

SyncMos: Scalable Motion Synchronisation for Multi-Agent Scene Interaction

Supplementary Material

This supplementary document provides additional details supporting our main paper: Section A provides further explanation of the planner’s internal reasoning process and implementation details. Section B presents the benchmark design, test cases, baselines, and evaluation metrics for the high-level planner. Section C reports extended multi-agent scalability results for Scene Office and further detail of evaluation metrics. Section D analyses the low-level motion refinement module and justifies the chosen partial denoising step.

A. Detail of High Level Planner

A.1. Top-View Grid Map Generation.

We generate the top-view grid map M directly from the scene metadata provided with each environment. An example of the top-view grid for the House scene is shown in Fig. 1. The scene configuration contains three components: (1) the global scene bounding box, (2) the grid parameters such as cell size and user-defined overrides, and (3) a list of all scene objects with their 3D locations, sizes, and oriented bounding boxes.

Given these files, we project all object bounding boxes onto the ground plane and rasterize them into grid occupancy. The grid cell size (typically 0.3 m) and the spatial extent of the map are derived from the scene bounding box. Each occupied cell is labeled using object footprints generated from the oriented 3D bounding boxes. Free-space regions are computed as all grid cells that do not intersect any object footprint. This produces a discrete 2D representation that encodes navigable space, furniture placement, and small object positions.

The resulting grid map is stored as a 2D image and a JSON file and is used as the spatial interface shared across user instructions and the Top-View Spatial Reasoner.

A.2. Dependency-Aware Story Planner

The Dependency-Aware Story Planner converts the scene description D and the user instruction T into a structured, machine-readable event dependency graph. It uses an LLM with a structured prompt that consists of: (i) a system-level prompt describing the role and objectives of the planner, (ii) a set of few-shot examples illustrating the desired event decomposition and dependency patterns, (iii) the scene description D generated by the Scene Understanding Module, (iv) the textual user instruction T , and (v) the set of available action labels A .

