Supplementary Material
Neural Amortized Inference for Nested Multi-agent Reasoning

Kunal Jha[1], **Tuan Anh Le**[2], **Chuanyang Jin**[3], **Yen-Ling Kuo**[4], **Joshua B. Tenenbaum**[5], **Tianmin Shu**[5,6]

[1]Dartmouth College, [2]Google Research, [3]New York University, [4]University of Virginia, [5]Massachusetts Institute of Technology, [6]Johns Hopkins University

kunal.a.jha.24@dartmouth.edu, tuananhl@google.com, cj2133@nyu.edu, ylkuo@virginia.edu, jbt@mit.edu, tianmin.shu@jhu.edu

Derivation of Eq. (4)

First, we have

$$\begin{aligned}
&p(is_{i,\ell}^{1:t}, o_i^{1:t} | a_i^{1:t-1}) \\
&= p(s^{1:t}, b_{j,\ell-1}^{1:t-1}, \theta_j, o_i^{1:t} | a_i^{1:t-1}) \\
&= p(o_i^{1:t} | s^{1:t}) p(\theta_j) p(b_{j,\ell-1}^{1:t} | is_{i,\ell}^{1:t-1}, o_i^{1:t}, a_i^{1:t-1}) \\
&\cdot \sum_{a_j^{1:t-1}} p(s^{1:t} | a_i^{1:t-1}, a_j^{1:t-1}) \pi_{j,\ell-1}(a_j^{1:t-1} | b_{j,\ell-1}^{1:t-1}, \theta_j). \quad (1)
\end{aligned}$$

If we assume that the observation of an agent is deterministic given a state and that the prior of θ_j is a uniform distribution, then we can simplify the above equation as

$$\begin{aligned}
&p(is_{i,\ell}^{1:t}, o_i^{1:t} | a_i^{1:t-1}) \\
&= p(b_{j,\ell-1}^{1:t} | is_{i,\ell}^{1:t-1}, o_i^{1:t}, a_i^{1:t-1}) \\
&\cdot \sum_{a_j^{1:t-1}} p(s^{1:t} | a_i^{1:t-1}, a_j^{1:t-1}) \pi_{j,\ell-1}(a_j^{1:t-1} | b_{j,\ell-1}^{1:t-1}, \theta_j) \quad (2)
\end{aligned}$$

As discussed in the main paper, we have

$$p(b_{j,\ell-1}^{1:t} | is_{i,\ell}^{1:t-1}, o_i^{1:t}, a_i^{1:t-1}) = q_\phi^\ell(b_{j,\ell-1}^{1:t} | is_{i,\ell}^{1:t-1}, o_i^{1:t}, a_i^{1:t-1}). \quad (3)$$

Therefore,

$$\begin{aligned}
w_t &= \frac{p(is_{i,\ell}^{1:t}, o_i^{1:t} | a_i^{1:t-1})}{q_\phi^\ell(is_{i,\ell}^{1:t} | o_i^{1:t}, a_i^{1:t-1})} \\
&= \frac{\sum_{a_j^{1:t-1}} p(s^{1:t} | a_i^{1:t-1}, a_j^{1:t-1}) \pi_{j,\ell-1}(a_j^{1:t-1} | b_{j,\ell-1}^{1:t-1}, \theta_j)}{q_\phi(s^{1:t} | o_i^{1:t}, a_i^{1:t-1}) q_\phi^\ell(\theta_{j,\ell} | o_i^{1:t}, a_i^{1:t-1})}. \quad (4)
\end{aligned}$$

Implementation Details for Construction Environment

Planners

We adapt BFS with a set of heuristics for the policies for both agents. For Alice's policy, she imagines the resulting state of taking each one of the actions in the action space. She then estimates the value of that action by assigning it to be the length of the shortest path from the resulting state to the block she is missing from her pair (path found with BFS). Alice then chooses to perform the action with the maximum utility. Bob similarly transitions to all possible immediate next states by enumerating over the actions space. His policy then diverges contingent on whether he is helping or hindering Alice:

If he is helpful and does not have any items in his inventory, he values a possible next state by the length of the shortest path to the missing block from the pair he infers Alice wants to form. If he has a block in his inventory, he values the next states by the shortest path to Alice.

