CUBE: Collaborative Multi-Agent Block-Pushing Environment for Collective Planning with LLM Agents

Hanqing Yang[*]
Carnegie Mellon University
hanqing3@andrew.cmu.edu

Narjes Nourzad[*†]
University of Southern California
nourzad@usc.edu

Shiyu Chen
Carnegie Mellon University
shiyuc@andrew.cmu.edu

Carlee Joe-Wong
Carnegie Mellon University
cjoewong@andrew.cmu.edu

Abstract

We introduce CUBE (Collaborative Multi-Agent Block-Pushing Environment), a lightweight yet expressive testbed for studying embodied cooperation in multi-agent systems. While traditional agent cooperation benchmarks designed for reinforcement learning emphasize low-level action spaces and scalar rewards, and symbolic planning domains emphasize logical reasoning under deterministic transitions, neither approach alone captures the combination of embodiment, uncertainty, and symbolic structure needed to evaluate emerging embodied LLM-based agents. CUBE addresses this gap by wrapping primitive block-pushing actions into a symbolic action vocabulary, enabling interpretable and compositional cooperation strategies. It also provides a library of symbolic concepts for customized feedback at both per-agent and collective levels. These features allow the same environment to support reinforcement learning and LLM-based agents, as well as hybrid architectures. For ease and fair comparison across experiments, a single parameter n specifies the number of agents, grid size, and block weights, creating a transparent curriculum that scales difficulty and cooperation demands. CUBE thus offers a flexible platform for scalable evaluation of algorithms that integrate symbolic reasoning with embodied multi-agent interaction. The project is open-sourced at: `https://happyeureka.github.io/cube`.

1 Introduction

As large language models (LLMs) take on a growing role as planners and decision-makers, the environments used to study them must adapt to evaluate these capabilities. Traditional benchmarks, built with reinforcement learning agents in mind [Chevalier-Boisvert et al., 2018, Juliani et al., 2020, Samvelyan et al., 2021], emphasize low-level action spaces and scalar rewards. These signals are effective for gradient-based training but provide little support for symbolic reasoning, interpretability, or debugging. For LLM agents, producing long strings of primitive moves and waiting for numerical rewards is both unnatural and inefficient. Recent work has highlighted that large language models struggle to plan reliably without structured support [Valmeekam et al., 2022], though methods such as policy sketches [Andreas et al., 2017] and program-induction prompting [Singh et al., 2022] point to ways of bridging symbolic and learned reasoning.

[*]Equal contribution.
[†]Work done during an internship at Carnegie Mellon University.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Scaling Environments for Agents (SEA).

Symbolic planning domains define actions through explicit preconditions and effects, making them clear and interpretable [Ghallab et al., 2004]. However, they typically assume deterministic transitions, simplifying away many uncertainties that characterize real environments [Boddy, 2003, Bai et al., 2024]. In addition, they rarely capture embodied dynamics such as collisions, congestion, or force accumulation, meaning agents can hardly influence one another's states through physical interaction. As a result, while symbolic domains provide strong abstractions for logical reasoning, they remain limited in representing the interactive constraints of multi-agent physical tasks.

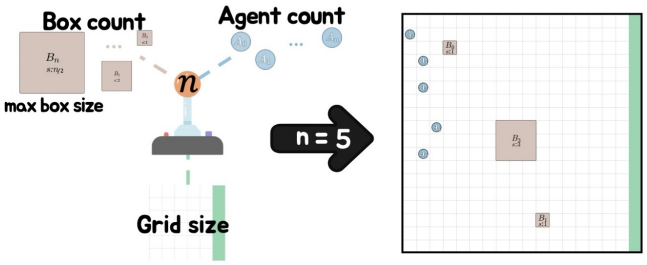


Figure 1: Illustration of CUBE's scaling mechanism. A single parameter n jointly determines the number of agents, the number and weights of blocks, and the grid size. Increasing n expands the agent population, block spectrum, and grid dimensions. Right: an example instance with $n = 5$ shows five agents and a set of blocks with varying weights placed on the grid, with the goal zone in green, where blocks must be delivered through coordinated pushes.

Human reasoning, in contrast, blends both symbolic and embodied perspectives. We rarely operate purely at the level of raw motor actions or purely at the level of logical abstractions. For embodied LLM agents, this same integration is essential. Just as people act and learn from their environment – forming symbolic prototypes from past experience, constructing mental models of a plan and its expected outcomes, and then relying on feedback from acting in the physical world to refine those prototypes [Johnson-Laird, 1983, Barsalou, 1999] – embodied LLM agents can likewise leverage such dual-layer representations to perceive, plan, anticipate, detect deviations, and adjust. In the language of perceptual control theory by Powers [1973], behavior can be understood as a continuous feedback process, where agents act to minimize the discrepancy between expected and perceived outcomes. This iterative loop of symbolic reasoning and embodied correction, forming, testing, and updating hypotheses, is central to how humans learn to coordinate and adapt. Thus, benchmarks for embodied LLM agents should combine symbolic structure with low-level, uncertain dynamics, which is precisely the goal of CUBE.

