Supplementary for:
MARPLE: A Benchmark for Long-Horizon Inference

The MARPLE website is at: `https://marple-benchmark.github.io/`.

The appendix is organized as the following. In Appendix A, we present details about the benchmark and inference scnearios. In Appendix B, we present details about the hierarchical simulator used to generate multimodal evidence and trajectories. In Appendix C, we provide details about our dataset and access. In Appendix D, we provide details on the computational resources and experiment details. In Appendix E and Appendix F, we present implementation details and ablations for the simulation method. In Appendix G, we present the prompts used for `GPT-4`. In Appendix H and Appendix I, we provide additional results benchmarking open-source LLMs and `GPT-4` with in-context learning. We include analysis of `GPT-4` reasoning in Appendix J. Lastly, in Appendix K, we present details on the human experiments.

A MARPLE Benchmark: Inference Scenarios

A.1 Overview

The MARPLE codebase can be found at `https://github.com/marple-benchmark/marple`. Our benchmark consists of 10 household missions paired to create a set of 5 inference scenarios, as shown in Table A.1. This provides a representative sample of the diversity and complexity possible by pairing missions.

Table A.1: Five inference scenarios in our benchmark, defined in terms of the inference question, agent `A`'s mission, and agent `B`'s mission. For these tasks, agent `A` is always the answer to the inference question. The tasks are in order of increasing difficulty, which is determined based on the average inference horizon and similarity between the two missions.

Inference Question	Agent A Mission	Agent B Mission	Avg. Horizon	Similarity
Who picked up the pillow?	Watch movie cozily	Watch news on TV	15	0.19
Who turned on the shower?	Take shower	Feed dog	26.4	0.30
Who picked up the snack?	Get snack	Clean living room table	36.8	0.46
Who picked up the plant?	Move plant at night	Get night snack	43.9	0.61
Who turned on the laundry?	Do laundry	Change outfit	51.3	0.87

A.2 Inference Scenario Setup

An inference scenario is defined by the missions performed by agents `A` and `B` and a query state. We provide details on the necessary components below:

Missions. We define 10 household missions: `Change Outfit`, `Clean Living Room Table`, `Do Laundry`, `Feed Dog`, `Get Night Snack`, `Get Snack`, `Move Plant at Night`, `Take Shower`, `Watch Movie Cozily`, `Watch News on TV`. These missions vary in the number of timesteps and types of actions. Each mission is defined by a list of subgoals, which we define next.

Subgoals. A mission's subgoal is a symbolic state that must be satisfied to complete the mission. It is represented as a dictionary with the keys "obj", "fur", "room", "pos", "action", "state", and "end_state." The "obj" and "fur" determine the target object type, "room" and "pos" describe the target location, and "action" is the action that the agent must perform on the target to result in the desired "state." The "state" is a tuple with the state name and boolean value, and "end_state" is `True` if the subgoal is the last one in the mission and `False` otherwise.

We provide an example of a mission and subgoal representation in Figure A.1.

Inference Scenario. To construct an inference scenario, we pair two missions (e.g., `do laundry` and `change outfit`) and select a query state unique to one agent (e.g., `Pickup(sandwich)` = `True`). The corresponding inference question is: "Which agent is more likely to have `[state action]` the `[state object]`?" For instance, if the query state is `Pickup(sandwich)`, the question would be: "Which agent is more likely to have `picked-up` the `sandwich`?"

A.3 Inference Scenario Difficulty

We identify two key factors that affect the difficulty of an inference scenario: the average inference horizon and the similarity between the two missions.

Mission and Subgoal Representation

```
Example of a Mission:  list of subgoals
get_night_snack = [
   toggle-on-*-light-Kitchen,
   open-*-*-electric_refrigerator-Kitchen,
   pickup-*-sandwich-electric_refrigerator-Kitchen,
   close-*-*-electric_refrigerator-Kitchen,
   toggle-off-*-light-Kitchen,
   drop-*-sandwich-table-Bedroom
]

Example of a Subgoal:  tuple with subgoal name, subgoal dictionary
(
  “toggle-on-*-light-Kitchen”,
  {
     “obj”:  None,
     “fur”:  “light”,
     “room”:  “Kitchen”,
     “pos”:  None,
     “action”:  “toggle”,
     “state”:  [“toggleable”, 1],
     “can_skip”:  False,
     “end_state”:  False
  }
)
```

Figure A.1: Example of a mission and subgoal representation, for the mission: `Get Night Snack`.