The LLM outputs a JSON object that encodes a struc-

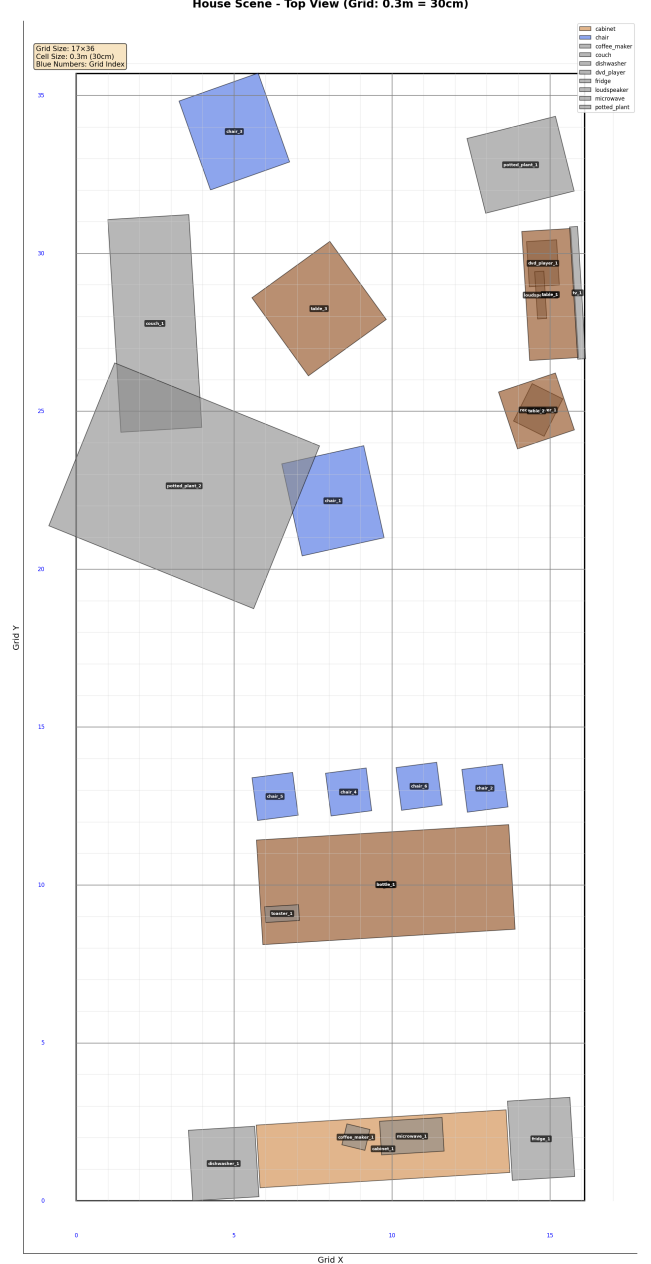


Figure 1. Top-view grid map for House generated from scene metadata. Object footprints are projected onto the ground plane and rasterized into grid occupancy, while free-space cells provide a navigable 2D coordinate system for grounding user instructions and spatial reasoning.

tured event dependency graph

$$G = (E, R), \quad E = \{e_i\}_{i=1}^N, \quad R = R_{\text{seq}} \cup R_{\text{par}},$$

where E denotes the set of *single-actor events*, and R encodes their temporal relationships. Each event

$$e_i = (\text{actor}_i, \text{event_description}_i)$$

specifies which character performs the action and what the action entails, for example, “*pick up bottle_1 using right hand*”. This explicit textual description includes the action label, the hand to use, and the interacted object, providing concrete cues for both spatial reasoning and downstream motion generation.

To handle interactions involving multiple characters, we encode explicit multi-actor decomposition rules in the prompt. Any multi-person activity mentioned in T is decomposed into strictly single-actor events. For instance, a handover is represented as two events, the giver’s “*hand over bottle_1 using right hand*” and the receiver’s “*receive bottle_1 using left hand*” which are linked by a parallel dependency in R_{par} . Similarly, conversational interactions are modeled as two parallel single-actor “*talk with*” events. Causal relations such as *pick up* before *hand over* are included in R_{seq} .

In practice, we parse the JSON output to extract the event set E and the dependency set R , together with their textual descriptions. These events are then passed to the Top-View Spatial Reasoner, which grounds each abstract event into a concrete location in the scene, and subsequently to the motion generation module.

A.3. Top-View Spatial Reasoner

This section provides additional details on the Top-View Spatial Reasoner, which grounds the abstract events produced by the Story Planner into concrete spatial targets in the 3D environment.

Goal and Inputs The Spatial Reasoner takes as input:

- the scene description D ,
- the complete event list E and dependencies R ,
- the set of available action labels A ,
- the top-view grid map M .

The top-view grid serves as a 2D discretized representation of the scene containing free space, furniture footprints, and object locations. It provides a unified coordinate system for grounding locomotion and interaction targets.

LLM-Based Spatial Grounding We use a LLM to convert each textual event into a spatially grounded representation. The prompt includes a role description, a small number of few-shot examples, and the structured inputs above. The model then predicts, for each event e_i , a tuple of the form:

$$g_i = (\text{grid}_i, \text{action}_i, \text{hand_target}_i, \text{hand_position}_i).$$

where grid_i is the 2D pelvis location on the grid map, hand_target_i optionally denotes the manipulated object, hand_position_i specifies a precise hand location (on a surface or in mid-air).

Spatial Constraints The Spatial Reasoner enforces a set of general constraints encoded directly in the prompt:

- **Free-space placement.** Characters must be placed only in navigable cells of M and never inside furniture.
- **Non-overlap.** Multiple characters may not share the same grid cell; interacting characters are placed on opposite sides of a shared point.
- **Reachability.** Pick-up and put-down actions require standing beside the supporting surface; precise hand positions are inferred near the object.
- **Continuity.** Events involving the same character are grounded in spatially consistent locations unless the instruction specifies a relocation.