If he is trying to hinder Alice and has nothing in his inventory, he determines the shortest path to the midpoint of Alice and the block she would like to move toward. He also determines the shortest path from his location to the block he infers Alice wants, as this is consistent with the behavior of stealing blocks and running away. If he grabs a block, Bob values the next states by determining how far they move him away from Alice and toward free space (away from walls and toward the centers of quadrants on the grid).

Neural Network Architectures

State Representation We represent the state of the construction environment by using multiple channels corresponding to different types of items. We end up with a 3-dimensional tensor to represent each state, with the dimensions $20\times20\times24$, such that each cell in the 20×20 gridworld is represented as a one-hot vector. The vector representation of each cell indicates whether a cell has a wall, is empty space, is Alice without any blocks selected in her inventory, is Alice with one of 10 possible blocks in her inventory, is Bob without any blocks grabbed, or is Bob with one of 10 possible blocks he can grab.

Action Representation We represent the action an agent takes from a particular state as a one-hot vector associated with the following actions Up, Down, Left, Right, Put-down, Stop. An agent grabs a block automatically by moving onto the cell the block is located without having anything in its inventory, assuming another agent is not already at that coordinate. Thus, we do not need to explicitly encode this ac-

tion and only need to give an agent the ability to drop items in its inventory.

State-Action Encoder Module We begin by expanding the tensor representation of an agent's action to be the same dimensions as the state tensor, then concatenate the two. From there, the state-action pair is encoded into a 128-dim hidden state by a convolutional layer with 128 channels and kernels of 1×1 with stride 1, and pass this through three fully connected (FC) layers (128-dim, 256-dim, 128-dim in order). For all three FC layers, we apply a ReLU activation function to obtain our final encoded state-action representation.

State Recognition Network, $q_\phi(s|\cdots)$ The state is fully observable in this environment. So we do not need to train a state recognition network for this environment.

Level-1 Goal Recognition Network, $q^1_\phi(\theta|\cdots)$ This network infers the goal of Alice (as a level-0 agent). Every (state, action) pair within a sequence of Alice's observed states and actions is passed through the state-action encoder described above to obtain a joint vector representation. From there, we concatenate each joint-vector representation and pass them through an LSTM with 128 hidden units. Since this is done for multiple different episodes within a single batch, we pad all the distinct sequences to have the same length as the longest rollout in the batch. After concatenating the padded output for each item in the batch, we pass the fused tensor through two FC layers of 256-dim and 128-dim in order, again using ReLU activations on both. Finally, we apply an FC layer with an output of 45 to the result from the previous layer and apply a log-softmax activation to this output. This final vector represents the probability distribution over all combinations of two blocks Alice might want to put together as part of its goal. We obtain this proposed distribution for every time step within an episode and every episode within a batch.

Level-2 Goal Recognition Network, $q^2_\phi(\theta|\cdots)$ This network infers the goal of Bob (as a level-1 agent). This was a nearly identical setup to the Level-1 Goal Recognition Network, with the main difference being the output of the final layer was of size 2 instead of 45, representing Bob's ability to only have 2 goals (helping or hindering Alice). Note that it has its own state-action encoder, which is not shared with the Level-2 goal recognition network. Since there are only 2 possible goal hypotheses for Bob, we always enumerate all 2 possible hypotheses instead of sampling top hypotheses based on the level-1 goal recognition network in the experiment. However, this network would be useful when there is a larger goal space for the level-1 agent.

Training

Data Generation For training q^1_ϕ, we initialize a state in which Alice is spawned at a random cell in the world (without Bob), and the blocks are scattered randomly throughout the gridworld. Alice is assigned two blocks to put together, and the walls are consistently bordering the grid. The choice of block locations and the pair Alice would like to create are sampled from a uniform distribution. Using the aforementioned policy, Alice tries to take the shortest path to move her two desired blocks together before some maximum number of time steps (40) pass. The state at each timestep and the action Alice took from there are saved as their aforementioned tensor representations, as well as a one-hot encoding representing which of the 45 block pairs Alice would like to bring together. We store the results of doing exact inference about Alice's intentions by enumerating all 45 possibilities, and this forms the target for q^1_ϕ. We generate 30,000 episodes for training.