In addition to bridging symbolic structure with low-level embodied actions, CUBE aims to enable the evaluation of agent cooperation, which is critical to solving many embodied tasks in the real world [Durante et al., 2024, Guo et al., 2024, Chen et al., 2025]. While larger teams promise greater collective capability, they also introduce amplified challenges – communication of intent, spatial-temporal coordination, and misalignment between interpretation, goals, and actions – leading to frequent failures [Cemri et al., 2025]. Despite progress, existing frameworks remain limited to small groups of agents, and even those claiming scalability are typically evaluated on restricted team sizes due to the absence of standardized environments for systematic large-scale testing [Zhang et al., Yang et al., Nourzad et al., 2025]. CUBE addresses this gap by providing a scalable, physically grounded, and symbolically represented environment where collaboration is both necessary and measurable, enabling evaluation of cooperative strategies across diverse team sizes and task complexities.

CUBE (Collaborative Multi-Agent Block-Pushing Environment, Fig. 1) is a lightweight yet expressive testbed designed with the above-mentioned needs in mind. Inspired by prior cooperative block-pushing tasks [Juliani et al., 2020], CUBE extends this paradigm with symbolic interfaces and scalable cooperative settings. The environment integrates symbolic reasoning with the uncertainty and interaction of an embodied multi-agent world: agents must coordinate to push weighted blocks into a goal zone. Success depends on communication of intention, spatial-temporal alignment, joint action synchronization, making cooperation necessary. CUBE provides a dual-layer interface described in Section 2 that combines the strengths of the symbolic domain – compositionality, interpretability, and verifiability – with the raw vector representations used in traditional learning-based approaches, thus supporting diverse agent architectures and encouraging the development of novel methods that advance the next generation of cooperative intelligence.

Our work makes the following key contributions:

- We introduce CUBE, a lightweight, portable, and scalable multi-agent environment that supports variation in both the number of agents and the difficulty of tasks.

- We provide a reproducible difficulty curriculum specification for fair comparison, governed by a single interpretable parameter n, enabling systematic study of cooperation across consistent difficulty levels while allowing customization for targeted case studies.
- We design a dual interface that supports both symbolic and vector-based representations across observation, action, and feedback channels, enabling cooperative learning across reinforcement- and language-based agents as well as novel hybrid approaches.
- We perform baseline and scalability evaluations, demonstrating the computational efficiency of CUBE and revealing behaviors in both heuristics and LLM-based agents.

Section 2 introduces the environment design and dual-layer interface of CUBE. Section 3 examines embodied mechanics, emergent failure modes, and baseline evaluations. Section 4 concludes with a discussion of the broader challenges and research opportunities enabled by CUBE.

2 Environment Design

At its base level, CUBE is a grid-world environment built on PettingZoo's parallel API [Terry et al., 2021], where agents must cooperate to push square blocks into a designated goal region. The environment consists of agents and movable blocks placed on a grid. We denote the set of agents by $\mathcal{A}$ and the set of blocks available at the start of an episode by $\mathcal{B}$. At step t, $\mathcal{B}^{(t)} \subseteq \mathcal{B}$ denotes the subset of blocks that have not yet been delivered. Each block $B_j \in \mathcal{B}$ has an integer weight $w(B_j) \geq 1$, indicating the number of agents needed to push it. The block occupies a contiguous square of side length $w(B_j)$, so its physical size grows in direct proportion to its weight. This proportionality ensures consistency, as larger blocks both span more grid cells and require a greater quorum of agents to move. Each agent $A_i \in \mathcal{A}$ occupies a single grid cell, with position denoted x_i^t at step t. Episodes *terminate* successfully when all blocks have been delivered to the goal region. Episodes *truncate* if the `max_steps` is reached without delivering all blocks.

CUBE features a dual-layer interface that integrates symbolic reasoning with vector-based representations across **observation, action, and feedback channels** (see Sections 2.3–2.5 for details). The symbolic layer abstracts the dynamics of the environment into discrete entities and relations such as the distance between the agents and the quorum, allowing high-level reasoning, planning, and language interaction. Complementing this, the vector-based layer provides dense spatial and state features suitable for reinforcement learning and low-level control. Together, these layers allow agents to ground symbolic reasoning in embodied experience, supporting cooperative learning across reinforcement, language, and hybrid agent architectures.