These constraints allow the LLM to reason about plausible spatial configurations while leaving geometric consistency checking to the grid representation.

Multi-Actor Interactions For events involving multiple characters (e.g., handovers), the Spatial Reasoner uses simple but effective rules:

- A shared 3D handover point is chosen, often in mid-air between the interacting characters.
- The giver and receiver are placed at free-space cells on opposite sides of this point.
- When needed, object references in natural language (e.g., “hand over the bottle”) are mapped to available action labels (e.g., *put down* and *pick up*) to fit the motion vocabulary.

This ensures that synchronized events in R_{par} result in spatially compatible grounding.

Output Representation The final output is a JSON-like structure containing a grounded event g_i for each planner event e_i . These grounded targets are passed to the motion generation module, which uses the pelvis and hand coordinates to drive locomotion and interaction synthesis.

B. Further detail of Benchmark and Result for Planner

Test Scenes and Scene Descriptions We employ the three test scenes, House, Office and Restaurant from Event-Driven Storytelling [2]. They construct scene by placing the objects from the HSSD dataset [1]. Each scene contains two to three separate areas with distinct contextual meaning. And for the Scene Descriptions, since our method can perform human object interaction. We modified the prompt of scene describer to include all the objects in the scene.

Test cases To evaluate the effectiveness of the Dependency-Aware Story Planner, we conduct experiments on a custom benchmark of 30 narrative test instructions designed for multi-character story planning. The benchmark aims to assess the model’s ability to generate structured event graphs that capture both causal dependencies and parallel relations among multiple actors.

The benchmark is split into two equally sized subsets: (1) **Synchronization** (15 scenarios), which mainly require detecting and preserving parallel relations among 2–5 characters, and (2) **Dependency** (15 scenarios), which stress long-horizon temporal dependencies, branching, and convergence of multiple event chains. Each subset further contains 5 *easy*, 5 *medium*, and 5 *complex* instructions, and covers three scene categories (House, Office, Restaurant) with varying numbers of actors (2–5). In total, the benchmark contains 331 events and 315 dependencies, including 81 parallel and 234 sequential relations.

Each test case consists of a scene description D , a set of characters $C = \{c_1, c_2, \dots, c_N\}$ where N is the number of participants, a list of available actions $A = \{a_1, a_2, \dots, a_M\}$ where M is the number of possible actions, a narrative instruction T describing multi-character interactions, and a structured event dependency graph $G = (E, R)$, where E denotes the list of events and R represents the temporal relations among them, including both sequential and parallel dependencies. An example test case is illustrated in Fig. X, showing the complete input–output structure used for evaluation.

When evaluating a test case, we provide the scene description D , characters C , available actions A , and narrative instruction T to the Dependency-Aware Story Planner. The planner then generates the most appropriate event dependency graph $\hat{G} = (\hat{E}, \hat{R})$ by jointly reasoning over all inputs. To evaluate performance, we align the predicted events \hat{E} and relations \hat{R} with their ground-truth counterparts E and R , and compute the defined metrics accordingly.

Baseline We compare our planner against an event-driven storytelling baseline [2], which generates events sequentially (one by one) in an autoregressive manner without modeling global temporal structure. The baseline is also capable of producing events that involve multiple characters; we treat such cases as parallel events. It maintains an internal event history to track previously generated actions, where each event can have two states: ongoing or completed. When a new event is generated while a previous one is still ongoing, we interpret this as either a parallel event (if multiple characters are involved) or an independent event without explicit dependency. Conversely, if a later event is generated while its prerequisite event is still ongoing, because of a defined sequential dependency requiring

the former to complete first, we consider this a violation of dependency.