Inference Horizon. The inference horizon is the number of steps that it takes for agent `A` to reach its inference state. As the inference horizon increases, difficulty increases because models must understand and predict more future steps. The uncertainty in predictions also compounds over time, leading to greater prediction errors and variation in possible outcomes.

Mission Similarity. An inference scenario becomes more challenging when the two agents have similar trajectories, which are largely determined by their missions' subgoals. Thus, we define the similarity between a pair of missions, M_1 and M_2, as follows:

$$\text{similarity}(M_1, M_2) = \frac{1}{1.5}\left(\frac{|M_1 \text{ subgoal actions} \cap M_2 \text{ subgoal actions}|}{M_1| \text{ subgoal actions} \cup M_2 \text{ subgoal actions}|} + 0.5\frac{|M_1 \text{ subgoal rooms} \cap M_2 \text{ subgoal rooms}|}{M_1| \text{ subgoal rooms} \cup M_2 \text{ subgoal rooms}|}\right)$$

Our chosen set of inference scenarios represents a range of similarities, as shown in Table A.2.

Table A.2: Similarity of all possible pairs by combining the 10 missions. Of these pairs, the similarity ranges from 0.19 to 0.87. Our chosen set of inference scenarios is highlighted in blue, and they span a wide range of the similarity values to represent a range of difficulties.

	change outfit	clean living room table	do laundry	feed dog	get night snack	get snack	move plant at night	take shower	watch movie cozily	watch news on tv
change outfit	1.00	0.53	0.87	0.78	0.6	0.53	0.29	0.44	0.28	0.19
clean living room table	0.53	1.00	0.33	0.64	0.56	0.46	0.48	0.19	0.25	0.28
do laundry	**0.87**	0.33	1.00	0.46	0.56	0.74	0.64	0.71	0.64	0.56
feed dog	0.60	0.64	0.46	1.00	0.87	0.67	0.37	0.30	0.28	0.19
get night snack	0.64	0.56	0.56	0.87	1.00	0.78	0.61	0.61	0.37	0.29
get snack	0.53	**0.46**	0.74	0.67	0.78	1.00	0.64	0.61	0.55	0.46
move plant at night	0.29	0.48	0.64	0.37	**0.61**	0.64	1.00	0.35	0.55	0.60
take shower	0.44	0.19	0.71	**0.30**	0.42	0.61	0.35	1.00	0.70	0.62
watch movie cozily	0.28	0.25	0.64	0.28	0.37	0.55	0.55	0.70	1.00	0.19
watch news on tv	0.19	0.28	0.56	0.19	0.29	0.46	0.60	0.62	**0.19**	1.00

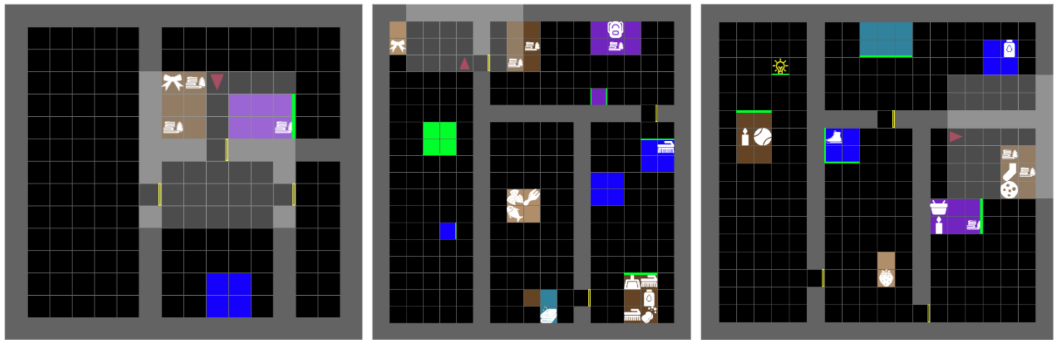

Figure B.1: Examples of the Visual Representation of the MARPLE Simulation Environment.

