A APPENDIX

Here we provide additional implementation details and results.

A.1 Dynamic Replanning Algorithm

Our dynamic replanning algorithm (Alg. 1) unfolds as follows: for a given agent i, we first initialize the character embeddings for teammates from past trajectories, if available. We also initialize empty queues for the observation history, h, and the planned sequence of observations, τ_{plan}.

At each step, after the agent makes a new observation in the environment, we check if replanning is needed (line 4). If a plan has been previously generated and is not depleted, we compute the mean squared error between the current observation and the observation predicted in the plan, *i.e.*, we set $\delta_{\text{obs}} \coloneqq \|\eta(o_i) - \eta(\hat{o_i})\|^2$, where $\eta : \Omega \to [0, 1]$ is an observation normalizer detailed in Sec. A.5.2.

If the observation difference is greater than $\lambda = 0.2$, we generate a new plan (lines 5–11). First, the mental embedding is computed from the current trajectory, and all conditioning variables—any combination of returns ($R_i(\tau_i)$), observer profile (ψ_i), character embedding (e_{char}), mental embedding (e_{ment})—are concatenated into $y(\tau_i)$ (line 5). We then generate a new plan through conditional sampling, starting with $\hat{\boldsymbol{\tau}}^K \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. At each diffusion step k, the portion of the observer agent in $\hat{\boldsymbol{\tau}}^k$ is "in-painted" with the corresponding current observation and C steps of observation history (line 8). We iteratively refine (denoise) trajectories $\hat{\boldsymbol{\tau}}^k$ using the perturbed noise $\hat{\epsilon}$ following [1, 31], where $\epsilon_\delta(\hat{\boldsymbol{\tau}}^k, k)$ is the unconditional noise, ω is a scalar for extracting the distinct portions of the trajectory that exhibit $y(\tau_i)$, and `denoise` is the noise scheduler predicting the previous sample, $\hat{\boldsymbol{\tau}}^{k-1}$, from $\hat{\boldsymbol{\tau}}^k$ by propagating the diffusion process from $\hat{\epsilon}$.

Once the final, denoised trajectory, $\hat{\boldsymbol{\tau}}^0$, is generated, we discard the first $C + 1$ steps—since they were only used to condition the generation process—and extract agent i's portion, $\hat{\tau}_i$, re-initializing the plan queue, τ_{plan} (line 11). The agent then retrieves an action by using the inverse dynamics model I_ξ on consecutive observations extracted from the plan (line 12).[7]

A.2 Agent Profiles

The profiles used in our experiments are listed in Table A.1, where for each profile (column) we list the weight θ_k (row value) associated with each feature ϕ_k. *Drop*, *Pickup*, and *Delivery* correspond to binary features indicating whether the agent's action at the previous timestep triggered the corresponding event, while *Potting Onion* assesses whether an onion was put in a pot by the agent (again corresponding to agent's previous action). X and Y are numeric features corresponding to agent locations and relative distances to the teammate in the gridworld, and *Path Dist* is the Manhattan distance to the teammate. *Task Rew Frac* is the fraction of the task reward received by the agent, corresponding to a reward of $R_{\text{task}} = 20$ if *any* agent successfully delivers a soup at timestep t, and 0 otherwise.

[7] Here the dequeue function extracts observations from the queue in a FIFO manner.

A.3 Multiagent RL Training

For training, we adopted the PPO algorithm [23] implementation in Ray's RLLib toolkit.[8] Each pair was trained for 1 000 iterations with 64 parallel workers. We used a learning rate of $8{\times}10^{-4}$ and a linearly decreasing entropy loss coefficient. At each iteration, a batch of data was collected from workers, each spwaning the overcooked environment with the layout shown in Fig. 2, randomizing the initial locations of agents. Each episode ran for a period of 400 timesteps. We collected 25 600 timesteps per batch, with a mini batch size of 6 400 for gradient updates, training for 420 iterations of 8 epochs each.

