# DRAMA: Next-Gen Dynamic Orchestration for Resilient Multi-Agent Ecosystems in Flux

Supplementary Material

## A. Details of the Experiments

In this section, we explain some implementation details of our experiments.

**Dataset** As noted in part experiment setup, we use the Communicative Watch-And-Help(C-WAH) benchmark. In fact, we are using the test set in this benchmark, which contains a total of 50 household tasks and includes five types of tasks, named *Preparation*, *Cleaning*, *Cooking*, *Groceries*, and *Setting up* (Table 1).

Since the original task set was designed for two agents, in order to adapt to the situation of more agents, the original 3-5 subgoals were modified to 8-15 subgoals, which to some extent increased the complexity of the tasks.

Table 1. Task descriptions with relative objects

| Task Type | Relative Object |
|---|---|
| Preparation | cupcake, pudding, apple, juice, wine, coffeetable |
| Cleaning | plate, fork, dishwasher |
| Cooking | coffeepot, cupcake, pancake, poundcake, pudding, apple, juice, wine, dinnertable |
| Groceries | cupcake, pancake, poundcake, pudding, apple, juice, wine, fridge |
| Setting up | plate, fork, dinnertable |

**Baseline** Due to task set adaptation reasons, all baseline methods are based on CoELA.

- **CoELA**: Since CoELA itself is a framework designed for two agents, to support a greater number of agents, we use a modified version by Guo et al.
- **MCTS**: Use the original adapted MCTS in CoELA.
- **AgentVerse-Satic**: We designed a task allocator to assign roles and responsibilities to different agents.
- **AgentVerse-Dynamic**: Considering that there is an evaluator in AgentVerse that assesses the organizational structure after the task is completed, in static, since each task is independent, the evaluation would be meaningless. Therefore, we designed a more dynamic version and created an evaluator to evaluate the organizational structure when the sub-goals are completed.
- **ProAgent**: Transplant the intention prediction and belief revision involved in ProAgent.

**Dropout Processing** In DRAMA, we have designed a set of processes, including having the agent dropout, drop detection and drop handling. In terms of handling lost contact, we processed the relevant information of the disconnected agent. In other methods, the failure to process this information would lead to the failure of the task. In the experiment, we added this processing flow to all the compared methods.

## B. Detailed experimental data

To demonstrate the robustness and foundational effectiveness of our approach, this section presents results from a preliminary study conducted on the original C–WAH benchmark. This dataset represents a scenario with lower task complexity (i.e., fewer objects per task) compared to the extended environment in our main analysis.

The version of our method tested here, along with the baselines, utilizes foundational metrics: Average Steps (AS) for temporal efficiency and Total Steps (TS) for resource expenditure. While our main paper introduces a more advanced methodology and more fine-grained metrics (CR, TP) to analyze complex interactions, the results from this initial investigation (Table 2) were crucial. They established the core performance advantage of our agent coordination strategy, justifying the further development that led to the final version of DRAMA presented in the main paper. All trials were re-conducted in the event of a simulator-induced failure.

In the part of stableness of DRAMA, we choose the two tasks in the dataset. We ran in two different scenarios: dropout(5→4) and addition (3→5), each independently running 50 times to count AS (Average Steps), CR (Conflict Rate) and TP (Throughput). The detailed data is as follows (Table 3)

To validate the distinct contributions of our key architectural innovations, we conduct two critical ablation studies. We investigate the impact of: (1) removing the Collective Intelligence Layer, forcing agents to rely on blind search without semantic priors, and (2) disabling the Trust Chain mechanism, thus removing the system's explicit fault tolerance capability. All experiments are conducted on our most challenging dynamic scenarios, and the results are summarized in Table 4.