2.1 Episode Initialization and Fair Benchmarking

CUBE is designed to reduce the burden of fine-grained environment engineering and to prevent unfair or inconsistent comparisons caused by unclear environment settings, ensuring fair benchmarking across algorithms. Each episode is automatically generated from a single governing variable n, which defines both the environment scale and cooperative difficulty. By default, the grid side length is $k = \max(20, n)$, with n agents placed on the board and a structured set of blocks whose weights range from $\lfloor n/2 \rfloor + 1$ down to one. Lighter blocks appear in greater numbers, creating a balanced mix of large, coordination-intensive tasks and smaller, easily solvable ones. The block distribution is chosen to ensure that episodes remain neither trivial nor overcrowded across scales, with roughly half of the grid covered by blocks. Agents are initialized along the wall opposite the goal region, and blocks are placed under constraints that prevent adjacency to walls or other blocks, since only pushing is allowed and agents would have no way to move them if a block were placed in a corner.

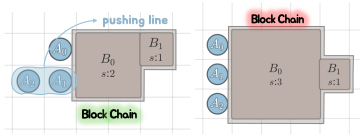
The parameter n simultaneously determines the number of agents, grid size, and block weights, creating a transparent and controllable progression of task difficulty. As n increases, heavier blocks require larger agent quorums, and more-block configurations lead to greater congestion and synchronization demands, since each agent contributes only one unit of force. Although individual layouts vary, tasks at the same n exhibit comparable cooperative complexity, enabling systematic evaluation of coordination and planning strategies from minimal to large-scale embodied settings that involve negotiation, allocation, synchronization, and sub-team organization. In addition, we allow user-specified scenario design to override the default configuration, enabling control over grid size, number of agents, block weights, and their distributions for targeted case studies.

2.2 Environment Dynamics

CUBE is a grid-world where agents and blocks move according to simple rules. Their dynamics create coordination challenges through collisions, congestion, and enforced collaboration.

2.2.1 Block Chain and Agent Chain

Block chain. A block $B_j \in \mathcal{B}$ of weight $w(B_j)$ may occupy one or more grid cells. When another block $B_k \in \mathcal{B}$ lies directly in front of B_j along a push direction $d \in \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$, the blocks form a *block chain*. The chain behaves as a composite structure whose motion depends on whether the total applied force at its leading face is sufficient to move all blocks within the chain. Let i index agents in the corresponding *agent chain*, each contributing unit force $f_i = 1$ along direction d. A chain advances one cell in direction d if and only if $\sum_i f_i \geq \sum_{B \in \text{chain}} w(B)$ and all destination cells are unoccupied and within bounds. Upon success, all blocks in the chain advance by one cell; otherwise, the entire chain of blocks remains in place.



(a) Successful chain (b) Failed chain

Figure 2: Illustration of chains: a chain succeeds or fails depending on whether available agents meet the block's weight requirement and the maximum force exerted on the block face.

Agent chain. Agents exert unit forces on blocks through pushing chains. A pushing chain forms when multiple agents align *collinearly* behind a block and all push in the direction of the block face. The effective force at the contact face is the number of aligned agents in that direction. If this total meets or exceeds the required block (or chain) weight, the structure advances. Otherwise, the attempt fails and all participating agents remain in place, as shown in Fig. 2.

2.2.2 Agent Movement and Collisions

While the environment supports a dual-layer action interface, primitive and symbolic (see Section 2.4 for details), each symbolic action is unrolled into a sequence of primitive actions, with all movements and collisions resolved at the primitive level. At each timestep, each agent $A_i \in \mathcal{A}$ issues a primitive action $u_i \in \mathcal{U}$, which specifies a target cell on the grid. A move is valid if the target cell lies within bounds and is not occupied by a block (including newly moved blocks). If multiple agents attempt to move into the same unoccupied cell, the agent with the smallest index i successfully claims the cell, while all others involved in the conflict remain unmoved. If any agent attempts to move into a cell occupied by a stationary agent, the move fails and both agents stay in place. This rule prevents overlap and introduces a consistent tie-breaking mechanism for simultaneous movements.

2.3 Observation Space

CUBE provides two observation modalities: a symbolic observation and a multi-channel observation. This dual interface supports diverse agent architectures, enabling reinforcement learning agents to rely on grid-based encodings, LLM-based agents to operate over symbolic state descriptions, or novel approaches that combine both.

Symbolic Observation. At each step t, every agent A_i receives a symbolic dictionary describing the current state. This includes global environment information (grid size, positions of all agents) as well as a compact summary for each block (block ID, weight, position, and distance to the goal column). The dictionary also records all symbolic actions taken so far in the episode, along with their corresponding primitive actions and the status (start, in progress, or end) at each timestep. This structured interface allows reasoning directly about concepts such as which blocks remain, how far they are from the goal, and where teammates are, supporting high-level planning and coordination.

Multi-channel Observation. In addition, a five-channel grid encodes agent locations, block weights, the goal column, a channel marking which agent occupies each cell, and a channel marking which block occupies each cell. This representation resembles standard reinforcement learning observations and is primarily included for compatibility with reinforcement learning pipelines and for visualization.