For training q^2_ϕ, we follow a similar procedure, except that we also spawn Bob in a random location that is not occupied by Alice or a block. We also uniformly sample Bob's goal (helping or hindering Alice). Bob is assumed to be reasoning about which two blocks Alice would like to move together through our approach with a trained q^1_ϕ and 5 particles to enumerate over at each time step. He acts in accordance with his previously described policy, contingent on his beliefs about Alice's goals. In this scenario, we save Bob's actions from each state, and a one-hot representation of his social intention from the set helping, hindering. We store the results of doing exact inference about Bob's intentions by enumerating the 2 possibilities, and this forms the target for q^2_ϕ. We generate 1020 episodes to train the level-2 goal recognition network.

Loss ans Hyperparameter Specification Goal recognition networks are trained with KL divergence loss as defined in the main paper. For all networks, we use the Adam optimizer and a learning rate of 0.0001. We found the best results using a batch size of 128. All networks are trained on a single GPU.

Hypothesis Sampling

Sampling Hypotheses From Goal Recognition Networks. We rank the hypotheses based on their probabilities in q^l_ϕ and select the top hypotheses as the sampling result. This ensures that the top hypotheses will always be considered in the inference.

ToMnet Baseline

The neural network baseline for goal inference modeled after ToMnet has the same architecture as the level-2 goal recognition network. However, it is trained with cross-entropy loss based on the ground-truth goals. We used the same learning rate and batch size as the q^1_ϕ and q^2_ϕ networks.

Implementation Details for Driving Environment

Planners

All drivers rely on a hierarchical planner. The high-level planner decides whether to move along the optimal path, stop, or signal danger at each step (i.e., a subgoal). Note that the level-0 drivers will not consider signaling danger as they do not model other drivers. Based on the inference of other drivers, it will stop if it anticipates collision with other drivers at any point in the future 10 steps; it will signal

danger if it infers that another driver is not aware of a car and is about to collide with that car; otherwise, it will move along the optimal path. For the low-level plan, we search for the shortest path for the "moving along" subgoal; the low-level plan for "stop" is selecting the "brake" action; and the low-level plan for "signal danger" is selecting the "signal danger" action.

For the path planner, specifically, if a car is next to a set of walls, it moves forward. If it does not have any walls in front of it or at its side (meaning it has entered the intersection), it begins turning in the direction of its goal. Note that for the actions, "accelerate", "brake," "rotate left," and "rotate right," we implement built-in motion control to execute them in the environment. Therefore we do not need to consider motion planning for the drivers. In particular, the "brake" action immediately stops the car from moving, and the "accelerate" action immediately accelerates the car to a constant speed.

Simulating Inattentive Drivers

We model inattentive drivers as drivers with a small field of view and poor forward projection capabilities. Specifically, we implement inattentive drivers as level-1 drivers who have 45 degrees of view, no chance of signaling for danger, and only look one step into the future. This contrasts with normal drivers who have 135 degrees of view, will consider other drivers' beliefs for "signal danger" actions, and will look 10 steps into the future.

Neural Network Architectures

Belief State Representation We sample each driver's initial location uniformly from a set of 16 possible locations at an intersection as shown in Figure 1. We used a single 145-dim vector to represent the state, concatenating information about each driver's (existence, x coordinate, y coordinate, heading angle, one-hot vector for its previous action). If a driver's existence was given a value of 0, the other telemetry data points relevant to that vehicle were also given values of 0. The one-hot action representation is described below. The final element in the vector is the current time step of the world.

Action Representation We represent an action as a one-hot vector over the action space {Accelerate, Rotate Left, Rotate Right, Brake, Signal}

Inference Pair Representation This is a 2-dim vector, indicating two drivers' IDs, i and j. i is inferring j (as a lower-level driver) in this pair.

Belief State Encoding Module We decrease the size of the Belief State tensor by passing the Belief State through 5 fully connected (FC) layers (size 32, 64, 64, 32, 16 in order). Each layer uses ReLU activations.

Inference Pair Encoding Module We increase the size of the Inference Pair tensor by passing it through 3 FC layers (size 32, 32, 64 in order). Again, each layer uses ReLU activations.