Table 2. A detailed comparison of all methods on the original C-WAH benchmark, featuring a reduced number of objects per task. Results are shown for both static scenarios (with 2, 3, and 4 agents) and dynamic scenarios involving agent dropout and addition.

| | | Static Scenarios | | | Dynamic Scenarios | |
|---|---|---|---|---|---|---|
| **Method** | **Metric** | **2 Agents** | **3 Agents** | **4 Agents** | **3 → 2** | **3 → 4** |
| *Comparison Methods* | | | | | | |
| CoELA | AS(Steps) | 70.32 | 52 | 49.58 | 60.58 | 54.47 |
| | TS(Steps) | 125.36 | 119 | 133.14 | 99.66 | 121.55 |
| MCTS | AS(Steps) | 87.4 | 68 | 60.67 | 81.34 | 73.78 |
| | TS(Steps) | 140.08 | 135.22 | 132.28 | 135.89 | 172.60 |
| ProAgent | AS(Steps) | 66.65 | 53.67 | 49.04 | 63.16 | 53.83 |
| | TS(Steps) | 109.78 | 114.73 | 123.52 | 104.80 | 121.54 |
| AV-Static | AS(Steps) | 68.12 | 52.96 | 51.6 | 65.04 | 53.05 |
| | TS(Steps) | 113.22 | 110.38 | 131.82 | 107.82 | 125.30 |
| AV-Dynamic | AS(Steps) | 68.54 | 57.48 | 51.5 | 67.90 | 61.91 |
| | TS(Steps) | 111.78 | 121.22 | 126.76 | 126.48 | 142.95 |
| *Our Method* | | | | | | |
| **DRAMA** | **AS(Steps)** | **67.8** | **49.4** | **46.34** | **58.74** | **51.26** |
| | **TS(Steps)** | **110.38** | **98.6** | **102.2** | **99.96** | **116** |

Table 3. Detailed statistical comparison of DRAMA against baselines under dynamic scenarios for Average Steps (AS), Conflict Rate (CR), and Throughput (TP).

| | AS | | | | CR | | | | TP | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Method** | **Mean** | **Var** | **Min** | **Max** | **Mean** | **Var** | **Min** | **Max** | **Mean** | **Var** | **Min** | **Max** |
| **Dynamic: 5 Agents to 3 Agents** | | | | | | | | | | | | |
| CoELA | 63.2 | 73.16 | 46 | 85 | 0.0718 | 0.0013 | 0.0152 | 0.1786 | 0.2417 | 0.0011 | 0.1765 | 0.3261 |
| ProAgent | 63.68 | 114.86 | 42 | 90 | 0.1245 | 0.0021 | 0.0435 | 0.25 | 0.2431 | 0.0021 | 0.1667 | 0.3571 |
| AV-Static | 60.82 | 94.63 | 43 | 87 | 0.0505 | 0.0017 | 0 | 0.1724 | 0.2529 | 0.0016 | 0.1724 | 0.3488 |
| AV-Dynamic | 63.2 | 115.84 | 45 | 91 | 0.05 | 0.0014 | 0 | 0.1852 | 0.2439 | 0.0015 | 0.1648 | 0.3333 |
| **DRAMA** | **56.44** | **67.45** | **39** | **76** | **0.0372** | **0.0008** | **0** | **0.0933** | **0.2715** | **0.0016** | **0.1974** | **0.3846** |
| **Dynamic: 3 Agents to 5 Agents** | | | | | | | | | | | | |
| CoELA | 56.5 | 60.33 | 37 | 72 | 0.104 | 0.0025 | 0.0164 | 0.2128 | 0.2709 | 0.0016 | 0.2083 | 0.4054 |
| ProAgent | 54.56 | 35.41 | 41 | 70 | 0.1088 | 0.0024 | 0 | 0.193 | 0.2783 | 0.001 | 0.2143 | 0.3659 |
| AV-Static | 54.86 | 48.64 | 45 | 87 | 0.0537 | 0.0019 | 0 | 0.2041 | 0.2779 | 0.0013 | 0.2113 | 0.3571 |
| AV-Dynamic | 60.44 | 78.17 | 42 | 71 | 0.0469 | 0.0013 | 0 | 0.1765 | 0.2535 | 0.0014 | 0.1724 | 0.3333 |
| **DRAMA** | **52.1** | **40.41** | **39** | **65** | **0.0457** | **0.001** | **0** | **0.1628** | **0.2923** | **0.0013** | **0.2308** | **0.3846** |

## C. Experiment Environment

This environment is running on Unity Simulator version linux_exec.v2.3.0, equipped with an NVIDIA 4090 GPU and a 32 vCPU Intel(R) Xeon(R) Platinum 8352V CPU at 2.10GHz. It supports CUDA 11.8 and uses Python 3.8.0.