B.1. Metrics

We evaluate the quality of the generated event graphs using four metrics:

Event Coverage (EC). The fraction of ground-truth events in E that are correctly generated by the model. An event is considered covered if its actor and action type match the annotated ground truth.

Dependency Accuracy (DA). The proportion of ground-truth temporal relations in R that are correctly inferred. Both sequential (*after/before*) and parallel relations are evaluated.

Passed Scenarios (PS). The number of scenarios for which the model produces a fully valid event graph, i.e., all events are covered and all dependencies are correct.

Scenario Pass Rate (SPR). The percentage of successfully passed scenarios over all scenarios in the subset.

$$\text{SPR} = \frac{\text{PS}}{15} \times 100\%.$$

Each of the two benchmark subsets (*Synchronisation* and *Dependency*) contains 15 scenarios spanning three scene types and multiple difficulty levels, with a total of 331 events and 315 dependencies.

B.2. Ablation on Planner

Table 1 analyses the contribution of few-shot (FS) and chain-of-thought (CoT) prompting to the planner. With both components enabled, our method consistently achieves the strongest performance. On GPT-4o, dependency accuracy increases from **67.1 / 20.5%** (Synchronisation / Dependency) in the baseline to **86.3 / 96.9%**, while the scenario pass rate improves from **33.3 / 0.0%** to **53.3 / 80.0%**.

Removing either FS or CoT degrades performance, particularly on the *Dependency* subset that requires reasoning over multi-step causal relations. Without FS, dependency accuracy drops to **83.5%**, and without CoT it decreases to **90.0%**. Similar trends are observed for GPT-4o-mini, indicating that FS mainly stabilises the planning format, while CoT improves reasoning over long event dependencies. Together, these components enable the planner to better structure event ordering and enforce prerequisite constraints between actions.

C. Further detail of Multi-Agent Scalability Evaluation

C.1. Metrics

We report three metrics that jointly quantify temporal synchronisation and interaction quality.

