, by explicitly modeling the interaction between assistant and user as a two-player game where the assistant cannot observe the user's goal. Despite their potential, assistance games have only been explored in simple settings. Scaling them to more complex environments is difficult because it requires both accurately modeling human users' behavior and determining optimal actions in uncertain sequential decision-making problems. We tackle these challenges by introducing a deep reinforcement learning (RL) algorithm called AssistanceZero for solving assistance games and applying it to a Minecraft-based assistance game with over $10^{400}$ possible goals. We show that an AssistanceZero assistant effectively assists simulated humans in achieving unseen goals and outperforms assistants trained with imitation learning and model-free RL. Our results suggest that assistance games are more tractable than previously thought, and that they are an effective framework for assistance at scale.

## 1. Introduction

Reinforcement learning from human feedback (RLHF) and its variants have become the dominant paradigm for training general AI assistants. RLHF involves fine-tuning pretrained foundation models to take actions (i.e., produce responses) that are preferred by human annotators according to criteria like helpfulness and harmlessness (Bai et al., 2022).

However, RLHF-trained assistants have a number of drawbacks. In particular, the objective in RLHF—generating single actions preferred by annotators—is not always aligned with the overall goal of effectively assisting users. For example, imagine a coding assistant trained with RLHF that interacts with a user in a pair-programming setup. One misalignment between RLHF and assisting the user is that convincing deceptive actions may be rated highly by annotators but will ultimately cause harm (Lang et al., 2024). For example, annotators may accidentally choose subtly buggy code, causing the assistant to introduce bugs that are difficult to detect during deployment. This issue will only become more significant as AI systems become more intelligent, since their outputs may become harder for humans to reliably evaluate. Furthermore, RLHF does not encourage models to maintain *uncertainty* about a user's goals. An assistant that accounts for this uncertainty might ask clarifying questions and preserve option value (the ability to help with a variety of possible goals). Instead, since RLHF-based models are training on single-turn responses, the primary incentive is to immediately act based on a best-guess about the user's goal. For example, when considering a function whose purpose is ambiguous, the coding assistant might choose an incorrect interpretation and implement it without consulting the user. Finally, RLHF does not explicitly account for the interactive, collaborative nature of assistance. When an AI assistant and user interact in a shared environment, it is often better for the assistant to take actions that *complement* the user's actions rather than *replace* them. For example, it may be more helpful for the coding assistant to look for existing bugs or write helper functions. Instead, current assistants like GitHub Copilot (Chen et al., 2021) try to predict what the user will write next and write it for them. Since RLHF does not consider the joint effects of the assistant's and user's actions, or their effects on one another, it may not produce the most helpful assistant.

An alternative paradigm for training AI assistants is *assistance games* (Fern et al., 2014; Hadfield-Menell et al., 2016; Shah et al., 2020). Assistance games avoid the aforementioned drawbacks of RLHF by explicitly accounting for both the interactive nature of assistance and uncertainty about the user's goal. In particular, an assistance game is a two-player game in which an assistant and a user take actions in a shared environment. The two agents share a reward function, but crucially the assistant is initially uncertain about it. Assistance games remove incentives for deception since the assistant's performance depends on the true latent reward

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

function, rather than human feedback. They also incentivize the assistant to interact with the user to resolve its uncertainty about the reward function. Thus, solving an assistance game can be viewed as a form of "meta-learning" where the assistant learns how to learn about the user's goals. Finally, solving assistance games results in assistants whose actions complement the user's to achieve optimal joint performance.

Given the advantages of assistance games, why do they remain a poorly studied method for training AI assistants? While assistance games have been used to solve very small-scale problems, there are two major challenges in applying them to more realistic settings. First, there are many cooperative equilibria in an assistance game, and humans are unlikely to exactly play *any* of them. If the AI assistant fails to account for human irrationality and conventions, it could perform poorly with real humans (Carroll et al., 2020). Second, the AI assistant must maintain uncertainty over reward functions and reason under that uncertainty, which deep learning-based AI systems struggle to do (Gleave & Irving, 2022). Furthermore, solving sequential decision-making problems with uncertainty is considered computationally intractable in many cases (Papadimitriou & Tsitsiklis, 1987; Madani et al., 2003). While prior work on interacting with humans in uncertain environments has been limited to small amounts of unstructured uncertainty (Hu et al., 2020), real human preferences are complex and structured.

We tackle these challenges and show that complex assistance games can be tractably solved. We overcome the first challenge by fixing a reward-conditioned human policy and seeking to find a best-response AI policy. This reduces the assistance game to a partially-observable Markov decision process (POMDP), which unlike a game has a well-defined solution. We address the second challenge by developing a hybrid learning–planning approach called AssistanceZero to effectively solve the assistance POMDP. AssistanceZero extends AlphaZero (Silver et al., 2017) by predicting the unseen goal and human actions, allowing it to effectively plan how to best assist the human.

We test AssistanceZero in a new environment, the Minecraft Building Assistance Game (MBAG), in which an AI assistant must help a human build a goal structure in a Minecraft-based environment without prior knowledge of the goal (Figure 1). The assistant must interact with the user to learn about their reward function (which in this case has a one-to-one relationship with the goal structure) and help them optimize it. The distribution over goal structures in MBAG is complex but structured, reflecting human preferences in other domains. Creating an effective assistant in MBAG is a major challenge because it has a far larger number of possible goals than in prior work (over $10^{400}$, compared to less than 20). Despite this challenge, we show that assistants trained with AssistanceZero are highly effective at collaborating with simulated humans. We also compare AssistanceZero to other methods of solving assistance games and other paradigms for building AI assistants. We find that AssistanceZero greatly outperforms a highly optimized PPO baseline and imitation-learning based methods. Finally, we also shed light on the choice of human policy by training a number of human models and evaluating their accuracy at predicting real human behavior in MBAG. Our results suggest that assistance games are tractable to scale and can be a superior framework for training helpful assistants in challenging environments.


*Figure 1.* The Minecraft Building Assistance Game (MBAG), in which we test our AssistanceZero algorithm for scalably solving complex assistance games. See Section 4 for a full description.

Our contributions may be summarized as: we introduce AssistanceZero for tractably solving complex assistance games; we demonstrate that it can be used to solve MBAG, an assistance game with exponentially more possible goals than those in previous work; and, we empirically investigate a number of human models for MBAG.

## 2. Background and Related Work

We begin by introducing the assistance game formalism and surveying related work. An assistance game is a Markov game in which two players, the human $\mathbf{H}$ and the assistant $\mathbf{R}$, interact to optimize a shared reward function. It consists of a state space $\mathcal{S}$, action spaces $\mathcal{A}^{\mathbf{H}}$ and $\mathcal{A}^{\mathbf{R}}$ for the human and assistant, a set of possible reward parameters $\Theta$, and a discount factor $\gamma \in [0, 1]$. Reward parameters and an initial state are sampled from a predefined distribution $p(s_1, \theta)$. At each timestep $t = 1, \ldots, T$, both agents select actions $a_t^{\mathbf{H}} \in \mathcal{A}^{\mathbf{H}}, a_t^{\mathbf{R}} \in \mathcal{A}^{\mathbf{R}}$; receive shared reward $R(s_t, a_t^{\mathbf{H}}, a_t^{\mathbf{R}}; \theta)$; and the environment transitions to state $s_{t+1}$ according to a transition distribution $p(s_{t+1} \mid s_t, a_t^{\mathbf{H}}, a_t^{\mathbf{R}})$.