Table 4. **Ablation study on the core components of our framework.** Performance of the full model versus variants lacking the Collective Intelligence Layer and the Trust Chain mechanism across all test scenarios.

| | | Static | | | Dropout | | Addition | | Recovery | Replacement |
|---|---|---|---|---|---|---|---|---|---|---|
| **Method** | **Matrix** | **3** | **4** | **5** | **4→3** | **5→4→3** | **4→5** | **3→4→5** | **4→3→4** | **4→3→4′** |
| Full Model (Ours) | AS | 59.977 | 50.242 | 46.697 | 57.205 | 52.796 | 48.213 | 53.367 | 55.609 | 55.429 |
| | CR | 0.027 | 0.062 | 0.083 | 0.026 | 0.076 | 0.068 | 0.057 | 0.044 | 0.061 |
| | TP | 0.189 | 0.217 | 0.252 | 0.2 | 0.216 | 0.235 | 0.211 | 0.204 | 0.205 |
| w/o Collective Intelligence Layer | AS | 65.432 | 56.674 | 48.043 | 61.021 | 57.659 | 53.918 | 54.918 | 57.767 | 60.78 |
| | CR | 0.037 | 0.054 | 0.119 | 0.029 | 0.079 | 0.073 | 0.064 | 0.062 | 0.059 |
| | TP | 0.171 | 0.205 | 0.242 | 0.188 | 0.211 | 0.211 | 0.208 | 0.198 | 0.186 |
| w/o Trust Chain mechanism | AS | 61.644 | 53.367 | 45.14 | 60.851 | 54.381 | 49.837 | 55.52 | 57.083 | 58.804 |
| | CR | 0.022 | 0.046 | 0.104 | 0.044 | 0.079 | 0.072 | 0.061 | 0.049 | 0.056 |
| | TP | 0.182 | 0.214 | 0.263 | 0.189 | 0.213 | 0.22 | 0.207 | 0.197 | 0.194 |

# D. Implementation Details of the Probabilistic Belief Network

To enable agents to reason about object locations under uncertainty, our framework includes a probabilistic belief network for each object, parameterized by a lightweight Multi-Layer Perceptron (MLP). This network's primary role is not to memorize locations, but to learn a **generalizable belief update function**. It maintains a categorical distribution over potential rooms for each object, updating its belief based on sparse evidence from `[grab]` actions. This section details its architecture, its efficient training regime, and its operational logic.

## D.1. Problem Formulation and Architecture

The core task is to maintain and update a belief distribution $\mathbf{p}(o) \in \Delta^{n-1}$ for each object $o$ over $n$ possible rooms, where $\Delta^{n-1}$ is the $(n-1)$-simplex.

- **Network Input:** The MLP takes a $2n$-dimensional feature vector as input, which is a concatenation of the prior belief and the current observation: $\mathbf{x} = [\mathbf{p}_{t-1}, \mathbf{o}_t]$.
  - $\mathbf{p}_{t-1} \in \mathbb{R}^n$: The belief distribution over rooms from the previous step.
  - $\mathbf{o}_t \in \mathbb{R}^n$: A one-hot encoding of the room where a `[grab]` action occurred.
- **MLP Architecture:** The network is a simple yet effective MLP with one hidden layer:

$$\text{Linear}(2n \to 64) \to \text{ReLU} \to \text{Linear}(64 \to n)$$

The output is a vector of $n$ logits. A Softmax function is applied during inference to produce the final probability distribution. A fixed, consistent ordering of rooms is used to ensure vector alignment.