2.4 Action Space

CUBE supports two sets of action spaces. The primitive action space provides low-level grid movements, enabling agents to interact directly with the environment through discrete directional moves. The symbolic action space abstracts these primitives into higher-level cooperative strategies, such as aligning on a block face, synchronizing for a push, or waiting for teammates. Together, these two levels allow experiments to target both reinforcement learning agents, which operate naturally over primitive actions, and LLM-based agents, which benefit from reasoning over symbolic actions.

Primitive Actions. Each agent selects from a discrete 5-action set

$$\mathcal{U} = \{\texttt{STAY} = 0,\ \texttt{UP} = 1,\ \texttt{DOWN} = 2,\ \texttt{LEFT} = 3,\ \texttt{RIGHT} = 4\}.$$

At time t, each agent i issues an action u_i^t specifying a movement direction, and all agents act in parallel to move one unit in their respective directions. Moves succeed only if the target cell is free; collisions with walls, agents, or insufficiently supported blocks cause the agent to remain in place. A push succeeds if the aligned agents' combined force exceeds the total weight of the aligned blocks and the destination cell is free, in which case both the blocks and agents advance one step.

Symbolic Actions. Beyond primitive grid movements, CUBE provides a library of *symbolic actions* that capture higher-level coordination patterns such as aligning on a block face, synchronizing for a push, or waiting for teammates. Each symbolic action is grounded in the underlying transition dynamics, with execution compiling into the necessary sequence of primitive moves until the specified condition is met. For instance, `push_block` compiles into primitive moves where each aligned agent repeatedly issues the action directed into the block face. Similarly, a `move_to_block` unfolds into a path of primitive moves that positions the agent on the specified face of the block.

Action	Arguments	Effect
`move`	(*direction*, *steps*)	Move in the specified direction for a set number of steps.
`move_to_block`	(*block*, *face*)	Move toward a block and align on the given face.
`rendezvous`	(*block*, *face*, *count*, *timeout*)	If aligned with the block face, enter a **rendezvous state defined by the block and face**; wait until enough agents enter the same rendezvous state or timeout.
`push_block`	(*block*, *steps*)	Push the block for a set number of steps; the push direction is inferred (toward the block).
`yield_block`	(*block*, *steps*)	Move away from the block for a set number of steps; the face is inferred as the opposite of the direction toward the block.
`idle`	(*steps*)	Remain idle for a set number of steps.
`wait_agents`	(*count*, *timeout*)	Enter the **wait state** and remain idle until enough agents are also in the wait state or timeout.

Table 1: Symbolic actions in CUBE. Each action specifies a high-level effect that can be decomposed into primitive actions.

Although the symbolic action is compact, the inclusion of arguments such as block identifiers, sides, timeouts, and step counts allows each action to be instantiated in many different ways. This parameterization substantially expands the effective planning and action space, making it far richer and more expressive than it initially appears.

Plan. An agent can submit a plan to the environment for execution. A *plan* π_i for agent A_i is composed of a sequence of parameterized symbolic actions:

$$\pi_i = [a_i^1(\theta_i^1), a_i^2(\theta_i^2), \ldots, a_i^T(\theta_i^T)],$$

where each a_i^k is a symbolic action with arguments θ_i^k. Each symbolic action is executed sequentially and decomposed into primitive actions during execution. Thus, within this space, agents must determine how to align with a block, synchronize with teammates, wait for collaborators, or yield when obstructing others. For example, `move_to_block` may be followed by `rendezvous` to ensure that sufficient agents are placed on the same block face before executing `push_block`. Such combinations illustrate that symbolic actions are not only flexible but can also be sequenced logically to achieve more effective coordination.

Table 1 lists each symbolic action with its arguments, preconditions, and effects. The executing agent is implicit and denoted by A_i. The symbolic actions designed for synchronization, `rendezvous` and `wait_agent`, align agents for interaction and temporarily change their state from *normal*. `rendezvous` is valid only when an agent is aligned with a block face; it then enters a rendezvous state defined by that block face, waiting until enough agents enter the same rendezvous state. `wait_agent`

places the agent in a wait state until enough agents are also waiting. When the condition is met or timeout, the agent returns to `normal` and proceeds with its plan or generates a new plan. By exposing a shared *vocabulary of strategies*, CUBE enables LLM-based agents to reason, communicate, and adapt at the level of meaningful cooperative behaviors rather than isolated unit steps.

2.5 Feedback

CUBE provides both scalar and symbolic feedback channels to evaluate agent performance and task progress. This dual-layer design enables low-level reinforcement learning signals alongside interpretable, concept-based metrics for higher-level reasoning.