State Recognition Network, $q_\phi(s|\cdots)$ This network infers the state of the world from a car's perspective (as a level-0 driver). Every (state, action) pair within a sequence of Alice's observed states and actions is represented as a Belief state tensor as described above and passed through the Belief State encoder to obtain a smaller vector representation. We then split the training about the beliefs of a driver into two networks: one for determining whether a driver exists in the world and one for determining the telemetry data for each driver. For the telemetry data network, we pass each vector through an LSTM with 16 hidden units. Since this is done for multiple different episodes within a single batch, we pad all the distinct sequences to have the same length as the longest rollout in the batch. After concatenating the padded output for each item in the batch, we pass the fused tensor through four FC layers of 32-dim, 64-dim, 32-dim, and 48-dim in order, again using ReLU activations on both. This final vector represents the x-coordinate, y-coordinate, and speed of each car within a state at a single time step. We obtain this predicted telemetry data for every time step within an episode and every episode within a batch. The network that proposes a driver's belief about the existence of other cars in the world has the same LSTM structure as the telemetry prediction network, but it passes the output through three FC layers of sizes 16, 32, and 32 in order. The first two of these FC layers utilize ReLU activations. The output of the final layer represents the (probability of a car not existing, the probability of a car existing) for each of the potential 16 vehicles. We reshape the 32-dim vector into a 16×2 tensor, then perform a log softmax activation on the second dimension. Again, we obtain this output for every time step for every episode within a batch.

Level-1 Goal Recognition Network, $q^1_\phi(\theta|\cdots)$ This network infers the goal of a specific car (as a level-0 driver) from another's perspective. Every (state, action) pair within a sequence of the inferring car's observed states and actions is passed through the Belief State encoder described above. We also pass the IDs of the reasoning car and its target as an Inference Pair tensor into the Inference Pair Encoding module described above. We concatenate the outputs of both of these encoders to develop a joint-vector representation of the task. From there, we concatenate each joint-vector representation and pass them through an LSTM with 128 hidden units. Since this is done for multiple different episodes within a single batch, we pad all the distinct sequences to have the same length as the longest rollout in the batch. After concatenating the padded output for each item in the batch, we pass the fused tensor through two FC layers of 256-dim and 128-dim in order, again using ReLU activations on both. Finally, we apply an FC layer with an output of 3 to the result from the previous layer and apply a log-softmax activation to this output. This final vector represents the probability distribution over all directions the target car might want to turn in to accomplish its goal. We obtain this proposed distribution for every time step within an episode and every episode within a batch.

Level-2 Goal Recognition Network, $q^2_\phi(\theta|\cdots)$ This network infers the goal of one car (as a level-1 driver) from an-

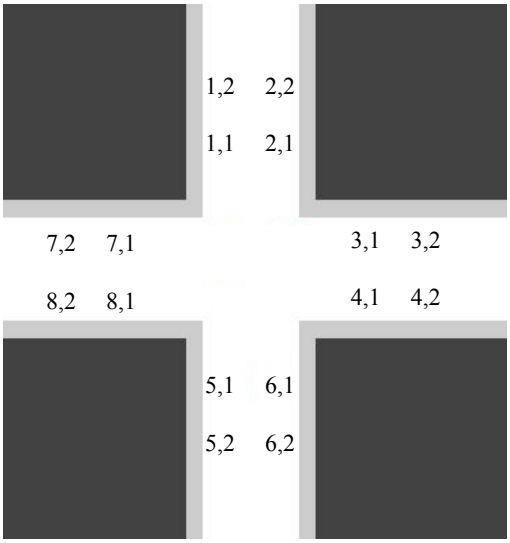

Figure 1: Indices of each car represented in a driver's belief in **Driving**.

other's perspective. This was an identical setup to the Level-1 Goal Recognition Network, with the main difference being the input would include a slightly larger effective actions space (since level-1 cars are capable of signaling danger). Note that it has its own Belief State State and Inference Pair encoders, which are not shared with the Level-1 goal recognition network.

Training

Belief Representation and Sampling For each driver's belief, we represent the world state as the states of the cars in all 8 lanes. As introduced in the main paper, for the k-th lane, we model up to 2 cars that are closest to the intersection $m \in \{1, 2\}$, as shown in Figure 1. The indices of cars, m, are ordered by their distances to the intersection. k, m indicates whether a car m exists in the k-th lane.