A human policy $\pi_{\mathbf{H}} : \mathcal{S} \times \Theta \to \Delta(\mathcal{A}^{\mathbf{H}})$ defines a distribution over actions $\pi_{\mathbf{H}}(a^{\mathbf{H}} \mid s, \theta)$ given an environment state and reward parameters. An assistant policy

$\pi_{\mathbf{R}} : (\mathcal{S} \times \mathcal{A}^{\mathbf{H}} \times \mathcal{A}^{\mathbf{R}})^* \times \mathcal{S} \to \Delta(\mathcal{A}^{\mathbf{R}})$ defines a distribution over actions $\pi_{\mathbf{R}}(a_t^{\mathbf{R}} \mid h_t)$ conditioned on the state-action history up until the current timestep: $h_t = (s_1, a_1^{\mathbf{H}}, a_1^{\mathbf{R}}, \ldots, s_{t-1}, a_{t-1}^{\mathbf{H}}, a_{t-1}^{\mathbf{R}}, s_t)$. Note that the assistant policy is *not* conditioned on the reward parameters since it cannot observe them. While in general a human policy might also depend on $h_t$, for simplicity we assume that $\pi_{\mathbf{H}}$ is only conditioned on $(s, \theta)$; previous results show there is an optimal human policy conditioned only on $(s, \theta)$ (Hadfield-Menell et al., 2016). Given a pair of policies $(\pi_{\mathbf{H}}, \pi_{\mathbf{R}})$, we can define their joint expected return as

$$J(\pi_{\mathbf{H}}, \pi_{\mathbf{R}}) = \mathbb{E}\left[\sum_{t=1}^{T} \gamma^t R(s_t, a_t^{\mathbf{H}}, a_t^{\mathbf{R}}; \theta)\right],$$

the expected discounted sum of their shared reward, where $(s_1, \theta) \sim p(s_1, \theta)$; $a_t^{\mathbf{H}} \sim \pi_{\mathbf{H}}(a^{\mathbf{H}} \mid s_t, \theta)$; $a_t^{\mathbf{R}} \sim \pi_{\mathbf{R}}(a^{\mathbf{R}} \mid h_t)$; and $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t^{\mathbf{H}}, a_t^{\mathbf{R}})$. For a fixed human policy $\pi_{\mathbf{H}}$, we define a *best response* to it as an assistant policy $\pi_{\mathbf{R}}$ that maximizes $J(\pi_{\mathbf{H}}, \pi_{\mathbf{R}})$.

**Related work** Assistance games were introduced by Fern et al. (2014) and Hadfield-Menell et al. (2016) under the names "hidden-goal MDPs" and "cooperative inverse reinforcement learning." A few prior works have explored small-scale assistance games (Dragan & Srinivasa, 2013; Javdani et al., 2015; Malik et al., 2018; Fisac et al., 2020; Woodward et al., 2020; Zhi-Xuan et al., 2024) with around ten or fewer discrete reward parameters. We aim to scale assistance games to much larger structured reward parameter spaces, similar to the goals real humans have when interacting with assistants; in our environment $|\Theta| \approx 10^{400}$.

Our approach to solving assistance games builds on techniques for scalably solving games (Silver et al., 2017; Brown et al., 2020; Hu et al., 2021a), modeling human behavior (Carroll et al., 2020; Laidlaw & Dragan, 2021; Yang et al., 2022; Jacob et al., 2022), and training effective collaborative agents (Stone et al., 2010; Hu et al., 2020; Treutlein et al., 2021; Strouse et al., 2021; Hu et al., 2021b; Bakhtin et al., 2022). Minecraft and Minecraft-like environments have been previously used as testbeds for assistance and collaboration (Szlam et al., 2019; Gray et al., 2019; Bara et al., 2021; Skrynnik et al., 2022; Kiseleva et al., 2022; Zholus et al., 2022; Mehta et al., 2024) as well as for general interactive learning (Kanervisto et al., 2022; Baker et al., 2022; Fan et al., 2022; Milani et al., 2023; Wang et al., 2023).

## 3. Solving Assistance Games with AssistanceZero

Solving an assistance game requires finding an assistant policy $\pi_{\mathbf{R}}$ that performs well with real users. Shah et al. (2020) propose to fix a human policy $\pi_{\mathbf{H}}(a^{\mathbf{H}} \mid s, \theta)$ (i.e., human model) and find a best-response policy $\pi_{\mathbf{R}}$. However, these steps are difficult to scale to complex settings. Developing robust and accurate human models is an ongoing area of research, and simple models of human behavior like Boltzmann rationality fail to predict human behavior beyond the smallest of environments (Laidlaw & Dragan, 2021). Shah et al. (2020) show that finding a best response to a fixed human model can be reduced to solving a POMDP, which we call an *assistance POMDP*. Unfortunately, large-scale POMDPs are notoriously difficult to solve.

We explore how to find good human models in Section 5 and focus here on solving assistance POMDPs. We aim to do this with deep reinforcement learning (DRL) algorithms, since they are a scalable technique for solving complex sequential decision-making problems. We apply DRL by following the training procedure from Woodward et al. (2020). Each of several episodes of data are collected by sampling reward parameters $\theta \sim p(\theta)$ and rolling out the remainder of the episode according to the fixed human model $\pi_{\mathbf{H}}$ and the current assistant policy $\pi_{\mathbf{R}}^{\phi}$ with parameters $\phi$. Next, the parameters $\phi$ are updated according to some loss function defined over the episodes, and the process repeats by collecting more data. For example, proximal policy optimization (PPO) (Schulman et al., 2017) can be applied to an assistance POMDP; it uses the collected data to estimate $\nabla_{\phi} J(\pi_{\mathbf{H}}, \pi_{\mathbf{R}}^{\phi})$ and then updates $\phi$ with gradient ascent.

While PPO has shown promise in partially observable and multi-agent settings (Yu et al., 2022), we find that it struggles to solve assistance POMDPs, which require reasoning about structured uncertainty over a potentially large space of reward parameters $\theta \in \Theta$. Solving an assistance POMDP requires balancing learning more about $\theta$ and using that information to help the human. We generally found that applying vanilla PPO to assistance POMDPs results in an assistant policy that does nothing. Thus, we turned to a different DRL algorithm: AlphaZero (Silver et al., 2017). AlphaZero has achieved superhuman performance in complex competitive games like Go and chess, but it is not clear if it is applicable to solving assistance POMDPs.