## D.2. Few-Shot Online Training and Belief Update

A key feature of our design is its exceptional data efficiency. The network is trained online using a minimal dataset to learn a robust belief update policy.

- **Training Data:** The MLP is pre-trained on a small, representative dataset consisting of **only one trajectory from each of the five unique task types**. This "five-shot" training approach is designed to teach the model a general rule for how to update a belief given an observation, rather than overfitting to specific object locations.
- **Training Trigger:** After pre-training, the model continues to learn online. A single gradient step is performed each time an agent executes a `[grab]` action for a tracked object.
- **Objective:** The network is trained using a standard Cross-Entropy loss on the output logits, with the one-hot vector of the observed room $\mathbf{o}_t$ as the target label.
- **Inference and Update Logic:** Upon a new observation $\mathbf{o}_t$, the new belief $\mathbf{p}_t$ is computed via a stable blending process:
  1. **Prediction:** $\mathbf{p}_{\text{net}} = \text{Softmax}(\text{MLP}([\mathbf{p}_{t-1}, \mathbf{o}_t]))$.
  2. **Blending:** $\mathbf{p}_{\text{blend}} = (1-\eta) \cdot \mathbf{p}_{t-1} + \eta \cdot \mathbf{p}_{\text{net}}$, where $\eta = 0.9$.
  3. **Normalization & Write-back:** The blended vector is L1-normalized and persisted.

## D.3. Initialization, Persistence, and Hyperparameters

The system is designed for continuous learning and persistence across tasks.

- **Initialization:** The system loads the pre-trained MLP weights and a JSON file containing the last known belief distributions for all objects (priors). If a prior for an object does not exist, a uniform distribution is used.
- **Persistence:** At the end of each task, the updated MLP weights (saved as `mlp.pt`) and the latest object belief maps (saved as `global_prob_maps.json`) are written to disk.
- **Key Hyperparameters:** The system is configured with a hidden layer width of 64, a learning rate of $1 \times 10^{-3}$ for the Adam optimizer, and a blend factor of $\eta = 0.9$.

### D.4. Notes on Implementation

The implementation is highly efficient, with the belief update being an $O(n)$ operation and the MLP forward/backward pass being computationally trivial on a CPU. The primary limitation is a potential cold-start problem, though the pre-training on five tasks and the blending mechanism help mitigate this.

## E. Prompts Templates

We list the used prompts at Table 5.

## F. Case Study

As illustrated in Figure 1, the system demonstrates strong resilience to agent dropout events. During *Task_0*, all agents initially operate in coordinated fashion, with task affinities computed across Alex, Bob, Carl, and David. When the guardian mechanism detects that *David* has lost connection in the living room (*heartbeat stopped*), his primary guardian, *Carl*, is promptly notified. Through the *takeover module*, Carl temporarily assumes David's remaining responsibilities—including the delivery of `cupcake#2` and `cupcake#3`—to prevent task interruption. This reassignment is transparently recorded within the guardian chain, preserving operational continuity and accountability at the team level.

When any worker agent completes a task, a rescheduling procedure is automatically triggered. In this episode, Carl completes a cupcake delivery, prompting the control plane to recompute the affinity matrix based on the updated workload and spatial configuration. Consequently, the scheduler redistributes the remaining subtasks and reconstructs the guardian chains to represent the revised agent–task dependencies. Meanwhile, Alex continues moving toward the kitchen to collect puddings, and Bob proceeds toward the living room to execute his assigned delivery. This dynamic coordination among the control, executor, and guardian pathways enables the team to adapt effectively to unexpected failures, maintaining stable progress toward the shared objective even under partial agent loss.

## G. Limitations

Experiments are conducted in a simulated embodied environment, which may not fully capture the perception noise and actuator uncertainty of real-world systems. Extending DRAMA to physical multi-robot settings or mixed-reality platforms would validate its practicality under real-world constraints.