A.4 ToMnet

A.4.1 Dataset/Feature Preparation. We started with the pairwise RL agent dataset, $\mathcal{D}$, generated as described above. We sampled 2 000 datapoints for each agent/profile to train the ToMnet, where in each trial we selected a teammate uniformly at random. Each datapoint consisted of the following: we randomly sampled $N_{\text{past}} = 4$ trajectories of the agent pair of $T_{\text{past}} = 100$ steps from $\mathcal{D}$ to serve as "past" trajectories for the Character net; we sampled an additional trajectory and split it at a timestep, $t \sim \mathcal{U}(1, T - 30)$, selected uniformly at random. The observation of the observer agent at t was used as the "current" observation to be fed to the Prediction net, while the past $T_{\text{cur}} = 10$ timesteps acted as the "current" trajectory for the Mental net, where the data was reversed and zero-padded whenever $t < T_{\text{cur}}$. The teammate's future data in the current trajectory (a minimum of 30 timesteps) was used for calculating various ground-truth targets against which to train the Prediction net. Namely, we used the following prediction targets: (a) the teammate's next step action, modeled as a discrete distribution, $\Delta(A)$; (b) the sign of each of its profile reward weights, θ_i^k, modeled as a distribution over $\{-1, 0, 1\}$; and (c) the successor representation (SR), corresponding to statistics of the teammate's future behavior. In particular, SR consisted of three types of data: (i) *binary* features—whether the closest pot is empty, cooking or ready and similarly for the next closest pot (total 6 features), encoded as Bernoulli distributions; (ii) *categorical* features—relative frequency of the teammate carrying an onion, soup, dish, tomato or nothing (total 1 feature), encoded as a discrete distribution; (iii) *numeric* features—mean path distance to the teammate and x and y locations in the environment (total 3 features), encoded as a multivariate Gaussian distribution. To obtain the SR means, we used discounted averaging over the teammate future data with a discount factor of 0.99.

A.4.2 Training and Losses. The ToMnet architecture consisted of three networks as detailed in Sec. 3.2—a Character Net, a Mental Net and a Prediction Net. The Character and Mental Nets consisted of an LSTM layer of hidden size 64 (dropout of 0.2) that produces an initial embedding (final hidden state of the LSTM). The Character Net takes N_{past} trajectory data each of length $T_{\text{past}} = 100$ as input, and to calculate the Character embedding, e_{char}, we first concatenated the LSTM's embedding with the observer's profile weights, sending the result through a dense/linear layer to obtain an embedding of size $|e_{\text{char}}| = 8$. The Mental Net takes the current trajectory of length $T_{\text{cur}} = 10$ as input, where we masked out incomplete

[8] https://docs.ray.io/en/latest/rllib/index.html

Table A.1: The different reward profiles used in our experiments.

Feature, ϕ_k	Cook	Server	Helper	Far Helper	Follower	Sparse	Random
Onion Drop	−5.00	1.00	0.00	0.00	0.00	0.00	0.00
Onion Pickup	1.00	−5.00	0.10	0.10	0.00	0.00	0.00
Dish Drop	1.00	−5.00	0.00	0.00	0.00	0.00	0.00
Dish Pickup	−5.00	1.00	0.10	0.10	0.00	0.00	0.00
Potting Onion	5.00	−1.00	10.00	10.00	0.00	0.00	0.00
Soup Delivery	0.00	5.00	0.00	0.00	0.00	0.00	0.00
Soup Drop	1.00	−5.00	−20.00	−20.00	0.00	0.00	0.00
Soup Pickup	−5.00	10.00	15.00	15.00	0.00	0.00	0.00
Self Pos X	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Self Pos Y	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Dist To Other Player X	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Dist To Other Player Y	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Path Dist To Other Player	0.00	0.00	0.00	0.01	−1.00	0.00	0.00
Task Rew Frac	1.00	1.00	1.00	1.00	0.00	1.00	0.00

timesteps of a trajectory. To calculate the Mental embedding, e_{ment}, we concatenated the LSTM's embedding with the profile's weights and e_{char} before passing the result through a dense layer, resulting in an embedding of size $|e_{\text{ment}}| = 8$.

Finally, to obtain the different predictions, we first concatenated both embeddings with the observer's profile weights and the current observation to obtain the prediction input. This is then sent through the different prediction heads to produce probabilistic predictions for the different targets, *i.e.*, , next action, profile, SR features detailed earlier. Each prediction head consists of a dense layer of size 64 (dropout of 0.2) with output dimension equaling the number of possible outputs for that prediction, namely, $|A| = 6$ for the next action, $|\theta_j| \times 3 = 39$ for the profile sign prediction, 6 for the SR binary features, 5 for the SR categorical features, and 3 for the SR numeric features. We used the negative log-likelihood (NLL) loss for the next action and profile predictions, binary cross-entropy loss for SR binary predictions, cross-entropy loss for SR categorical prediction, and a Gaussian NLL loss for SR numeric features' predictions.

For training the ToMnet, we used the 2 000 datapoints generated as explained above for each agent type, resulting in 1 400 datapoints, using a train-validation split of 80%-20% respectively. We used an Adam optimizer with weight decay of 5×10^{-4}. We trained the model for 2 000 epochs via mini-batch gradient descent with batch size of 128 and learning rate of 5×10^{-4}, using an early stopping criterion based on a validation loss (maximum number of steps without improvement set to 10 and a maximum tolerance of 0.01).