B MARPLE Household Simulator: Details

The MARPLE Household Simulator consists of two components: a multimodal simulator and a hierarchical agent planner.

B.1 Simulator: A Multimodal GridWorld Environment

The simulator is built on top of Mini-BEHAVIOR [22], a GridWorld environment that is fast, simple, and easy-to-use. It supports procedural generation of diverse environments, symbolic states, and high-level agent actions, making it suitable for simulating realistic, long-horizon tasks.

Our simulator inherits several features from Mini-BEHAVIOR, including the standard $m \times n$ grid layout and asset library of furniture and object classes, action space, and state space. The asset library statistics are in Table B.1.

Our simulator further extends Mini-BEHAVIOR to support multimodal stimuli as follows: **Visual.** The visual representation of our environment is a $m \times n$ grid of cells. We inherit Mini-BEHAVIOR's visualization of agents, objects, and furniture, which are represented as triangles, icons, and colored backgrounds, respectively. Each cell can contain an object and a furniture, and the furniture states are indicated by green borders along the cell edges.

Each environment state has a corresponding array and a scene graph representation. An $m \times n$ environment has a $m \times n \times 8$ array representation. The 8 channels indicate the room type, furniture type, furniture states, object types, object states, object ids, agent position, and agent direction at each cell.

Meanwhile, the scene graph representation is a standard scene graph with a set of nodes and directed edges. The nodes represent entities, and the directed edges represent physical relations between entities, such as object-object relations and object-room relations.

Language. Our simulator supports two kinds of language descriptions that can be generated by an agent: intent and testimony. An agent's intent describes an action that they are about to perform, e.g. *"I am going to open the closet in the Bedroom."* An agent's testimony provides information on previous state changes in the environment that it observed, e.g. *The clothes in the closet in the Bedroom were picked up.* The language descriptions are generated from templates which takes in the action and relevant room and objects.

Audio. To simulate the sounds produced by agent actions, we incorporate realistic audio recordings and define an action-audio mapping. The audio files are obtained from `https://freesound.org`, and they are clipped to be 1 second long.

Table B.1: MARPLE Household Simulator Elements Type Statistics.

Environment Elements			Behavior Elements		Engine Elements
Room Types	**Furniture Types**	**Object Types**	**Mission Types**	**Action Types**	**State Types**
6	22	82	10	10	18

```
{"Grid":{
    "auto": {
        "max_num_agent": 5,
        "room_split_dirs": ["vert", "horz"],
        "min_room_dim": 5,"max_num_room": 4}, (universal settings)
        "width": 15, "height": 15,
    "agents": {
        "num": 1,
        "Initial":[{
            "name": "A","gender": "male","pos": [13, 13],"dir": null,
            "color": "red","step_size": 1,
            "mission_preference_initial": {"get_night_snack": 1},
            "cur_mission": null,"cur_subgoal": null,"carrying": null}]}
        "rooms": {
            "num": 2,
            "Initial":[
                {"type": "Bedroom", "top": [1, 1], "size": [9, 13],
                    "furnitures":{"num": 2,"initial":[
                        {"type": "bed","state": null,"pos": [1, 1],
                            "objs": {"num": 1,"initial":[{
                                "type": "remote","state": null,"pos": null}]}},
                        {"type": "table","state": {"dustyable": 1},"pos": [6, 6],
                            "objs": {"num": 0,"initial": []}}]}},
                {"type": "Kitchen","top": [11, 1],"size": [3, 13],
                    "furnitures":{"num": 2,"initial":[
                        {"type": "light","state": {"toggleable": 1}, "pos":[12,3],
                            "objs": {"num": 0, "initial": []}},
                {"type":"electric-refrigerator", "state": {"openable": 1}, "pos":[12,10],
                            "objs": {"num": 1, "initial":[{
                                "type": "sandwich", "state":null,"pos":null}]}}
            ]
        }
    }
}
```

Figure B.2: Example of a simple configuration json file for the mission: `get night snack`.

B.2 Planner: A Hierarchical Planner for Agent Behavior Generation

The hierarchical planner generates agent trajectories given its `mission preferences`, a distribution over all possible missions. It consists of 3 components: a high-, mid-, and low-level planner.