Rewards. We use r_s for the step cost (default 0.01) and r_d for the delivery reward (default 1.0). Delivered blocks are removed immediately, and if multiple blocks reach the goal in the same step, their rewards are summed. The per-step reward for agent A_i is given below, with $D^{(t)} = 0$ if no block is delivered. All agents share a global reward signal that reflects collective progress, following a similar environment in Juliani et al. [2020]: $r_i^{(t)} = -r_s + \frac{D^{(t)}}{|\mathcal{A}|}$, where $D^{(t)} = \sum_{B \in \text{delivered}} r_d \cdot w(B)$.

Symbolic Concepts for Customized Feedback. In addition to symbolic actions, CUBE exposes a library of *symbolic concepts* that capture the geometric and relational properties of the environment. These concepts do not prescribe feedback directly; instead, they provide a flexible vocabulary that researchers can combine to design adaptive feedback signals for task progress and cooperation quality.

Table 2 summarizes the available concepts implemented in the environment. They include utilities for querying blocks, computing distances, reasoning about alignment, inferring push directions, and pathfinding. Together, these concepts make it possible to define higher-level evaluative criteria without modifying the environment's core mechanics.

Concept	Arguments	Return
`get_distance`	(*entity*$_1$, *entity*$_2$)	Manhattan distance between two entities (agent, block, or position).
`is_aligned_with_block`	(*block*, *side*)	Boolean indicating whether an agent is aligned with the given block face.
`count_aligned_agents`	(*block*, *side*)	Number of agents aligned with a given block face.
`all_aligned_positions`	(*block*, *side*)	Set of all valid alignment positions for a block face.
`block_progress`	(*block*)	Distance of the block to the goal.
`delivered`	(*block*)	Boolean indicating whether the block has been delivered.
`quorum_status`	(*block*, *side*)	Boolean indicating whether aligned agents meet or exceed block weight.
`quorum_deficit`	(*block*, *side*)	Number of additional agents required to push the block.
`blocked`	(*block*, *side*)	Boolean indicating whether a block face is blocked by a wall, block, or agent.

Table 2: Symbolic concepts available in CUBE. Each concept is a function returning a property or relation, providing reusable primitives for defining customized feedback or evaluation signals.

3 Evaluation

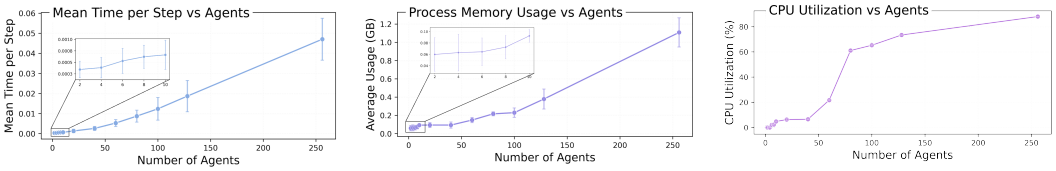
In our evaluation, we empirically demonstrate the scalability of CUBE, showing that the environment sustains high performance with negligible computational overhead even when scaled to hundreds of agents. We further evaluate the integration of different classes of agents, illustrating how LLM-based agents can operate effectively within the environment. LLM inference dominates the total runtime, with inference time $\gg$ CUBE's time per step (Table 3, Figure 3a), confirming that the overhead of CUBE is negligible. The environment is implemented in native Python, accelerated with Numba for efficiency, and executes entirely on a single CPU core without requiring GPU resources. It is designed to be lightweight, portable, and easily customizable, and operates independently of the broader LLM infrastructure.

Table 3: Average LLM inference time on Azure over 10 runs for generating a 3-step plan.

Model	Time (s)
gpt-4o	0.7 ± 0.09
gpt-4o-mini	1.6 ± 0.07

3.1 Scalability and Computational Footprint

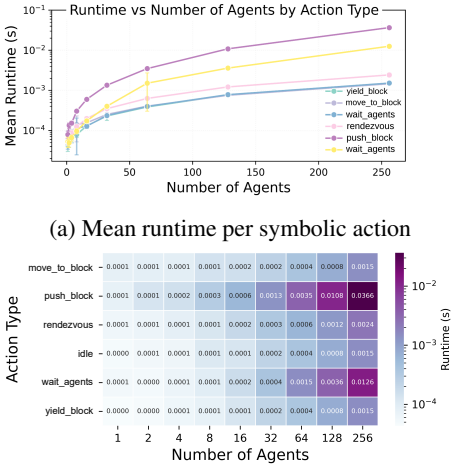
We evaluate CUBE by examining how well different classes of agents leverage its dual interfaces to solve cooperative block-pushing tasks, highlighting CUBE's scalability with respect to n.



(a) Mean Time per Step (b) Memory Usage (c) CPU Utilization

Figure 3: Scalability analysis of primitive actions with respect to the number of agents and blocks (n). (a) Mean time per step grows smoothly with n, remaining negligible at small scales and rising gradually as agent count increases. (b) Memory usage follows an approximately linear trend, reaching under 1 GB at $n = 256$, with overhead in the tens of MB for small n. (c) CPU utilization increases steadily with n and remains below 90% with 256 agents operating simultaneously.