A.5 MADiff

A.5.1 Trajectory Augmentation. To train the MADiff component, we sampled joint trajectories similarly to how we trained the ToMnet, but augmented trajectories by computing the ToM embeddings for each timestep using the previously-trained ToMnet as explained in Sec. 4.2. We used the same data parameters used to train the ToMnet, *i.e.*, , $N_{\text{past}} = 4$ past trajectories of $T_{\text{past}} = 100$ steps each and $T_{\text{cur}} = 10$. We then generated trajectories to train the MADiff module by consecutively sampling from the augmented trajectories in a sliding window manner, using $C = 16$ steps to constrain trajectory generation using in-painting, and the subsequent $H = 64$ steps as the planning horizon. Trajectories were zero-padded at the beginning ($t < H$)) and end ($t > T - H$) of each augmented trajectory during sampling.

A.5.2 Training. For each sampled trajectory (length $H+C$) we augmented the observer agent's observations by including the one-hot encoding of the teammate's actions and then performed Cumulative Density Function (CDF) based normalization, corresponding to function η in Alg. 1. For returns conditioning, we computed the discounted cumulative task reward in the original trajectory starting from the timestep in which the MADiff trajectory was sampled and then divided it by the maximum task reward that agents can receive, *i.e.*, 20 in our case corresponding to a soup delivery. The reason for considering the whole episode instead of just the forward horizon for computing returns is that this way we denote the future potential of the sampled trajectory in achieving the task's goal (delivering soups), and not just what happens within the sample window itself. For the other conditioning variables, *i.e.*, observer profile, and character and mental embeddings, we normalized them using CDF similarly to observations. In summary, a single datapoint used for training the diffusion model consisted of a window of augmented observations, a return, an observer profile vector, and a character and a mental embedding.

We trained the MADiff model with a history $C = 64$, horizon $H = 16$, and $K = 200$ diffusion steps, and an embedding and hidden dimensions of 128 and 256, respectively. We used epsilon noise prediction and loss computed as in Eq. 1. We used dropout for conditions with a dropout rate of 0.25 and conditional guidance with a factor of $\omega = 1.2$. We used 42 000 episodes, corresponding to 2 000 episodes per team (total of 21 agent pairs). We used Adam optimizer with a batch size of 32, learning rate of 2×10^{-4} and trained each MADiff models for a total of 10^6 training steps. For the different experiments reported in the paper, each MADiff model was trained with different conditioning variables, $y(\tau_i)$.

A.5.3 Losses. The integrated MADiff model, including the noise model δ and inverse dynamics model ξ, is trained to optimize the loss $\mathcal{L}(\delta, \xi)$ [31]:

$$\mathcal{L}(\delta, \xi) = \mathcal{L}_{\text{Diff}}(\delta) + \mathcal{L}_{\text{Dyn}}(\xi) \tag{1}$$

$$\mathcal{L}_{\text{Diff}}(\delta) = \mathbb{E}_{k,\boldsymbol{\tau}^0,\beta}\left[\|\epsilon - \epsilon_\delta(\hat{\boldsymbol{\tau}}^k, (1-\beta)y(\tau_i^0) + \beta\varnothing, k)\|^2\right] \tag{2}$$

$$\mathcal{L}_{\text{Dyn}}(\xi) = \mathbb{E}_{(o_i,a_i,o_i')\in\mathcal{D}}\left[\|a_i - I_\xi(o_i, o_i')\|^2\right] \tag{3}$$

where $i \sim \mathcal{U}(N)$ is the observer, $\boldsymbol{\tau}^0 \in \mathcal{D}$ is a joint trajectory sample where τ_i^0 is the observer's portion of it, $\beta \sim \text{Bern}(p)$ balances conditional ($y(\tau_i^0)$) and unconditional ($\varnothing$) diffusion training, and ϵ_δ is the noise model optimizing the surrogate loss [11], which estimates the noise $\epsilon \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ added to sample $\boldsymbol{\tau}^0$ at denoising step $k \sim \mathcal{U}(1, K)$.

A.6 Experiments

Table A.2 presents the numeric data of the plots in Figs. 4a and 4b of the replanning experiment.

Table A.2: Impact of various replanning schemes on the agents' cumulative task and individual rewards.

Condition	Plan Count	Task Rwd	Indiv. Rwd
Always	200.00 ± 0	22.72 ± 2.95	−1.08 ± 6.65
10 Steps	23.00 ± 0	18.24 ± 2.58	−21.29 ± 9.06
Horizon	4.00 ± 0	13.76 ± 2.07	−28.20 ± 9.15
Dynamic	64.89 ± 3.82	23.52 ± 2.89	−1.74 ± 6.90