High-Level Planner. The high-level planner first samples a mission according to the agent's mission preferences. If the current mission becomes infeasible at any point, the current mission terminates, and the high-level planner resamples a new mission.

Mid-Level Planner. Given a mission, the mid-level planner is a Finite State Machine that determines the next subgoal to accomplish. It is given the sequence of subgoals to accomplish, and it finds the first one in the sequence that has not been executed yet. If the first unaccomplished subgoal is not feasible, (e.g. there is no light in the Kitchen), the current mission terminates.

Low Level Planner. The low-level planner decomposes a subgoal into a sequence of agent actions to accomplish the subgoal, using the A-star algorithm. It generates the shortest path to navigate to the target object, positions the agent, and performs the specified action. The simulator then propagates the environment state based on these actions. When a feasible trajectory is found, the trajectory is saved; otherwise, the current mission terminates.

B.3 Usage: Ensuring diversity and complexity

Inference scenarios are procedurally generated according to a configuration file, as shown in Figure B.2. This file specifies initial conditions such as objects, states, and positions. Optional constraints include environment size, number of additional rooms, furniture, objects, and their positions.

The environment is first instantiated with the specified elements, and the additional ones are randomly selected from the asset library. They are placed randomly throughout the environment, resulting in diverse environment instances. The planner then generates agent trajectories within the environment.

To ensure complexity, the environment size, number of objects, number of rooms can all be scaled as needed. An $m \times n$ environment has a $m \times n \times 8$ state representation, causing the state space to grow exponentially with the array size.

C MARPLE Dataset

C.1 Dataset Details

Dataset description. We provide a dataset description in a datasheet: `https://github.com/marple-benchmark/marple/blob/main/datasheet.md`.

Link and license. The dataset is uploaded for public download at `https://drive.google.com/drive/folders/1zXsErNVOMYjBMWzTnmZS4e4aIljWlRce?usp=sharing`. It will be released under the CC-BY-4.0 license.

Author statement. The authors bear all responsibility in case of violation of rights. All dataset trajectories were collected by the authors and we are releasing the dataset under CC-BY-4.0.

Format. The data is uploaded in a simple zip format, with a zip file for each inference scenario in each train and test dataset. Upon decompressing the archive, a directory is provided for each instance that contains two subdirectories, one per agent. These are named with the agent's mission, and they contain files for the array and scene graph representations of each step in the trajectory, labelled by the timestep.

C.2 Data Generation

For each inference scenario, we provide training and testing datasets. Each testing dataset contains 500 paired trajectories, instantiated in 10 diverse, procedurally generated rooms. We provide two types of training sets, each containing 5000 paired trajectories. For one type, 500 trajectories are generated in each of the 10 testing environments. For the second, 5000 environments are procedurally generated with 1 trajectory each. The configuration files used to generate all of the data are provided in our codebase.

D Computational Resources and Experiment Details

D.1 MARPLE Simulator: Computational Resources

Our simulator operates at 600 frames per second (FPS) and requires only 1 frame for a primitive action. We run our experiments on the Stanford SC computational cluster with 1 NVIDIA TITAN RTX GPU and 8 CPU per job. With these resources, each inference trial takes 1.5 hours. The speed and efficiency of our simulator allows researchers to effectively evaluate their methods and focus on solving high-level, long-horizon inference challenges.

In contrast, a realistic physical simulator such as BEHAVIOR [26] runs at 60 FPS and requires 100 frames to perform a primitive action, making larger-scale experiments impractical. Such detailed physics simulation is also unnecessary for our inference setup, which focuses on understanding high-level agent behavior rather than physical interactions or photorealistic rendering.

D.2 Experiment Resource Requirements

We ran experiments on the Stanford SC computational cluster with 1 NVIDIA TITAN RTX GPU, 8 CPU, and 30 GB RAM for each job. With these resources, each trial for a mental-simulation baseline took 1.5 hours to run. Each trial for `GPT-4` took 1 minute to run and required 32 API calls, resulting in a cost of $11 * 8 * \$0.50 = \$44.00$ per trial.