For each driver's belief, we sample the existence and telemetry data for all unobserved cars as follows:

1. If a car is important to the driver's turning objective, but not visible, its existence is sampled from a prior of $p(existence = 1) = 0.65$. We assume that the car is at the location within the blind spot that is closest to the driver and it is moving forward at a default speed.
2. For unobserved and unimportant cars, we assume that they do not exist (i.e., the driver will ignore them).

A sampled belief state is processed into the aforementioned state tensor representation. The importance of a car is determined by the following heuristics:

1. If a driver is turning left, the driver needs to make sure that no car is crossing the street perpendicular to the driver's current lane, and that no cars traveling parallel to the driver's current lane in the opposite direction.
2. If a driver is turning right, the driver needs to make sure that no car is crossing the street perpendicular to the right lane the driver wants to turn into, and that no car is turning left into the lane the driver wants to turn right into.
3. If a driver is moving forward, the driver needs to make sure that no car is crossing the street perpendicular to the driver's current lane and that no car in the lane parallel to the driver but on a separate side of the street is turning left.

This means that we need to only consider up to 3 lanes and we only would consider the first car that can potentially exist in the blind spot in one of these 3 lanes. At any moment, there are at most 3 relevant vehicles that a driver cannot see which it needs to form a belief over. For each of the 3 vehicles, we must sample whether it exists or not. Therefore, we need at most 8 particles to include all hypotheses about the state. These heuristics can be replaced with learned attention mechanisms for deciding which cars are significant to one's planning process, which we intend to investigate in the future.

Data Generation For training q_ϕ, we initialize a state in which a car is spawned randomly in one of 16 possible locations, and its goal of moving forward or turning left/right is sampled from a uniform distribution. Buildings on either side of the car's initial location inhibit its vision of the world, rendering this a standard POMDP. We constrain each episode to a maximum number of time steps (30). 2 more drivers independently acting in the world are also generated randomly in the world, and their goals are sampled uniformly from the goal space. The state at each time step and the action the driver took from there are saved as their aforementioned tensor representations. We also save the ground truth belief a driver forms about the physical state of the world. At this level, drivers move by only being concerned with whether or not they will personally crash, and trying to navigate toward their goals accordingly. If any car in the world crashes, the episode terminates.

For training q_ϕ^1, we follow a similar procedure. Each car is assumed to be reasoning about the state of the world with a trained q_ϕ and 1 particle for the inferred physical state out of 8 potential options to enumerate over at each time step. They then try to take the best actions to avoid crashing into each other while completing their objective. Once a rollout terminates, we have a model perform exact inference over some car B's intentions from the perspective of another car A. This produces a tensor which is $(T \times 3)$, where T is the number of time steps in a rollout and the vector for each time is the probability of B wanting to move forward, left, or right from A's perspective.

Training q_ϕ^2 is a similar process as before, but each driver is now assumed to be reasoning at level-1 through the use of our approach and the trained q_ϕ^1 and q_ϕ networks. The policy for these level-1 drivers is described in prior sections. In this scenario, we save each car's actions from each state, the tensor representation of each car's inference pair as aforementioned, and the results of doing exact inference at level-2, while performing our approach using q_ϕ^1 and q_ϕ at level-1.

At level-2 inference, the exact inference over the entire hypothesis space would require 72 hypotheses: (3 particles to cover all level-2 goal hypotheses) $\times$ (3 particles to

cover all level-1 goal hypotheses) $\times$ (8 particles to cover all level-0 state belief hypotheses) = 72 total particles.

We generate 6,000 episodes at random for each of the datasets used to train q_ϕ and q_ϕ^1, and q_ϕ^2.

Loss and Hyperparameter Specification The telemetry prediction network that is part of q_ϕ was trained using the MSE loss between its proposed telemetry values for each driver and the ground truth information stored within our beliefs. The driver existence prediction network is trained to minimize the KL divergence loss between its proposed distribution and the ground truth distribution based on a driver's observations and its prior. Both of these models performed best with a batch size of 32 and a learning rate of 0.001, using the Adam optimizer. The goal recognition networks are trained using the KL divergence loss between its proposed distribution and the ground truth distribution based on exact inference. We use the Adam optimizer and a learning rate of 0.0001 for both goal recognition networks. We found the best results using a batch size of 64. All networks are trained on a single GPU.