Scalability of Primitive Actions. To evaluate the computational footprint of our environment, we measured runtime and system resource usage[1] as a function of the number of agents (n), using randomly acting agents to isolate the overhead of the environment from the complexity of the policy. Each setting was run five times to account for variability (see Figure 3). Mean time per step grows roughly linearly with the number of agents, remaining sub-millisecond for small teams and reaching roughly 0.05s at 256 agents. Process memory usage also increases approximately linearly with n, from tens of MB at small n to 0.9 GB at 256 agents. CPU utilization rises gradually at first, then steepens around mid-scale team sizes, approaching $\sim 90\%$ by 256 agents. Overall, the environment stays lightweight for small and moderate teams and scales to a few hundred agents on a single-core processor; the main pressure at high n is CPU, while memory remains below 1 GB.



(a) Mean runtime per symbolic action

(b) Runtime breakdown per symbolic action

Figure 4: Scalability of symbolic actions as the number of agents increases. Lighter actions (e.g., `idle`) remain inexpensive across scales, while heavier actions that require inspecting environment objects (e.g., `push_block`, which infers direction from a block reference and checks agent alignment) incur higher computational cost at larger n.

Scalability of Symbolic Actions. We profile per-action runtime as a function of the number of scripted agents, with all agents hard-coded to follow the same symbolic action sequence. These measurements capture the additional overhead of symbolic mediation, which is checking preconditions and decomposing into primitive steps, relative to directly executing primitive actions. We note that some symbolic actions are easier to optimize for performance than others. For example, `move_to_block` can be efficiently accelerated with Numba, while actions more tightly coupled to Python class logic are less amenable to such optimization. Both the line plot and the heatmap in Figure 4 exhibit the same trend[2]. Most symbolic actions remain inexpensive across scales, around 10^{-4} to 10^{-3} seconds even at $n = 256$. In contrast, symbolic actions that require accessing Python objects to perform multi-agent checks, such as `rendezvous`, `wait_agents`, and `push_block`, grow progressively more expensive. Nevertheless, even the most expensive actions remain lightweight, with per-action overhead under 0.05 seconds.

3.2 Failure: Interaction, Congestion, and Geometry

Although the underlying rules of CUBE are simple, their interaction yields complex cooperative dynamics. Agents must not only reason symbolically about goals and block relations but also contend with the effects of embodied actions, making an apparently simple environment fundamentally nontrivial in practice.

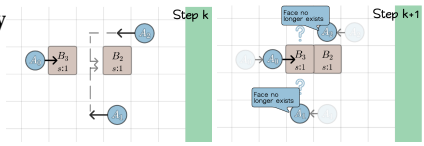


Figure 5: Failure due to agent and environment dynamics. Spatial congestion blocks the targeted face, halting progress.

[1]Experiments were run on Apple M2 (8-core CPU, 10-core GPU, 16GB unified memory).
[2]Experiments were run on Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20 GHz.

In CUBE, embodied movement introduces uncertainty during execution, as a previously feasible plan could become infeasible over time. Figure 5 illustrates this effect. At step k, the left face of block B_2 is available, and agents A_1 and A_2 plan to approach it while A_0 pushes block B_3 rightward. When B_3 moves adjacent to B_2 at step $k{+}1$, the left face of B_2 no longer exists as a valid approach direction. The agents that had committed to this face now encounter an invalid target, creating ambiguity in their subsequent actions. This example highlights how valid tasks in CUBE can disappear dynamically as a result of embodied motion and spatial reconfiguration, requiring agents to adapt their strategies during execution.

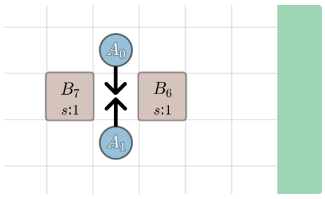


Figure 6: Failure caused by a cell access race during concurrent movement.

Moreover, a cell race occurs when agents A_0 and A_1 attempt to move into the same cell, as shown in Figure 6. When multiple agents target the same location, the one with the smaller ID claims the position, while the others remain stationary. This rule introduces unanticipated outcomes from each agent's local perspective, often invalidating their intended plans. Such racing behavior underscores the limitations of purely reactive or independent control policies and motivates the development of higher-level coordination mechanisms to prevent conflicts through shared expectations.

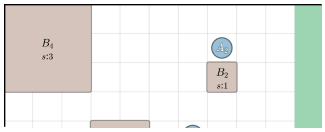


Figure 7: A boundary-aligned block (B_4) becomes unactionable under the push action.