We evaluate each baseline on 50 trials. Each mental-simulation baseline took 75 hours total (jobs were submitted in parallel), and we evaluate on 4 variants of the simulation baseline for a total of 300 hours. For GPT-4, the 50 trials took 1 hour and cost $2200. For humans, it took roughly 3 hours to complete the set of 50 trials.

D.3 Statistical Significance

We choose to evaluate on 50 trials. This provides a good balance between statistical power and computational resources, as performing inference for a single trial is resource-intensive.

We plot the inference accuracy across the 50 trials with 95% CI, as shown in Figure 4 and Figure 6. The error bars in Figure 4 and Figure 6 are calculated using the standard formula: $CI = \bar{x} \pm \frac{\sigma}{\sqrt{n}}$, where $\bar{x}$ is inference accuracy, σ is standard deviation, and $n = 50$ is the number of trials. Our figures indicate that 50 trials is sufficient, as the error bar is small enough to draw meaningful conclusions.

Algorithm E.1 Simulation with Monte Carlo Sampling and Learned Agent Models

1: **Input:** Observations of both agents o_τ^A, o_τ^B
2: **Output:** $P(A), P(B)$ that A or B caused s_T
3: Initialize $count \leftarrow 0$
4: **for** $i \leftarrow 0$ to $m-1$ **do**
5:     **for** $t \leftarrow \tau$ to T **do**
6:         Sample a_t^A according to $P(a|\pi^A, o_t^A)$
7:         Pass a_t^A to the simulator, obtain s_{t+1}^A, o_{t+1}^A
8:         **if** $s_{t+1}^A = s_T$ **then**
9:             $count \leftarrow count + 1$
10:             **break**
11:         **end if**
12:     **end for**
13: **end for**
14: $P(s_T|\pi^A, o_{0:\tau}^A) \leftarrow count/m$
15: Repeat 3-14 for agent B to get $P(s_T|\pi^B, o_{0:\tau}^B)$
16: Normalize using Equation (1) to get
$P(A), P(B) =$
softmax$(P(s_T|\pi^A, o_{0:\tau}^A), P(s_T|\pi^B, o_{0:\tau}^B))$

E Simulation with Learned Agent Models: Details

E.1 Algorithm

Algorithm E.1 is used to perform simulation with Monte Carlo sampling and learned agent models.

E.2 Implementation Details

Agent Model Architectures. We have four variations of our agent policy models: vision-only, audio-augmented, language-conditioned, and audio-augmented language-conditioned.

The vision-only and audio-augmented policy models are implemented with a Vision Transformer (ViT) as an encoder with a multi-layer perceptron (MLP) to predict the agent actions. After experimenting with different model and layer sizes, we use a ViT encoder with an image size of 20×20, patch size of 1×1, depth of 15, embedding dimension of 1024, 8 channels, and 16 heads and a 4-layer MLP with intermediate ReLU layers.

The language-conditioned and audio-augmented language-conditioned policy models are transformer-based with a ViT encoder and 4 decoders for the object, furniture, room, and action. Each decoder is a 2-layer MLP with an intermediate ReLU layer. After experimentation, we use a ViT encoder with an image size of 20×20, patch size of 1×1, depth of 15, embedding dimension of 1024, 8 channels, and 16 heads. Each decoder has an input dimension of 256, hidden dimension of 256, position embedding dimension of 64, depth of 8, dropout of 0.1, and gelu activation.

Agent Model Training Data. We learn agent models for all 10 of the provided missions. We train our agent models on two types of agent behavior datasets, as described in Appendix C.

Agent Model Training Details. We perform sweeps for hyperparameter tuning using WandB. Ultimately, we train our low-level policy models using a batch size of 64 and a learning rate of 1e-4, optimized with the Adam optimizer. The models are trained for 20 epochs, and this includes a gradual warmup scheduler with a multiplier of 1 and a warmup period of 4 epochs, followed by a cosine annealing learning rate scheduler over the remaining epochs. Additionally, we employ gradient accumulation to enhance the training efficiency and stability.

F Ablations of Simulation with Learned Agent Models

We provide extensive ablation of our simulation baselines, and we explore the effect of each modality (vision, audio, and language) on performance. These demonstrate that the vision-only baseline performs the worst, and the addition of audio and language are both beneficial. While language seems more valuable than audio in inference, the baseline using all 3 modalities consistently outperforms the others. This suggests that audio and language provide useful, distinct information in inference.