Hypothesis Sampling

Sampling Hypotheses From Goal Recognition Networks. We rank the hypotheses based on their probabilities in q_ϕ^l and select the top hypotheses as the sampling result. This ensures that the top hypotheses will always be considered in the inference.

Sampling Hypotheses From State Recognition Networks. Given the probability of existence that our recognition network assigns to each potential car, we sample whether that car will exist in our belief about the physical world and assign it its corresponding telemetry data.

ToMnet Baseline

The neural network baseline for action prediction modeled after ToMnet has the same architecture as the level-2 goal recognition network, but has an output size of $T \times 5$ instead, to represent the 5 possible actions a driver can take. It is trained with cross-entropy loss based on the ground-truth actions at the next time step from any given state and relies on the same dataset used to train q_ϕ^2. We used the same batch size of 64 as the q_ϕ^2 model, but a lower learning rate of 0.00001, since this yielded a higher test accuracy for the baseline.

Additional Qualitative Results

Construction Environment

Figure 2 and Figure 3 show our method's inference on two more examples in **Construction**.

Driving Environment

In Figure 4, we examine a specific frame (step 1) within a **Driving** scenario involving 3 cars controlled by normal drivers. We visualize the nested reasoning by our model that leads to the correct prediction that the green car will signal danger.

Because this scene occurs early in an episode and the green car has not reached the intersection, its previous action of accelerating is consistent with all 3 possible goals. This is reflected in a uniform distribution over θ_{green} at the highest level when we use 3 particles (level-2 inference).

Going down one level, at level-1 inference, based on the inferred belief, the green car believes that there could be two cars in its blind spots, i.e., the purple and orange cars on the middle level in Figure 4. At this level, we also infer how green infers the goals of the red car and the blue car, as shown in the corresponding distributions over θ_{red} and θ_{blue} in Figure 4. We use 2 particles for the level-1 inference. For the blue car, two goals are sampled: moving forward or turning left. These two goals are equally possible since there is little observation at this moment. There is a confident goal prediction for the red car as the red car has rotated left, which strongly indicates the goal of turning left.

If we look at the leaves of the tree (level-0 reasoning), we can get a better understanding of why the model predicts that the green car will signal danger. In the green car's mind, the red car may be unaware of other cars, as the red car cannot see other cars. Moreover, the green car infers that the blue car may be also unaware of the red car. Coupled with the inferred goals of the red car and the blue car, our model infers that the green car may predict a potential collision between the red car and the blue car. Therefore, our model predicts that the green car will signal danger.

Our approach generates similar "inference trees" at every time step. With the use of neural amortized inference, we only need to sample a small tree to conduct the inference accurately. This demonstrates that our approach is both interpretable and efficient.