Challenges further arise from action constraints when blocks are located at the grid boundary, since agents can only move a block by pushing, not pulling. In Figure 7, block B_4 lies along both the left and top edges of the environment, leaving no valid cell from which an agent can push it toward the goal zone. Adjacent cells that could serve for alignment or approach fall outside the reachable workspace. Consequently, the block becomes effectively isolated and no longer actionable under the push operation. This illustrates how boundary geometry alone can make certain tasks infeasible, even without coordination errors.

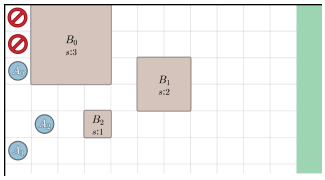


Figure 8: Failure caused by an agent occupying a feasible position adjacent to a block.

While moving, an agent may temporarily lock a task if it lacks spatial understanding and coordination. In Figure 8, all agents attempt to align on the left side of B_0. However, Agent 0 can temporarily occlude the remaining positions along that side during its movement. It needs to recognize that it is blocking the paths of other agents if it remains stationary. Agent 0 must move upward to create space for the other agents to gather. The agents must also understand the order of arrival, as this determines which cells they can claim behind a side of a block.

3.3 Agent Performance in the Environment

3.3.1 Baselines

Heuristic. We implement a heuristic baseline that follows a greedy strategy: at each step, the block closest to the goal is selected, and all agents are assigned to move it. Once a target block is chosen, each agent is given a step-by-step plan expressed as symbolic instructions such as `move_to_block`, `rendezvous`, and `push_block`. These instructions are repeated until the block is delivered, after which the process restarts for the next closest block. The baseline therefore produces valid cooperative behavior but does not attempt to optimize efficiency, parallelize work across blocks, or resolve congestion. It provides a straightforward point of comparison for more advanced approaches.

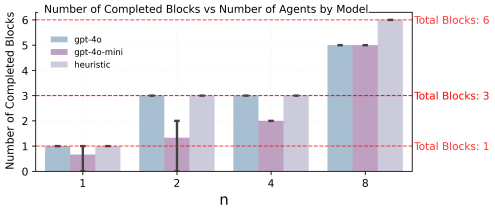


Figure 9: Number of completed blocks versus agent count n for `gpt-4o`, `gpt-4o-mini`, and the heuristic baseline. Red dashed lines indicate the total number of blocks present at each n (1, 3, 3, 6). The heuristic baseline consistently completes all available blocks. `gpt-4o` matches this except at $n = 6$. `gpt-4o-mini` not only underperforms but also exhibits high variance, indicating instability across runs.

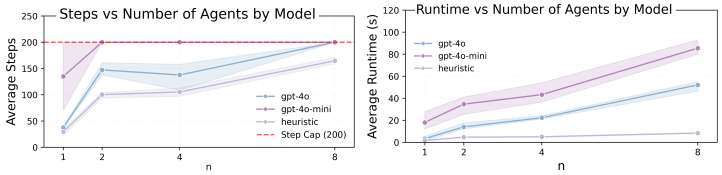
Naïve Language Agent. As a language-based baseline, we use OpenAI's `gpt-4o` and `gpt-4o-mini` models in a zero-shot setting. Each agent repeatedly generates short plans from the prompt based on

symbolic observation. The prompt encodes a simple strategy similar to the heuristic agent, always targeting the block closest to the goal zone, but expressed in natural language rather than in code.

3.3.2 Performance

Figures 9 and 10 present a comparison of our three baselines. The heuristic planner provides consistent cooperative behavior through a simple strategy. Naïve LLM agents (`gpt-4o` and `gpt-4o-mini`) show that pretrained language models can generate executable symbolic plans, but their performance is inconsistent and brittle, particularly when coordination across multiple agents is required. They succeed on simpler instances but degrade when plans involve other agents. Notably, `gpt-4o` generates longer and higher-quality plans than `gpt-4o-mini`, which is reflected in Fig. 9: `gpt-4o-mini` never completes an episode once $n \geq 2$ and also incurs longer runtimes, suggesting it replans more often.

This gap is further reflected in lower completion rates, longer execution traces, and higher variance relative to the heuristic baseline. Taken together, these baselines are not intended as competitive solutions but as reference points that highlight the potential of studying embodied cooperation in CUBE. While general-purpose LLMs can produce nontrivial symbolic behaviors, they fall short of robust cooperative performance on their own, particularly as n scales.



(a) Average steps to completion. (b) Average runtime.

Figure 10: **Baseline comparison across increasing agent counts (n).** The heuristic baseline consistently produces valid cooperative behavior, while naive LLM agents (`gpt-4o`, `gpt-4o-mini`) can generate symbolic plans but remain less reliable and less efficient, revealing the coordination gap that CUBE is designed to expose.

4 Challenges and Opportunities Enabled by CUBE

CUBE provides a scalable and reliable testbed for embodied cooperation that exposes the interplay between symbolic reasoning and embodied dynamics and reveals research challenges and opportunities for advancing multi-agent cooperative intelligence.