Table 5. Prompt of Actor.

**Actor Prompts**

"I'm $AGENT_NAME$. I'm in a hurry to finish the housework with my friends $TEAMMATE_NAME$ together. Given our shared goal, dialogue history, and my progress and previous actions, please help me choose the best available action to achieve the goal as soon as possible.
Hard requirement: - The ""action"" you output MUST be exactly one of the lines listed under ""Available actions"" below. Copy it verbatim (including brackets, object ids, and punctuation). Do NOT invent actions. If no listed action is appropriate, use ""None"".
Constraints and policy:
- I can hold up to 2 objects.
- Priority (strict):
1) If a SubGoal object is visible and reachable, Take/Grab it.
2) Else if any Goal object is visible and reachable, Take/Grab it.
3) Else Navigate/Explore to the room with the highest SubGoal prior.
- If target is inside a container, Open then Take.
- If multiple targets are reachable, pick up up to 2.
- If there are other agents in the same room with you, Take
Grab the subgoal object first to avoid conflict with other agents.
- Do NOT choose an action that targets the same object id or container id as any teammate in Previous actions (most recent step). Pick a different target/action instead.
- If my agent number is larger than a teammate in the same room (e.g., Agent_2 when Agent_1 is present), assume earlier-index agents may choose the same target; proactively pick a different object/container or a different room to avoid taking the same action.
- Tie-break: fewer steps; prefer immediate Take/Grab.
Organization instructions: $ORGANIZATION_INSTRUCTIONS$
Goal: $GOAL$
SubGoal: $SUBGOAL$
Progress: $PROGRESS$
Previous actions: $ACTION_HISTORY$
Available actions: $AVAILABLE_ACTIONS$
Response format:
{""thoughts"" : ""thoughts content"",
""action"" : ""choose one action from the available actions above or None""}
Validation:
- The ""action"" must be identical to one of the strings under ""Available actions"" (or ""None"" if nothing applies). Do not output any other string."

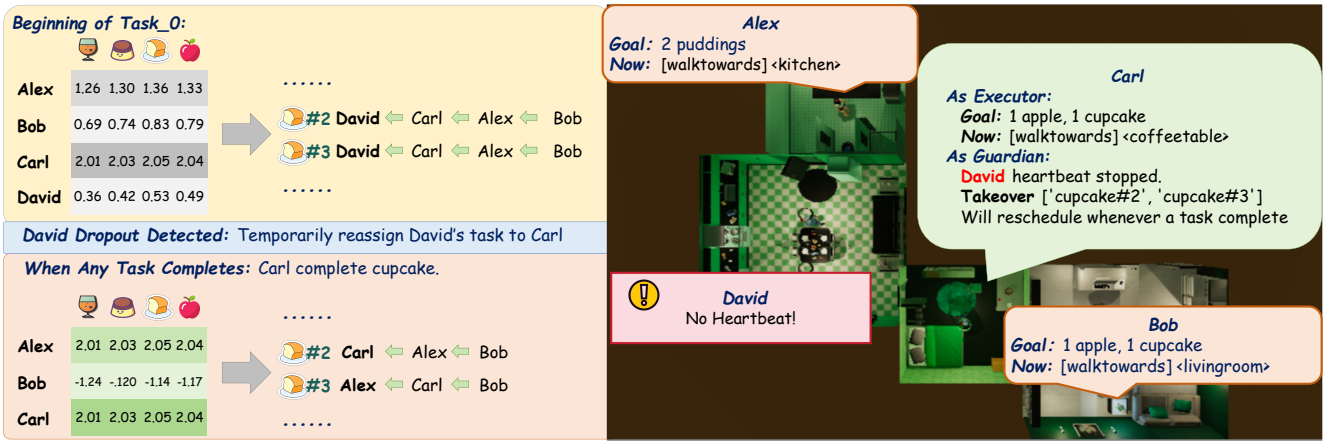


Figure 1. An example scenario demonstrating the dynamic adaptation of DRAMA after Bob disconnects.