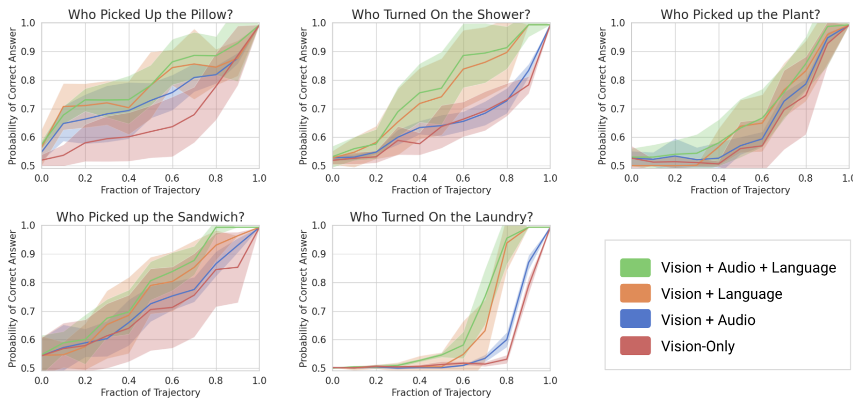

Figure F.1: Performance of each variant of the simulation baseline on all 5 inference scenarios. These baselines are tested in-distribution, on the same environments seen in training. The vision-only baseline performs the worst. While language seems more useful than audio, the baseline with all 3 modalities consistently outperforms the others. This suggests that both audio and language provide useful, distinct information.

G Prompts for `GPT-4`

We provide the prompt templates for `GPT-4`:

Prompt illustration for generating completions

```
Instructions:
Take a deep breath. Your task is to analyze and determine which agent (target agent,
other agent) is more likely to have performed specific actions leading to the final
state of the environment.

Remember, the states you are analyzing are select snapshots from a larger sequence.
If the agents have gone through e.g., 100 states, you might only be seeing a fraction
of these (like every 10th state for each agent), which means critical movements and
decisions may have occurred in the unseen states.

Initial State of Target Agent: [state here]

Current State of Target Agent: [state here]

Initial State of Other Agent: [state here]

Current State of Other Agent: [state here]

Final State: [state here]

Your analysis should consider how the changes and progression from the initial to
the current state for each agent might indicate their likely actions in the final
state. Reflect on the sequence of events and decisions made by each agent. Based
on analyzing the changes between the initial and current states, and the final state,
you must answer the following question about the final state:

Question: [inference question here]

Answer Options:
Provide an integer between 0 - 100 (where 0 = definitely target agent and 100 =
definitely other agent)

Strictly follow this response format:

Reasoning: [detailed 'Let's think step-by-step...' reasoning]
Answer: [answer as an integer between 0 and 100 here]
```

Figure G.1: Prompt template (simplified) for generating completions with `GPT-4`.

H Additional Results of Open-Source LLMs

We present additional results evaluating top state-of-the-art open-source LLMs (Llama-3.1-8B-Instruct and Qwen2-7B-Instruct) on our benchmark. We choose these models due to their large context length, as our prompt is over 11,000 tokens.

Both LLMs struggle to perform the inference task. Llama-3.1's performance is lower than but consistent with GPT-4's. For scenarios where GPT-4 does converge, Llama-3.1 does not necessarily converge, but it shows an increase in inference accuracy as the trajectory progresses, indicating some signal. For scenarios where GPT-4 does not converge ("Who turned on the shower" and "Who turned on the laundry"), Llama-3.1's inference accuracy does not improve with later evidence. We find that Llama-3.1 often reasons correctly about the state changes between timesteps, but it does not arrive at the correct conclusion. Meanwhile, Qwen2's inference accuracy does not increase as the trajectory progresses and struggles to reason accurately about the state changes.