We propose an extension of AlphaZero, which we call AssistanceZero, that can effectively solve assistance POMDPs better than even a carefully-tuned PPO baseline trained with auxiliary losses. Similarly to AlphaZero, AssistanceZero chooses actions using a variant of Monte Carlo tree search (MCTS) (Kocsis & Szepesvári, 2006). MCTS builds a search tree by simulating the results of taking different sequences of actions in the current state. It requires both the *reward* and the *next state* resulting from an action. However, in an assistance POMDP, neither is known: the next state depends on both the assistant's and human's actions, not just the assistant's action, and the reward $R(s, a^{\mathbf{H}}, a^{\mathbf{R}}; \theta)$ depends on the reward parameters $\theta$, which are not visible to the assistant. To overcome these challenges, AssistanceZero employs a recurrent neural network with parameters $\phi$ that takes as input a state-action history $h$ and has four heads:

a *policy* head $\pi^\phi(a^{\mathbf{R}} \mid h)$, a *value* head $\hat{V}^\phi(h)$, a *reward parameter prediction* head $\hat{p}^\phi(\theta \mid h)$, and a *human action prediction* head $\hat{p}^\phi(a^{\mathbf{H}} \mid h)$. The policy and value heads select actions and evaluate the value of states, respectively. The reward parameter and human action prediction heads predict distributions over $\theta$ and $a^{\mathbf{H}}$ so that MCTS can estimate the reward and next state given a selected action.

To train the AssistanceZero network, we collect episodes by selecting assistant actions using MCTS with the current network parameters; then, the four heads are trained using separate loss terms. As in AlphaZero, the policy head is updated to minimize the KL divergence towards the policy output from MCTS, and the value head to minimize the squared error with the reward-to-go. The reward parameter and human action prediction heads are trained with negative log-likelihood loss to predict $\theta$ and $a^{\mathbf{H}}$, respectively. The full AssistanceZero loss can be written for an episode as

$$\begin{aligned} L(\phi) = \frac{1}{T} \sum_{t=1}^{T} \Big[ & \lambda_{\text{policy}} D_{\text{KL}} \left( \pi_t^{\text{MCTS}} \| \pi^\phi(\cdot \mid h_t) \right) \\ & + \lambda_{\text{value}} \left( \hat{V}^\phi(h_t) - \sum_{t'=t}^{T} \gamma^{t'-t} R(s_{t'}, a_{t'}^{\mathbf{H}}, a_{t'}^{\mathbf{R}}; \theta) \right)^2 \\ & - \lambda_{\text{reward}} \log \hat{p}^\phi(\theta \mid h_t) - \lambda_{\text{action}} \log \hat{p}^\phi(a_t^{\mathbf{H}} \mid h_t) \Big], \quad (1) \end{aligned}$$

where $\lambda_{\text{policy}}$, $\lambda_{\text{value}}$, $\lambda_{\text{reward}}$, and $\lambda_{\text{action}}$ are weights that trade off the four loss terms, and $\pi_t^{\text{MCTS}}$ refers to the action distribution output by MCTS at timestep $t$. The technique of learning an approximate belief distribution over the reward parameters $\theta$ from rollouts is similar to learned belief search (Hu et al., 2021a). After a few epochs of gradient descent on $L(\phi)$ over the collected episodes, AssistanceZero collects new episodes by running MCTS with the updated network parameters and repeats the process. See Appendix A for a full description of AssistanceZero and our variant of MCTS.

## 4. The Minecraft Building Assistance Game

To investigate whether solving complex assistance games is possible with AssistanceZero, we introduce the Minecraft Building Assistance Game (MBAG). When designing MBAG, we aimed to satisfy a few desiderata to make it a useful environment for studying assistance games more broadly. First, we want the distribution over reward parameters $p(\theta)$ to be complex but structured, similarly to human preferences in other domains. As described in the related work, most past work on assistance games has considered only a small number of possible reward functions. Second, we want there to be a variety of ways for the assistant to help the human that require varying amounts of information about the reward function. Finally, we want an environment in which it is tractable for academic labs to to train RL agents, making it feasible to empirically study more complex assistance games. In the remainder of this section, we describe the structure and implementation of MBAG.

A state in MBAG consists of a 3-dimensional grid of blocks, player locations within the grid, and player inventories. Each location in the grid can be one of ten block types, including air; we use an $11 \times 10 \times 10$ grid for our experiments. Each agent, or player, can be at any discrete location within the 3-dimensional grid as long as that grid cell and the one above it are air. The action space consists of a no-op, moving in one of the six cardinal directions, placing a block, breaking a block, or giving a block to another player. Place, break, and give actions are parameterized by a location, and place and give actions are additionally parameterized by a block type. This means that in the $11 \times 10 \times 10$ environment there are over 20,000 possible actions, although in most states only a small subset of those can be taken.

The reward parameters $\theta$ consist of a goal grid of blocks. At the start of an episode, the goal is sampled from a dataset of houses based on the CraftAssist dataset (Gray et al., 2019). We maintain separate train and test datasets to evaluate generalization. While the human agent can observe the goal, it is not visible to the assistant. MBAG satisfies our first desideratum because there is an exponentially large number of possible goals (on the order of $10^{400}$), making the goal distribution much more complex than prior studies of assistance games. However, due to the structured nature of the houses, the assistant can still infer information about the goals from human interaction. MBAG also satisfies the second desideratum because some assistant strategies, like collecting resources or digging a foundation, require very little knowledge of the goal. On the other hand, adding final decorations requires specific information. For more details about the MBAG environment, see Appendix B.1.

## 5. Experiments

**Human models** Training and evaluating assistants in MBAG requires a human policy $\pi_{\mathbf{H}}(a^{\mathbf{H}} \mid s, \theta)$ that selects actions based on the current state $s$ and the goal structure $\theta$. We trained three human models for MBAG using PPO, AlphaZero, and behavior cloning (BC) using the same Transformer-based architecture (see Appendix B.3 for details). The reward function for PPO and AlphaZero is based on goal similarity: the agent receives a reward of 1 ($-1$) for correctly (incorrectly) placing and breaking blocks, and 0 otherwise. For BC, we collected 18 episodes of human data from 5 subjects; in half the episodes the subject played alone and in the other half they played with a human assistant. Subjects were able to see a "blueprint" overlay showing the goal structure, while the human assistant was not. The BC human model is trained to imitate human actions from the dataset of subjects playing alone, while the PPO and AlphaZero models are trained with goal structures sampled from the train house dataset. Besides initializing BC from random weights, we also fine-tuned the PPO and AlphaZero policy networks with BC; Yang et al. (2022) find that initializing imitation learning with a near-optimal policy can improve human modeling.

We evaluate each model's accuracy at predicting human actions and performance at building goal structures in the test dataset. We evaluate the human models on the first objective by measuring the cross-entropy (CE) between the model's predicted actions and human actions in the dataset; we use 5-fold cross-validation for the BC policies. For the second objective, we report the percentage of the goal structure completed after 5, 10, and 20 minutes of acting in the environment, where one timestep is 0.8 seconds.