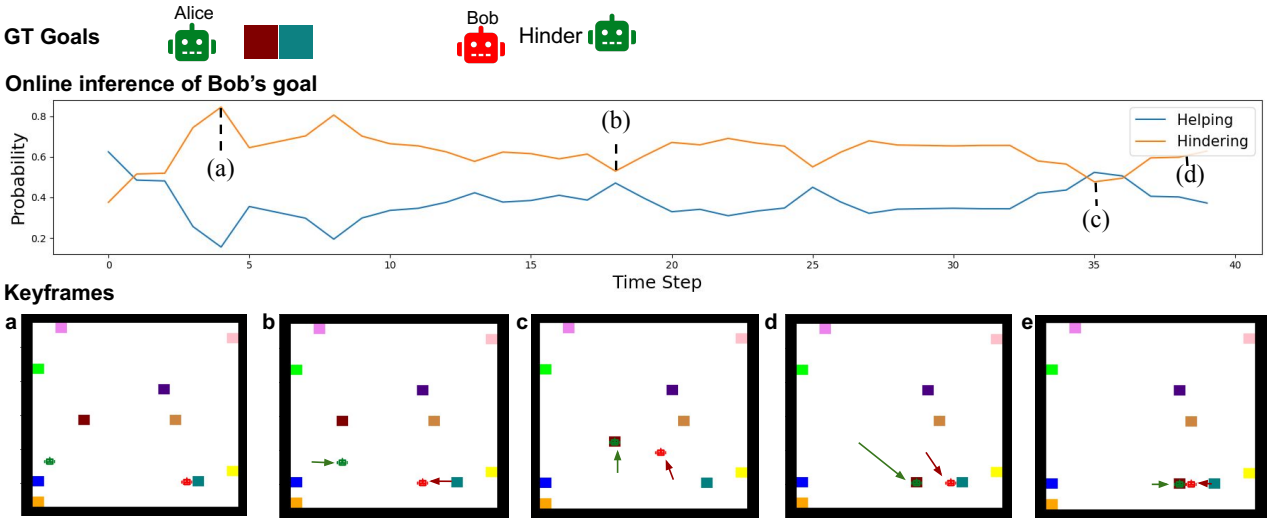

Figure 2: Online goal inference of our method (with 10 particles) in a typical episode in **Construction**, in which Alice wants to put brown and green blocks together and Bob tries to hinder Alice. The plot on the top shows the posterior probabilities of the two hypotheses based on our method's inference at any given time step. The keyframes on the bottom explain why our method adjusts its inference. The arrows in the frames show the trajectories of the agents. Initially, the model is uncertain (**a**). (**b**) As Bob begins to move toward Alice to intercept her rather than toward other blocks, our model starts to make a confident prediction that Bob is hindering. (**c**) Over time, our model maintains this prediction as Bob continues to exhibit hindering behavior. (**d**) The inference becomes uncertain at this moment as Bob moves very close to the block Alice wants (the green) after she has picked up the brown block, which could be either helping or hindering. (**e**) The inference becomes confident again when Bob moves to intercept Alice along her path to the green block and prevent her from completing her task.