As the number of agents and blocks increases, coordination in CUBE becomes increasingly complex. Pushing a block requires both *spatial reasoning* to align precisely with a block face and *temporal coordination* to *synchronize* collective actions. The environment further introduces soft spatial *dependencies* among blocks, where a front block may obstruct others along a delivery path. The formation of block chains adds another layer of difficulty: while chains can enable more efficient collective motion, they also require agents to *reason jointly* about force distribution, timing, and spatial configuration. Moving a block not only changes the current layout but can also *alter task difficulty* by forming new chains or reducing space for subsequent deliveries. Agents must therefore manage *task decomposition, negotiation, and allocation*, deciding how to divide labor and sequence sub-goals efficiently while maintaining *adaptive strategies* that respond to an *ever-changing* environment.

Failures such as those discussed in Section 3.3 demonstrate that simple local decision rules can produce *emergent dependencies* that disrupt otherwise coherent plans. Effective cooperation, therefore, requires joint reasoning over space, time, and intention, as well as *predictive modeling of others' actions and their effects*. Agents must anticipate completion of teammates' subtasks, infer how their own motion influences team accessibility, and yield or reroute proactively to prevent blocking.

CUBE is a lightweight, extensible environment that unifies symbolic reasoning with embodied multi-agent interaction. Its dual-layer interface for observation, action, and feedback enables systematic study of hybrid reasoning, combining symbolic planning and reinforcement learning within a shared environment. A compositional feedback system allows adaptive reward design. A single scaling parameter n defines a transparent curriculum for examining how coordination strategies generalize with team size and environmental complexity. Together, CUBE serves as a foundation for advancing cooperative intelligence by linking symbolic abstraction with embodied execution, bridging reasoning and action toward scalable, generalizable multi-agent cooperation.

Acknowledgments

This work was supported in part by the National Science Foundation (NSF) under grants CNS-2533813 and CNS-2312761.

References

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *International conference on machine learning*, pages 166–175. PMLR, 2017.

David Bai, Ishika Singh, David Traum, and Jesse Thomason. Twostep: Multi-agent task planning using classical planners and large language models. *arXiv preprint arXiv:2403.17246*, 2024.

Lawrence W Barsalou. Perceptual symbol systems. *Behavioral and brain sciences*, 22(4):577–660, 1999.

Mark S Boddy. Imperfect match: Pddl 2.1 and real applications. *Journal of Artificial Intelligence Research*, 20: 133–137, 2003.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*, 2025.

Jingdi Chen, Hanqing Yang, Zongjun Liu, and Carlee Joe-Wong. The five ws of multi-agent communication: Who talks to whom, when, what, and why-a survey from marl to emergent language and llms. *Authorea Preprints*, 2025.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: A platform to study the sample efficiency of grounded language learning. *arXiv preprint arXiv:1810.08272*, 2018.

Zane Durante, Qiuyuan Huang, Naoki Wake, Ran Gong, Jae Sung Park, Bidipta Sarkar, Rohan Taori, Yusuke Noda, Demetri Terzopoulos, Yejin Choi, et al. Agent ai: Surveying the horizons of multimodal interaction. *arXiv preprint arXiv:2401.03568*, 2024.

Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. In *IJCAI*, 2024.

Philip N. Johnson-Laird. *Mental Models: Towards a Cognitive Science of Language, Inference and Consciousness*. Cambridge University Press, Cambridge, UK, 1983.

Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2020. URL `https://arxiv.org/pdf/1809.02627.pdf`.

Narjes Nourzad, Hanqing Yang, Shiyu Chen, and Carlee Joe-Wong. Dr. well: Dynamic reasoning and learning with symbolic world model for embodied llm-based multi-agent collaboration, 2025.

William Treval Powers. Behavior: The control of perception. 1973.

Mikayel Samvelyan, Robert Kirk, Vitaly Kurin, Jack Parker-Holder, Minqi Jiang, Eric Hambro, Fabio Petroni, Heinrich Küttler, Edward Grefenstette, and Tim Rocktäschel. Minihack the planet: A sandbox for open-ended reinforcement learning research. *arXiv preprint arXiv:2109.13202*, 2021.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*, 2022.

Jordan Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34:15032–15043, 2021.

Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Large language models still can't plan (a benchmark for llms on planning and reasoning about change). In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

Hanqing Yang, Jingdi Chen, Marie Siew, Tania Lorido Botran, and Carlee Joe-Wong. Llm-powered decentralized generative agents with adaptive hierarchical knowledge graph for cooperative planning. In *The First MARW: Multi-Agent AI in the Real World Workshop at AAAI 2025*.

Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B Tenenbaum, Tianmin Shu, and Chuang Gan. Building cooperative embodied agents modularly with large language models. In *The Twelfth International Conference on Learning Representations*.