Table 5 shows the results of the evaluating the five human models. As expected, the BC models achieve the lowest CE since they are trained solely to imitate human actions; in contrast, the RL-based models are poor predictors of human behavior. Initializing BC from the PPO policy network results in slightly lower CE compared to initializing from AlphaZero. We also compare each model's goal percentages with those of the human subjects during data collection. The PPO and AlphaZero models are significantly better at building the goal structure than real humans. BC with random initialization performs worse than the human subjects, while BC models initialized from PPO and AlphaZero perform better. Overall, we found the BC model initialized from PPO to be the most human-like when considering both the CE and goal completion metrics.For this reason, we use this as the human policy $\pi_{\mathbf{H}}$ for the remainder of the experiments.

| Model | Cross-entropy | Goal percentage 5 min. | 10 min. | 20 min. |
|---|---|---|---|---|
| AlphaZero | 5.70 | 85.56 | 95.24 | 95.85 |
| PPO | 9.40 | 85.34 | 92.11 | 94.45 |
| BC (random init) | 2.32 | 22.96 | 41.51 | 58.33 |
| BC (AlphaZero init) | 2.41 | 49.86 | 79.39 | 91.69 |
| BC (PPO init) | 2.38 | 41.98 | 68.50 | 83.33 |
| Human subjects | – | 27.54 | 55.71 | 87.87 |

*Table 1.* We evaluate five policies as human models based on their accuracy at predicting human actions (cross-entropy) and performance at building goal structures (goal percentage).

**AI assistant policies** We now turn to developing effective AI assistant policies. In particular, we aim to find an assistant policy that performs well in the assistance POMDP defined using our fixed BC human model $\pi_{\mathbf{H}}$. We explore two methods of explicitly solving the assistance POMDP. First, we train a policy using AssistanceZero, as described in Section 3. To compare to a model-free baseline, we also train an assistant with PPO that uses the same policy network architecture (see Appendix B.3 for details). Our PPO baseline incorporates two auxiliary losses, without which we found training an even marginally effective PPO assistant was impossible; see Appendix B.4 for more information.

Besides assistants which explicitly solve the assistance POMDP, we also compare to baselines based on imitation learning. Assistants like GitHub Copilot (Chen et al., 2021) work by predicting human actions based on a large dataset of human behavior (e.g., all open source repositories on GitHub) and then taking those actions more quickly than a human can. To train an equivalent assistant in MBAG, we create a *non-goal-conditioned* (NGC) human model $\tilde{\pi}_{\mathbf{H}}$ based on $\pi_{\mathbf{H}}$, which we call, that marginalizes over the hidden goal $\theta$: $\tilde{\pi}_{\mathbf{H}}(a_t \mid h_t) = \int_\Theta p(\theta \mid h_t)\pi_{\mathbf{H}}(a_t \mid s_t, \theta)d\theta$. In practice, we approximate this integral by sampling 10,000 goal structures from the CraftAssist dataset, generating rollouts using $\pi_{\mathbf{H}}$, and training $\tilde{\pi}_{\mathbf{H}}$ with BC to imitate these rollouts. Similarly to how Copilot only makes a suggestion when it is relatively sure about the right action to take, we also explore thresholding $\tilde{\pi}_{\mathbf{H}}$'s actions based on their probability. In particular, if an action $a$ sampled from $\tilde{\pi}_{\mathbf{H}}$ has $\tilde{\pi}_{\mathbf{H}}(a \mid h) < c$, then it is replaced with a no-op, where $c$ is a tunable confidence threshold. Our third imitation learning-based assistant is trained by fine-tuning $\tilde{\pi}_{\mathbf{H}}$ on actions taken by the real human assistant during data collection. This assistant is analogous that produced in the supervised fine-tuning (SFT) phase of RLHF, so we call it the SFT assistant.

Table 5 shows the goal percentage achieved after 5, 10, and 20 minutes by each assistant paired with $\pi_{\mathbf{H}}$, evaluated over 100 episodes with goal structures from the test set. For reference, we show the performance of $\pi_{\mathbf{H}}$ alone and of real human subjects both with and without a human assistant. The confidence-thresholded non-goal-conditioned BC, SFT, and PPO assistant policies all appear to slightly outperform $\pi_{\mathbf{H}}$ alone at 5 and 10 minutes, although the results are not statistically significant. On the other hand, AssistanceZero significantly boosts performance, achieving 17 and 11 more goal percentage points at 5 and 10 minutes, respectively. This is greater than the performance increase in our human study between humans playing alone versus with a human assistant. Our results show that AssistanceZero is effective at solving complex assistance games. See this anonymized video link of AssistanceZero playing with a real human.

| Assistant | 5 min. | 10 min. | 20 min. |
|---|---|---|---|
| None | $42.0 \pm 0.9$ | $68.5 \pm 1.0$ | $83.3 \pm 0.9$ |
| NGC BC | $32.5 \pm 4.1$ | $52.3 \pm 3.9$ | $65.0 \pm 4.5$ |
| (w/ conf. threshold) | $45.4 \pm 3.2$ | $72.8 \pm 3.0$ | $84.3 \pm 2.6$ |
| SFT | $45.3 \pm 3.0$ | $70.2 \pm 3.0$ | $81.5 \pm 2.8$ |
| PPO | $44.5 \pm 3.3$ | $71.7 \pm 3.2$ | $85.7 \pm 2.7$ |
| AssistanceZero (ours) | $\mathbf{59.1 \pm 2.8}$ | $\mathbf{79.6 \pm 2.9}$ | $\mathbf{87.5 \pm 2.7}$ |
| Human subjects (alone) | $27.5 \pm 5.6$ | $55.7 \pm 12$ | $87.9 \pm 12$ |
| (w/ human assistant) | $34.4 \pm 10$ | $63.1 \pm 17$ | $88.5 \pm 10$ |

*Table 2.* The goal percentage achieved by AI assistant policies paired with the human model $\pi_{\mathbf{H}}$ after 5, 10, and 20 minutes (each timestep is 0.8 seconds) with 90% confidence intervals.

## References


Bai, Y., Jones, A., Ndousse, K., Askell, A., Chen, A., DasSarma, N., Drain, D., Fort, S., Ganguli, D., Henighan, T., Joseph, N., Kadavath, S., Kernion, J., Conerly, T., El-Showk, S., Elhage, N., Hatfield-Dodds, Z., Hernandez, D., Hume, T., Johnston, S., Kravec, S., Lovitt, L., Nanda, N., Olsson, C., Amodei, D., Brown, T., Clark, J., McCandlish, S., Olah, C., Mann, B., and Kaplan, J. Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback, April 2022. URL http://arxiv.org/abs/2204.05862. arXiv:2204.05862 [cs].

Baker, B., Akkaya, I., Zhokhov, P., Huizinga, J., Tang, J., Ecoffet, A., Houghton, B., Sampedro, R., and Clune, J. Video PreTraining (VPT): Learning to Act by Watching Unlabeled Online Videos, June 2022. URL http://arxiv.org/abs/2206.11795. arXiv:2206.11795 [cs].

Bakhtin, A., Wu, D. J., Lerer, A., Gray, J., Jacob, A. P., Farina, G., Miller, A. H., and Brown, N. Mastering the Game of No-Press Diplomacy via Human-Regularized Reinforcement Learning and Planning, October 2022. URL http://arxiv.org/abs/2210.05492. arXiv:2210.05492 [cs].