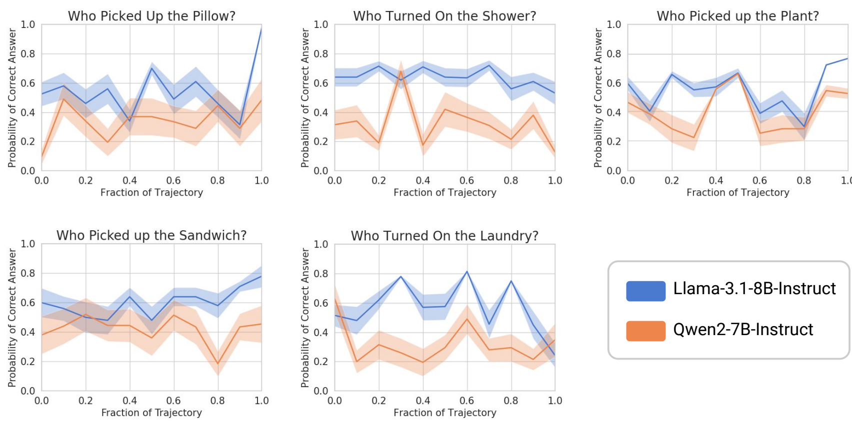

Figure H.1: Performance of state-of-the-art open-source LLMs on all 5 inference scenarios.

I Additional Results of GPT-4 with In-Context Learning

We conduct additional experiments using GPT-4 with in-context learning (ICL). We evaluate on the two scenarios where GPT-4 failed to converge with zero-shot prompting: "Who turned on the shower" and "Who turned on the laundry."

As shown in Figure I.1, GPT-4's performance improves with ICL — it fluctuates less and ends with a higher accuracy than the zero-shot baseline. However, it still fails to converge. In Appendix J, we provide examples of GPT-4 step-by-step reasoning to analyze this failure mode.

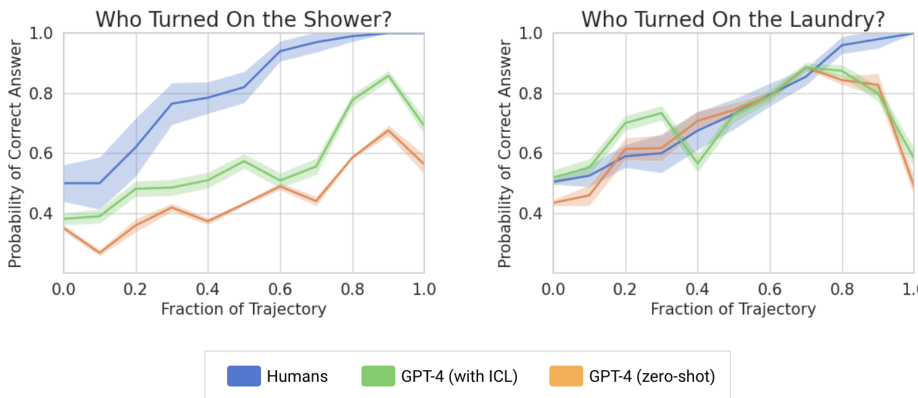

Figure I.1: Performance of GPT-4 baselines on two selected inference scenarios with the largest gap between human and GPT-4 performance.

J Analysis of GPT-4 Reasoning

We provide examples of GPT-4's incorrect reasoning in the "Who turned on the laundry?" scenario, where Agent A is performing `do laundry` and Agent B is performing `change outfit`. The correct answer is Agent A. In the input state for these examples, Agent A is stationary and turning on the laundry while Agent B is moving and carrying clothes. The key evidence is reflected in the change in environment state for Agent A: the laundry's `ToggledOn` state changes from `False` to `True`.

With both zero-shot prompting and in-context learning, we notice that GPT-4 tends to rely on agent state information and less on environment state information. This causes GPT-4 to make the wrong inference in this case.

Sample Reasoning from GPT-4 (zero-shot)

```
Let's think step-by-step and analyze the changes between the initial and current
states for each agent.  For the Agent A, the initial and current states are
identical.  The agent's position, direction, and carried items have not changed.
However, we can notice a change in the state of the laundry in the Bathroom.
Initially, the laundry was not toggleable, but in the current state, it became
toggleable.  However, there is no evidence that the Agent A has moved or interacted
with the environment.

On the other hand, the Agent B has shown some activity.  Initially, the Agent B was
carrying a piece of clothes, and the agent's position was [9, 5].  In the current
state, the Agent B has moved one step north to [9, 6], but the agent is still
carrying the same piece of clothes.  There are no changes in the state of the laundry
in the Bathroom for the Agent B.

In the final state, the laundry in the Bathroom is toggleable, which is the same as
in the current state of the Agent A. However, there is no clear indication that the
Agent A has moved or interacted with the laundry.  On the other hand, the Agent B
has shown activity by moving, but there is no direct evidence that the Agent B has
interacted with the laundry.

Given these observations, it's difficult to definitively say which agent is more
likely to have turned on the laundry.  However, since the Agent A's environment shows
a change in the state of the laundry and the Agent B has shown activity (despite no
direct interaction with the laundry), we can lean towards the Agent B.
```