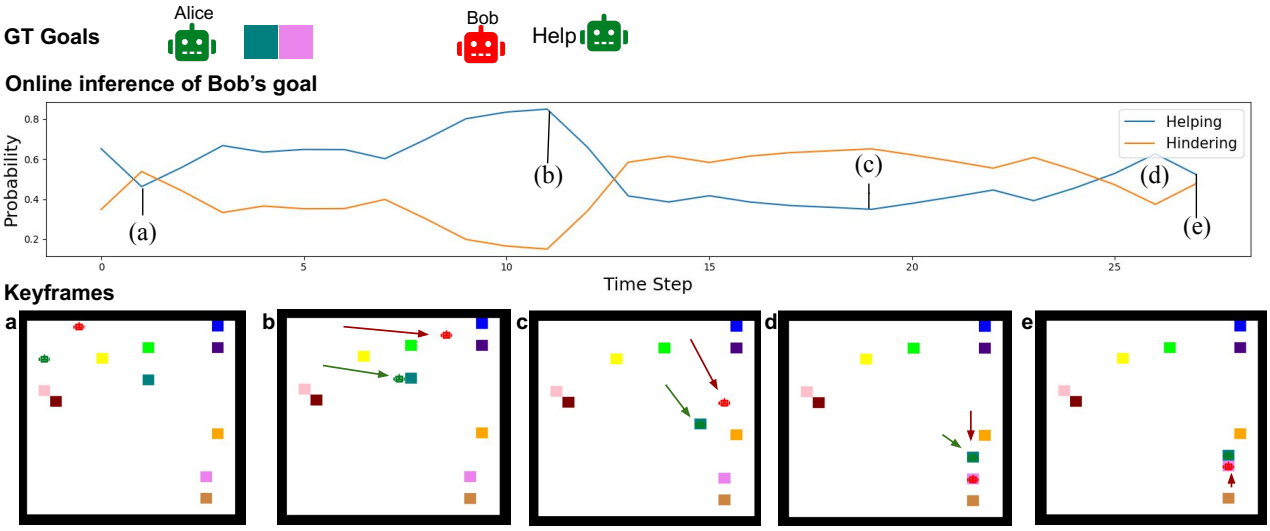

Figure 3: Online goal inference of our method (with 10 particles) in a typical episode in **Construction**, in which Alice wants to put green and magenta blocks together and Bob tries to help Alice. The plot on the top shows the posterior probabilities of the two hypotheses based on our method's inference at any given time step. The keyframes on the bottom explain why our method adjusts its inference. Note that the arrows in the frames show the trajectories of the agents. (**a**) our model is initially uncertain about Bob's goal. By (**b**), however, the model believes that Bob is likely to be helping since he has never tried to sabotage Alice once. (**c**) the model infers that hindering is slightly more likely since Bob's behavior could be interpreted as trying to block Alice or trying to grab the 2nd block before Alice can. However, by (**d**) the inference switches back to favoring helping as Bob grabs the block missing from Alice's pair and moves toward her, helping her complete her goal faster. (**e**) The slight dip in confidence at the end reflects Bob putting down the block, which can be confusing if he is doing this to allow Alice to pick up that block or so that he can hinder her by grabbing the other block from her and running away.

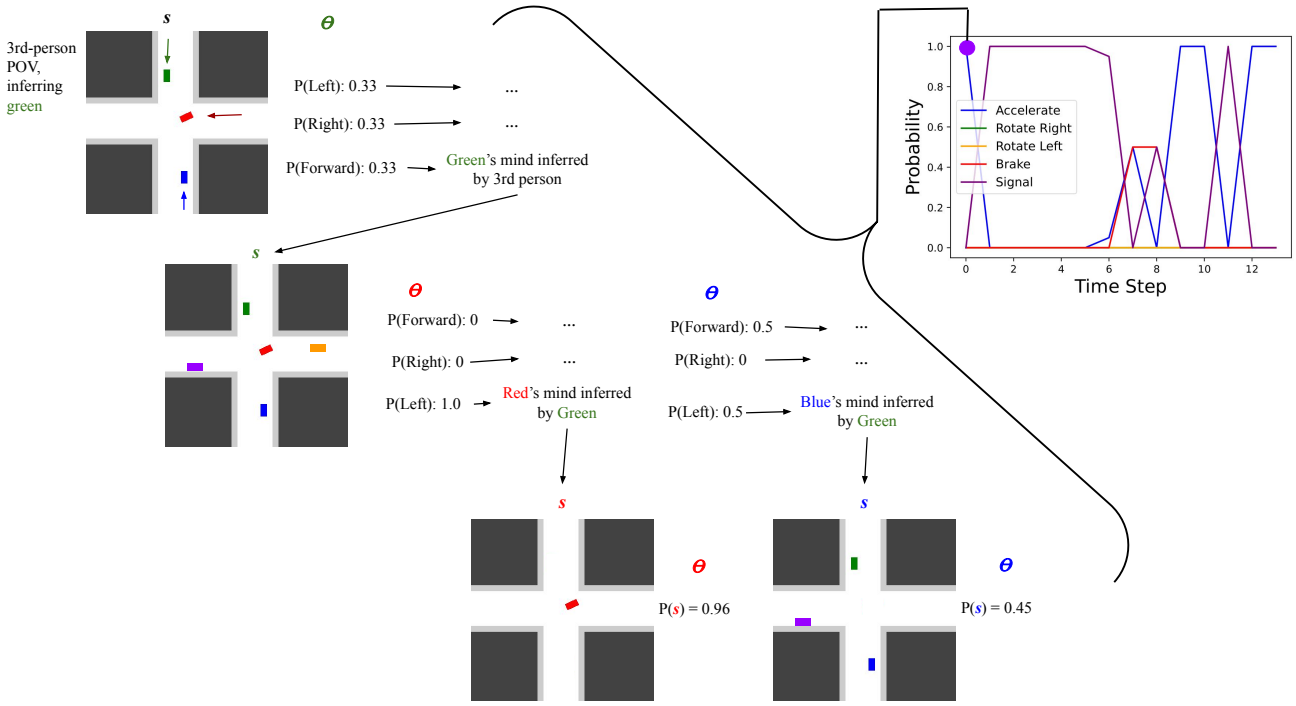

Figure 4: The shortened tree representation of nested inference done by our model's inference about the green driver's mind at time step 1 using 6 particles. Though uncertain about the green car's goals early on in the episode, our model correctly predicts that it will signal danger, as the leaves of the tree indicate that the green car does not believe that the blue or red cars can see each other and will consequently crash. The corresponding action probabilities our model proposes for the green car are depicted in the top right of this figure.