Bara, C.-P., CH-Wang, S., and Chai, J. MindCraft: Theory of Mind Modeling for Situated Dialogue in Collaborative Tasks. *arXiv:2109.06275 [cs]*, September 2021. URL http://arxiv.org/abs/2109.06275. arXiv: 2109.06275.

Brown, N., Bakhtin, A., Lerer, A., and Gong, Q. Combining Deep Reinforcement Learning and Search for Imperfect-Information Games, November 2020. URL http://arxiv.org/abs/2007.13544. arXiv:2007.13544 [cs].

Carroll, M., Shah, R., Ho, M. K., Griffiths, T. L., Seshia, S. A., Abbeel, P., and Dragan, A. On the Utility of Learning about Humans for Human-AI Coordination. *arXiv:1910.05789 [cs, stat]*, January 2020. URL http://arxiv.org/abs/1910.05789. arXiv: 1910.05789.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating Large Language Models Trained on Code, July 2021. URL http://arxiv.org/abs/2107.03374. arXiv:2107.03374 [cs].

Dragan, A. D. and Srinivasa, S. S. A policy-blending formalism for shared control. *The International Journal of Robotics Research*, 32(7):790–805, June 2013. ISSN 0278-3649. doi: 10.1177/0278364913490324. URL https://doi.org/10.1177/0278364913490324. Publisher: SAGE Publications Ltd STM.

Fan, L., Wang, G., Jiang, Y., Mandlekar, A., Yang, Y., Zhu, H., Tang, A., Huang, D.-A., Zhu, Y., and Anandkumar, A. MineDojo: Building Open-Ended Embodied Agents with Internet-Scale Knowledge, November 2022. URL http://arxiv.org/abs/2206.08853. arXiv:2206.08853 [cs].

Fern, A., Natarajan, S., Judah, K., and Tadepalli, P. A Decision-Theoretic Model of Assistance. *Journal of Artificial Intelligence Research*, 50:71–104, May 2014. ISSN 1076-9757. doi: 10.1613/jair.4213. URL https://www.jair.org/index.php/jair/article/view/10880.

Fisac, J. F., Gates, M. A., Hamrick, J. B., Liu, C., Hadfield-Menell, D., Palaniappan, M., Malik, D., Sastry, S. S., Griffiths, T. L., and Dragan, A. D. Pragmatic-Pedagogic Value Alignment. In Amato, N. M., Hager, G., Thomas, S., and Torres-Torriti, M. (eds.), *Robotics Research*, Springer Proceedings in Advanced Robotics, pp. 49–57, Cham, 2020. Springer International Publishing. ISBN 978-3-030-28619-4. doi: 10.1007/978-3-030-28619-4_7.

Gleave, A. and Irving, G. Uncertainty Estimation for Language Reward Models, March 2022. URL http://arxiv.org/abs/2203.07472. arXiv:2203.07472 [cs].

Gray, J., Srinet, K., Jernite, Y., Yu, H., Chen, Z., Guo, D., Goyal, S., Zitnick, C. L., and Szlam, A. CraftAssist: A Framework for Dialogue-enabled Interactive Agents. *arXiv:1907.08584 [cs]*, July 2019. URL http://arxiv.org/abs/1907.08584. arXiv: 1907.08584.

Hadfield-Menell, D., Russell, S. J., Abbeel, P., and Dragan, A. Cooperative Inverse Reinforcement Learning. In *Advances in Neural Information Processing Systems 29*, pp. 3909–3917. Curran Associates, Inc., 2016.

Hu, H., Lerer, A., Peysakhovich, A., and Foerster, J. "Other-Play" for Zero-Shot Coordination. In *International Con-*

*ference on Machine Learning*, pp. 4399–4410. PMLR, 2020.

Hu, H., Lerer, A., Brown, N., and Foerster, J. Learned Belief Search: Efficiently Improving Policies in Partially Observable Settings, June 2021a. URL http://arxiv.org/abs/2106.09086. arXiv:2106.09086 [cs].

Hu, H., Lerer, A., Cui, B., Pineda, L., Brown, N., and Foerster, J. Off-Belief Learning. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 4369–4379. PMLR, July 2021b. URL https://proceedings.mlr.press/v139/hu21c.html. ISSN: 2640-3498.

Jacob, A. P., Wu, D. J., Farina, G., Lerer, A., Hu, H., Bakhtin, A., Andreas, J., and Brown, N. Modeling Strong and Human-Like Gameplay with KL-Regularized Search, February 2022. URL http://arxiv.org/abs/2112.07544. arXiv:2112.07544 [cs].

Javdani, S., Srinivasa, S. S., and Bagnell, J. A. Shared Autonomy via Hindsight Optimization, April 2015. URL http://arxiv.org/abs/1503.07619. arXiv:1503.07619 [cs].

Johnson, M., Hofmann, K., Hutton, T., and Bignell, D. The Malmo platform for artificial intelligence experimentation. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pp. 4246–4247, New York, New York, USA, July 2016. AAAI Press. ISBN 978-1-57735-770-4.

Kanervisto, A., Milani, S., Ramanauskas, K., Topin, N., Lin, Z., Li, J., Shi, J., Ye, D., Fu, Q., Yang, W., Hong, W., Huang, Z., Chen, H., Zeng, G., Lin, Y., Micheli, V., Alonso, E., Fleuret, F., Nikulin, A., Belousov, Y., Svidchenko, O., and Shpilman, A. MineRL Diamond 2021 Competition: Overview, Results, and Lessons Learned, February 2022. URL http://arxiv.org/abs/2202.10583. arXiv:2202.10583 [cs].

Kiseleva, J., Li, Z., Aliannejadi, M., Mohanty, S., ter Hoeve, M., Burtsev, M., Skrynnik, A., Zholus, A., Panov, A., and Srinet, K. Interactive grounded language understanding in a collaborative environment: Iglu 2021. In *NeurIPS 2021 Competitions and Demonstrations Track*, pp. 146–161. PMLR, 2022. URL https://proceedings.mlr.press/v176/kiseleva22a.html.

Kocsis, L. and Szepesvári, C. Bandit Based Monte-Carlo Planning. In Fürnkranz, J., Scheffer, T., and Spiliopoulou, M. (eds.), *Machine Learning: ECML 2006*, Lecture Notes in Computer Science, pp. 282–293, Berlin, Heidelberg, 2006. Springer. ISBN 978-3-540-46056-5. doi: 10.1007/11871842_29.

Laidlaw, C. and Dragan, A. The Boltzmann Policy Distribution: Accounting for Systematic Suboptimality in Human Models. October 2021. URL https://openreview.net/forum?id=_l_QjPGN5ye.

Lang, L., Foote, D., Russell, S., Dragan, A., Jenner, E., and Emmons, S. When Your AIs Deceive You: Challenges with Partial Observability of Human Evaluators in Reward Learning, March 2024. URL http://arxiv.org/abs/2402.17747. arXiv:2402.17747 [cs, stat].

Madani, O., Hanks, S., and Condon, A. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence*, 147(1):5–34, July 2003. ISSN 0004-3702. doi: 10.1016/S0004-3702(02)00378-8. URL https://www.sciencedirect.com/science/article/pii/S0004370202003788.