Sample Reasoning from GPT-4 (in-context learning)

```
Reasoning:  Agent A started at position [13, 5] and remained at the same position in
the current state.  The direction of Agent A was 3 in the initial state and remained
3 in the current state.  This indicates that Agent A has not moved at all from the
initial state to the current state.

Agent B started at position [15, 9] and moved to position [4, 12] in the current
state.  The direction of Agent B was 1 in the initial state and changed to 3 in the
current state.  This indicates that Agent B has moved a considerable distance and
changed direction.

The final state shows an agent at position [13, 5] with direction 3.  This is exactly
the same position and direction as Agent A in both the initial and current states.
The final state also shows that the laundry is now turned on.  Given that Agent A
has not moved at all, it is highly unlikely that Agent A could have turned on the
laundry.  On the other hand, Agent B has shown movement and change in direction,
indicating that it is capable of performing actions.

Therefore, it is more likely that Agent B is the one who turned on the laundry.

Given that Agent A hasn't moved at all and Agent B has shown the capability of
interacting with objects (by picking up the 'clothes'), it's more likely that Agent
B is the one who moved the 'clothes' to the 'laundry' and turned it on, despite the
final position and direction matching with Agent A.
```

K Details on Human Experiments

We conduct experiments with 2 human experts. Each participant was provided with a habituation phase, in which they were familiarized with MARPLE domain knowledge, the inference setup, and a few examples of the agent trajectories beforehand. Each human participated in 50 inference trials which took around 3 hours.

For each trial, we show participants two agent trajectories, shown side-by-side with labels "Agent A" and "Agent B". They start from the initial step and move to the next timestep at their own pace, until they reach the end. This allows them to incrementally build an understanding of the agent trajectories and compare agent behaviors within the scenario. A diagrammatic illustration of the human study is shown in Figure K.1.

As they view the trajectories, we ask them to answer the inference question, e.g. "Which agent is more likely to have turned on the laundry?", at 11 evenly spaced timesteps, consistent with the mental-simulation and LLM baselines. The participants indicate their prediction using a scale from 0 to 100, with 0 being "definitely agent A" and 100 being "definitely agent B".

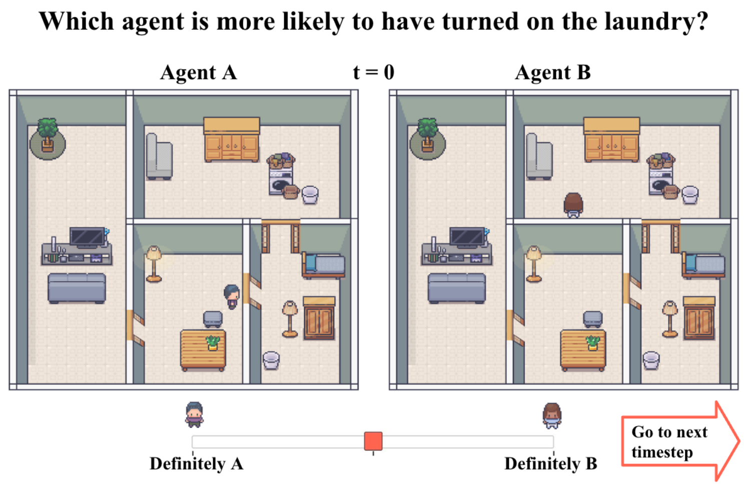

Figure K.1: Diagrammatic illustration of the human study for MARPLE. Participants saw Gridworld versions of the scenes. They started with initial scene, clicked the arrow sign to move to the next step, and then responded to the inference question by dragging the slider.