Malik, D., Palaniappan, M., Fisac, J. F., Hadfield-Menell, D., Russell, S., and Dragan, A. D. An Efficient, Generalized Bellman Update For Cooperative Inverse Reinforcement Learning. *arXiv:1806.03820 [cs]*, June 2018. URL http://arxiv.org/abs/1806.03820. arXiv: 1806.03820.

Mehta, N., Teruel, M., Sanz, P. F., Deng, X., Awadallah, A. H., and Kiseleva, J. Improving Grounded Language Understanding in a Collaborative Environment by Interacting with Agents Through Help Feedback, February 2024. URL http://arxiv.org/abs/2304.10750. arXiv:2304.10750 [cs].

Milani, S., Kanervisto, A., Ramanauskas, K., Schulhoff, S., Houghton, B., Mohanty, S., Galbraith, B., Chen, K., Song, Y., Zhou, T., Yu, B., Liu, H., Guan, K., Hu, Y., Lv, T., Malato, F., Leopold, F., Raut, A., Hautamäki, V., Melnik, A., Ishida, S., Henriques, J. F., Klassert, R., Laurito, W., Novoseller, E., Goecks, V. G., Waytowich, N., Watkins, D., Miller, J., and Shah, R. Towards Solving Fuzzy Tasks with Human Feedback: A Retrospective of the MineRL BASALT 2022 Competition, March 2023. URL http://arxiv.org/abs/2303.13512. arXiv:2303.13512 [cs].

Papadimitriou, C. H. and Tsitsiklis, J. N. The Complexity of Markov Decision Processes. *Mathematics of Operations Research*, 12(3):441–450, 1987. ISSN 0364-765X. URL https://www.jstor.org/stable/3689975. Publisher: INFORMS.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., and Silver, D. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *Nature*, 588(7839):604–609, December 2020. ISSN 0028-0836, 1476-4687. doi: 10.1038/

s41586-020-03051-4. URL http://arxiv.org/abs/1911.08265. arXiv:1911.08265 [cs, stat].

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal Policy Optimization Algorithms. *arXiv:1707.06347 [cs]*, August 2017. URL http://arxiv.org/abs/1707.06347. arXiv: 1707.06347.

Shah, R., Freire, P., Alex, N., Freedman, R., Krasheninnikov, D., Chan, L., Dennis, M. D., Abbeel, P., Dragan, A., and Russell, S. Benefits of Assistance over Reward Learning. October 2020. URL https://openreview.net/forum?id=DFIoGDZejIB.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, December 2017. URL http://arxiv.org/abs/1712.01815. arXiv:1712.01815 [cs].

Skrynnik, A., Volovikova, Z., Côté, M.-A., Voronov, A., Zholus, A., Arabzadeh, N., Mohanty, S., Teruel, M., Awadallah, A., Panov, A., Burtsev, M., and Kiseleva, J. Learning to Solve Voxel Building Embodied Tasks from Pixels and Natural Language Instructions, November 2022. URL http://arxiv.org/abs/2211.00688. arXiv:2211.00688 [cs].

Stone, P., Kaminka, G., Kraus, S., and Rosenschein, J. Ad Hoc Autonomous Agent Teams: Collaboration without Pre-Coordination. volume 3, January 2010.

Strouse, D., McKee, K., Botvinick, M., Hughes, E., and Everett, R. Collaborating with Humans without Human Data. In *Advances in Neural Information Processing Systems*, volume 34, pp. 14502–14515. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/hash/797134c3e42371bb4979a462eb2f042a-Abstract.html.

Szlam, A., Gray, J., Srinet, K., Jernite, Y., Joulin, A., Synnaeve, G., Kiela, D., Yu, H., Chen, Z., Goyal, S., Guo, D., Rothermel, D., Zitnick, C. L., and Weston, J. Why Build an Assistant in Minecraft? *arXiv:1907.09273 [cs]*, July 2019. URL http://arxiv.org/abs/1907.09273. arXiv: 1907.09273.

Treutlein, J., Dennis, M., Oesterheld, C., and Foerster, J. A New Formalism, Method and Open Issues for Zero-Shot Coordination. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 10413–10423. PMLR, July 2021. URL https://proceedings.mlr.press/v139/treutlein21a.html. ISSN: 2640-3498.

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. Voyager: An Open-Ended Embodied Agent with Large Language Models, October 2023. URL http://arxiv.org/abs/2305.16291. arXiv:2305.16291 [cs].

Woodward, M., Finn, C., and Hausman, K. Learning to Interactively Learn and Assist. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(03):2535–2543, April 2020. ISSN 2374-3468. doi: 10.1609/aaai.v34i03.5636. URL https://ojs.aaai.org/index.php/AAAI/article/view/5636. Number: 03.

Yang, M., Carroll, M., and Dragan, A. Optimal Behavior Prior: Data-Efficient Human Models for Improved Human-AI Collaboration, November 2022. URL http://arxiv.org/abs/2211.01602. arXiv:2211.01602 [cs].

Yu, C., Velu, A., Vinitsky, E., Gao, J., Wang, Y., Bayen, A., and Wu, Y. The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games, November 2022. URL http://arxiv.org/abs/2103.01955. arXiv:2103.01955 [cs].

Zhi-Xuan, T., Ying, L., Mansinghka, V., and Tenenbaum, J. B. Pragmatic Instruction Following and Goal Assistance via Cooperative Language-Guided Inverse Planning, February 2024. URL http://arxiv.org/abs/2402.17930. arXiv:2402.17930 [cs].

Zholus, A., Skrynnik, A., Mohanty, S., Volovikova, Z., Kiseleva, J., Szlam, A., Coté, M.-A., and Panov, A. I. IGLU Gridworld: Simple and Fast Environment for Embodied Dialog Agents, May 2022. URL http://arxiv.org/abs/2206.00142. arXiv:2206.00142 [cs].

# Appendix

## A. AssistanceZero details

In this appendix, we describe the full details of the AssistanceZero algorithm.

**MCTS** To choose actions during training and deployment, AssistanceZero uses Monte Carlo tree search. MCTS repeats a three-stage process for $N_{\text{sim}}$ simulations, adding one additional node during each simulation to a tree where nodes represent histories and branches are action pairs $(a^{\mathbf{H}}, a^{\mathbf{R}})$.

In the *selection* stage, an assistant action $a^{\mathbf{R}}$ is selected at the current history node $h$ that maximizes

$$Q(h, a^{\mathbf{R}}) + c_{\text{PUCT}}\, \pi^{\phi}(a^{\mathbf{R}} \mid h) \frac{\sqrt{\sum_{b \in \mathcal{A}^{\mathbf{R}}} N(h, b)}}{1 + N(h, a^{\mathbf{R}})}, \tag{2}$$

where $N(h, a^{\mathbf{R}})$ is the number of times action $a^{\mathbf{R}}$ has previously been selected at node $h$, $\pi^{\phi}(a^{\mathbf{R}} \mid h)$ is the output of the network's policy head, and $c_{\text{PUCT}}$ is a tunable parameter that balances exploration and exploitation. $Q(h, a^{\mathbf{R}})$ is an estimate of the Q-value of $a^{\mathbf{R}}$; we will describe how this is calculated later. Once an assistant action is chosen, then a human action $a^{\mathbf{H}}$ is sampled according to the probabilities output by the human action predictor head $\hat{p}^{\phi}(a^{\mathbf{H}} \mid h)$. Then, the state $s'$ resulting from taking actions $(a^{\mathbf{H}}, a^{\mathbf{R}})$ is calculated and the state and actions are appended to $h$ to reach a node $h'$. The reward associated with the transition is estimated by marginalizing over the reward parameter distribution output by the reward prediction head:

$$\hat{R}(h, a^{\mathbf{H}}, a^{\mathbf{R}}) = \sum_{\theta \in \Theta} R(s, a^{\mathbf{H}}, a^{\mathbf{R}}; \theta)\, \hat{p}^{\phi}(\theta \mid h').$$

Then, the selection process repeats until a node $h$ is reached which has not previously been reached.

In the *expansion stage*, the new node is input to the network to calculate the policy head outputs $\pi^{\phi}(a^{\mathbf{R}} \mid h)$, the value estimate $\hat{V}^{\phi}(h)$, the human action predictions $\hat{p}^{\phi}(a^{\mathbf{H}} \mid h)$, and the reward parameter predictions $\hat{p}^{\phi}(\theta \mid h)$. The policy outputs at the root node have Dirichlet noise applied, similarly to AlphaZero.

In the *backup stage*, the Q-values of all ancestor nodes of $h$ are recursively updated with the discounted sum of rewards along edges of the tree plus the value estimate $\hat{V}^{\phi}(h)$. As normally in MCTS, $Q(h, a^{\mathbf{R}})$ is simply the average of the Q-values estimated over all previous simulations that have taken $a^{\mathbf{R}}$ in node $h$. For actions with no visits, $Q(h, a^{\mathbf{R}})$ is set to the average of all backed-up values for node $h$:

$$Q(h, a^{\mathbf{R}}) = \frac{\sum_{b \in \mathcal{A}^{\mathbf{R}}} N(h, b) Q(h, b)}{\sum_{b \in \mathcal{A}^{\mathbf{R}}} N(h, b)} \qquad \text{if} \quad N(h, a^{\mathbf{R}}) = 0.$$

When selecting actions according to (2), we normalize Q-values by the highest and lowest value seen among all visits to that node, similarly to MuZero (Schrittwieser et al., 2020).

The resulting policy from MCTS is defined as

$$\pi^{\text{MCTS}}(a^{\mathbf{R}} \mid h) \propto N(h, a^{\mathbf{R}})^{\tau},$$

where $\tau$ is an inverse temperature parameter.

**Training procedure** As described in Section 3, AssistanceZero alternates between rolling out episodes in the environment by selecting actions with MCTS and updating the network according to the loss function in (1). In practice, we use a replay buffer to store rollouts; then, after storing a certain number of new rollouts, we randomly sample a number of episodes from the replay buffer and train on these. We found that using a replay buffer improves performance and stability.

## B. Experimental Details

### B.1. Environment

We make MBAG tractable to train and plan in by implementing it in a mix of pure Python and C, with no dependency on Minecraft for training. However, we also provide an interface with the Microsoft Malmo (Johnson et al., 2016) mod that allows the Python environment to sync with Minecraft. This can be used for video visualization of policies. It also enables human-AI play, in which human actions detected in Minecraft are translated into their equivalents in MBAG, and AI actions taken in MBAG are translated into actions in Minecraft.

We provide two versions of MBAG: one where the players must collect resources by breaking a regenerating "palette" of blocks located on one side of the environment, and one where the players have unlimited blocks. For the purposes of this paper, we investigate the second version; this version of the environment is more difficult to build an assistant for, since the assistant cannot simply collect resources to help the human.

#### B.1.1. Reward Function

The human policy and the AI assistant policy receive the same shared reward at each time step primarily based on goal distance, which is the fewest number of place and break actions needed to reach the goal from the current state. The joint reward is equal to the goal distance before the actions were taken minus the goal distance after. That is, letting $d(s, \theta)$ be the goal distance,

$$R(s, a^{\mathbf{H}}, a^{\mathbf{R}}; \theta) = d(s, \theta) - d(s', \theta),$$

where $s'$ is the state reached by taking actions $(a^{\mathbf{H}}, a^{\mathbf{R}})$ in state $s$. This definition of reward means that the maximum reward achievable starting in a state is always $d(s, \theta)$.

#### B.1.2. Goal Structures

We base the goal structures for MBAG on the CraftAssist houses dataset, which was collected by Gray et al. (2019); they gave study participants the open-ended task of building any house in Minecraft and recorded the resulting structure. Since we require that goal structures in MBAG have a one-block gap on all sides, they can only be at most of dimensions $9 \times 8 \times 8$. However, many of the goal structures in the CraftAssist dataset are much larger. When houses in the dataset are no more than twice the desired dimensions, we scale them down to fit.

### B.2. Data Collection

To train the human models, we collect 18 episodes of 5 human subjects building goal structures. For half of the total episodes, the subject is given a goal structure and is instructed to build it quickly and efficiently without assistance. For the other half, a single experienced human Minecraft player acts as the assistant to help build the house. The human assistant is instructed to help the human subjects build their goal structures, but they are not shown the goal structure themselves. While the human agent and assistant can observe each other's actions, there is otherwise no communication between them.

### B.3. Network Architecture

For both the human models and AI assistant policies, we use a Transformer architecture with 6 spatial layers, 64 hidden units, and 4 heads. Each of the 1,100 blocks in the environment is a separate "token" and they are identified by 12-dimensional positional embeddings. Due to the large world size, training would be computationally prohibitive if each spatial layer attended across all 1,100 blocks. Thus, instead, we restrict attention in each layer to blocks in a slice along only a single dimension. Layers 1 and 4 only allow attention along the X direction, layers 2 and 5 along the Y direction, and layers 3 and 6 along Z. The input to the Transformer at each block location is the concatenation of:

- an embedding representing the current block type present at that location,
- an embedding representing the goal block type at that location (if the goal is visible to the agent),
- an embedding representing which player, if any, is standing at that location,
- an embedding representing which player, if any, was the last to place or break a block at that location (this allows the agents' actions to be visible to each other),
- the counts of each type of block in the player's inventory divided by 64,
- and the current timestep divided by 1,000.

For recurrent policies, we add two additional layers after the 3rd and 6th transformer layers. Each of these layers consists of LSTM cells at each block location that share weights; these enable memory across time.

### B.4. Training Details

We develop different human and AI assistant policies using model-based RL, model-free RL, behavior cloning, and combinations of these methods. Each policy uses similar model architectures (described in B.3) and is trained for an episode length of $1,500$.

During training, we randomize the starting location of the human policy to improve generalization. Since some RL algorithms sample experience in fragments shorter than a full episode, we randomize the length of the first episode in the environment. This avoids the situation where in one iteration of PPO all fragments are from the beginning of episodes and in the next they are all from the end.

**Data augmentation** We apply data augmentation during behavior cloning for only the goal-conditioned human models. The data augmentation consists of choosing a random permutation of block types for each state and applying it to the current blocks in the world, the block types in the goal structure, the players' inventories, and any place or give actions. We found that using data augmentation led to improved generalization in cross validation.

**PPO human model (single-agent)** The hyperparameters we used to train the PPO human model are shown in Table 3.

**AlphaZero human model (single-agent)** The hyperparameters we used to train the AlphaZero human model are shown in Table 4.

We observed that the AlphaZero human policy could not successfully construct the goal structure when trained directly with the full $1500$ episode length. We hypothesize this is because, early in training, the policy gets stuck after the beginning of the episode and thus does not collect useful experience for the remainder. As the episode length increases, the useless experience where the policy is stuck becomes a greater proportion of the training data and leads to decreased performance. To address this issue, we terminate the episode if the policy does achieve a new minimum goal distance for $100$ time steps. This allows us to train with the full episode length while skipping less useful experience.

We found it helpful to add a penalty of $-0.2$ for no-ops to the reward function to the encourage the policy to act and explore.

**Behavior cloning human model (single-agent)** We train three main BC human model variants: 1) initialized from scratch, 2) initialized from a checkpoint of the PPO human model, and 3) initialized from a checkpoint of the AlphaZero human model. We use data from human subjects building goal structures on their own, as described in B.2.

Hyperparameters are shown in Table 5.

**PPO assistant** To effectively train an assistance PPO, we added two auxiliary loss terms and modified the reward function. The first loss term, which we call the "block-placing loss," is the cross-entropy between the block type placed by the assistant and the goal block type at that location, if there is one. This loss provides some training signal when the assistant places a block in a location that is part of the goal structure, even if the block type is incorrect. Without this loss, placing an incorrect block type would simply result in a reward of $0$, making it more challenging for the assistant to learn to place blocks at all. We linearly decay this loss coefficient from $1$ to $0$ over the first $2 \times 10^6$ time steps.

The second loss adds a goal prediction head similar to that used in AssistanceZero and trained with the same loss function.

Finally, we modify the reward function for PPO to only give reward directly attributable to the place/break actions of aand disregard any place/break actions taken by the human. This means that PPO's goal is not actually aligned with the assistance game objective; however, without this modification we found that the PPO assistant just learned to take no-op actions constantly.

All the hyperparameters for the PPO assistant are shown in Table 3.

**AssistanceZero assistant** For the first 25 iterations of AssistanceZero, we "pre-train" the assistant's value, human action prediction, and reward parameter prediction heads by having it only take no-op actions while observing the human policy. This provides good initialization of all three heads without requiring the expense of running MCTS during these initial iterations. After the pretraining iterations, we start using MCTS and training the policy head as well. We use the same interleaved transformer-LSTM model architecture for the assistant's network as for the PPO assistant.

| Hyperparameter | Human model | Assistant |
|---|---|---|
| Training iterations | 150 | 100 |
| Rollout length | 512 | 512 |
| Number of environments | 125 | 64 |
| SGD minibatch size | 512 | 512 |
| SGD epochs per iteration | 3 | 3 |
| Optimizer | Adam | Adam |
| Learning rate | $3 \times 10^{-4}$ | $3 \times 10^{-4}$ |
| Discount factor ($\gamma$) | 0.95 | 0.95 |
| GAE coefficient ($\lambda$) | 0.95 | 0.95 |
| Entropy coefficient (horizon) | 0.03 | $1 \to 0.01$ ($2 \times 10^6$) |
| Clipping parameter | 0.2 | 0.2 |
| Grad clip norm threshold | 10 | 10 |
| Recurrent network | No | Yes |
| KL target | 0.01 | 0.01 |
| KL coeff. | 0.2 | 0.2 |
| Value function coeff. | 0.01 | 0.01 |
| Goal loss coeff. | 0 | 3 |
| Place block loss coeff. (horizon) | 0 | $1 \to 0$ ($2 \times 10^6$) |

*Table 3.* PPO hyperparameters for the human model (single-agent) and assistant training.

Hyperparameters are shown in Table 4.

**Imitation learning assistants** We train two main imitation learning assistants: 1) a non-goal-conditioned BC assistant, and 2) a BC assistant fine-tuned on human assistant data. Hyperparameters are shown in Table 5.

The network architecture is the same as the recurrent network used for the PPO and AlphaZero assistants.

### B.5. Evaluation

We evaluate each human model's single-agent performance on 1,000 episodes with goal structures sampled from a held-out test set which are not seen during training. We then evaluate each assistant policy's performance on the same test set by pairing it with a human model and evaluating for 100 episodes with goal structures from the test set. The episode terminates when the goal structure is fully built or 1500 time steps have passed. When evaluating AlphaZero, we use 30 MCTS simulations for computational reasons and to match the maximum number of simulations that can be executed in real-time.

| Hyperparameter | Human model | Assistant |
|---|---|---|
| Training iterations | 70 | 55-70 |
| Rollout length per iteration | 512 | 512 |
| Number of environments | 64 | 64 |
| Timesteps sampled from replay buffer per iteration | 261,632 | 131,072 |
| SGD minibatch size | 512 | 512 |
| SGD epochs per iteration | 1 | 2 |
| Optimizer | Adam | Adam |
| Learning rate | $3 \times 10^{-3}$ | $3 \times 10^{-3}$ |
| Discount factor ($\gamma$) | 0.95 | 0.95 |
| Grad clip norm threshold | 10 | 10 |
| Recurrent network | No | Yes |
| Value function coeff. | 0.01 | 0.01 |
| Goal loss coeff. | 0.5 | 3 |
| Other agent action prediction loss coeff. | N/A | 1 |
| No-op reward | -0.2 | -0.2 |
| Number of MCTS simulations | 100 | 100 |
| Inverse temperature | 1.5 | 1.5 |
| Dirichlet noise (high-level action) | 0.25 | 0.25 |
| Dirichlet noise (low-level action) | 10 | 10 |
| Dirichlet epsilon | 0.25 | 0.25 |
| Prior temperature | 1 | 1 |
| PUCT coefficient | 1 | 1 |
| Replay buffer capacity | 5,232,640 | 131,072 |
| Terminate episode if no progress (steps) | 100 | N/A |

*Table 4.* AlphaZero hyperparameters for the human model (single-agent) and assistant training.

| Hyperparameter | Human model | Non-goal-conditioned assistant | Fine-tuned assistant |
|---|---|---|---|
| Training iterations | 20 | 20 | 20 |
| Training batch size | 9642 | 8192 | 9642 |
| SGD minibatch size | 128 | 512 | 512 |
| SGD epochs per iteration | 1 | 1 | 1 |
| Optimizer | Adam | Adam | Adam |
| Learning rate | $1 \times 10^{-3} \to 1 \times 10^{-4}$ (10 iters) | $1 \times 10^{-3}$ | $1 \times 10^{-3}$ |
| Grad clip norm threshold | 10 | 10 | 10 |
| Interleave spatial/temporal layers | No | Yes | Yes |

*Table 5.* BC hyperparameters for the human model (single-agent), non-goal-conditioned assistant, and fine-tuned assistant.