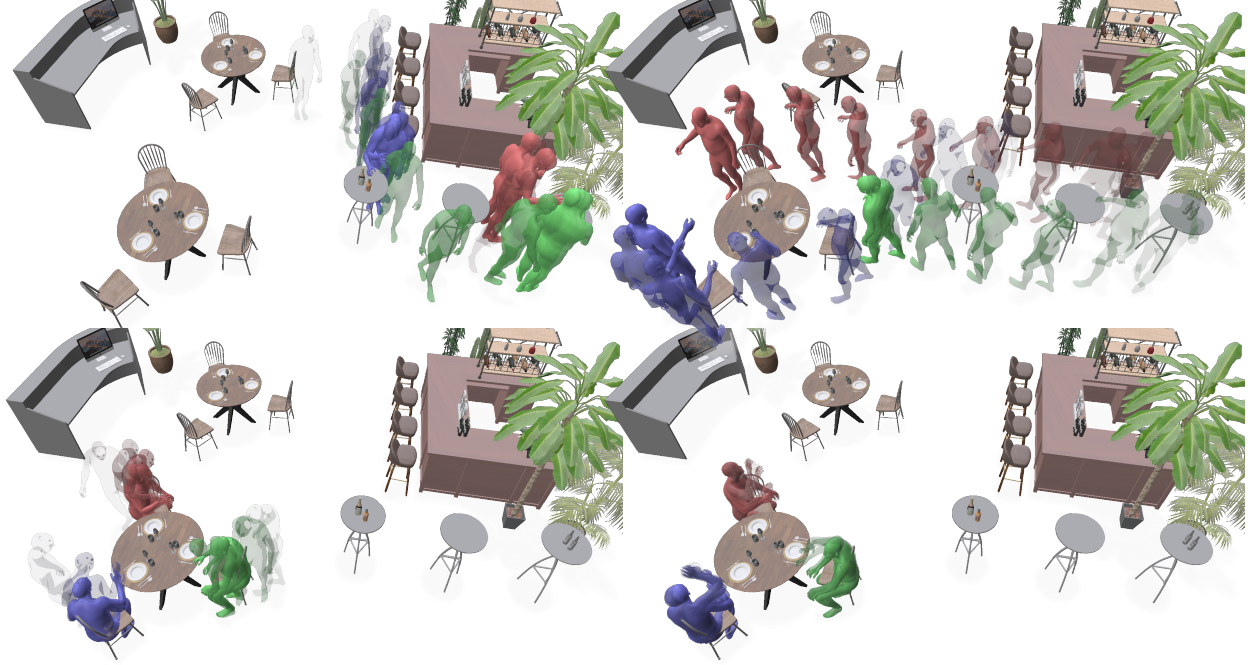


Figure 2. Left-to-right, top-to-bottom: the agents simultaneously pick up beers, walk to the table, sit, and perform synchronized cheer. Colors denote agents; transparency indicates time.

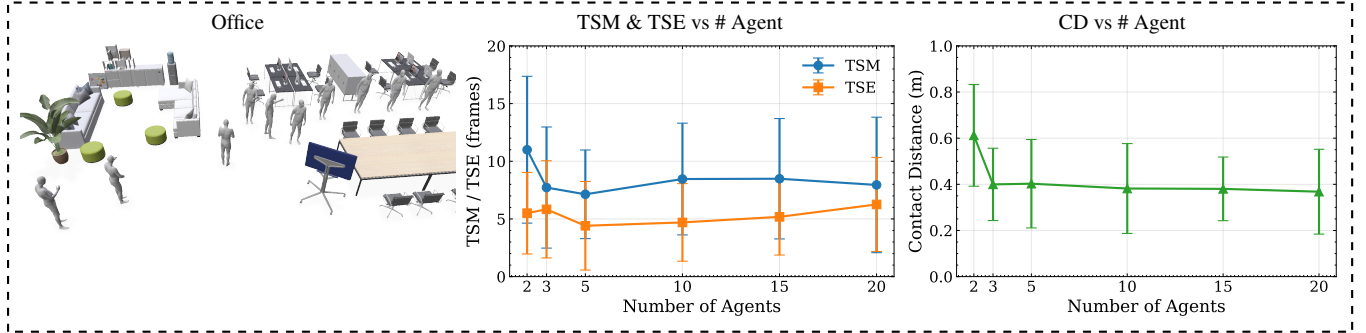


Figure 3. Multi-agent scalability for Scene Office. We plot TSM, TSE, and CD metrics as the number of agents increases, with each row showing results from one scene.

Table 1. Planner ablation on GPT-4o and GPT-4o mini w.r.t. FS and CoT. Dependency Accuracy (DA) and Scenario Pass Rate (SPR) are reported and denoted as Synchronisation/Dependency.

Method	GPT-4o		GPT-4o-mini	
	DA(%)	SPR(%)	DA(%)	SPR(%)
Baseline	67.1 / 20.5	33.3 / 0.0	71.4 / 16.2	33.3 / 0.0
Ours	86.3 / 96.9	53.3 / 80.0	67.2 / 80.4	33.3 / 46.7
w/o FS	77.9 / 83.5	46.7 / 60.0	67.5 / 56.2	33.3 / 26.7
w/o CoT	82.2 / 90.0	53.3 / 60.0	68.6 / 77.0	26.7 / 46.7

Temporal Synchronisation Magnitude (TSM). Let t_i^* denote the frame at which agent i achieves its best-contact moment before synchronisation. The temporal shift needed

to align all agents is

$$\text{TSM} = \frac{1}{N} \sum_{i=1}^N |t_i^* - \bar{t}^*|,$$

where \bar{t}^* is the mean best-contact frame across all N agents. For example, if two agents have best-contact frames at 10 and 20, then $\bar{t}^* = 15$ and the TSM is $\frac{|10-15|+|20-15|}{2} = 5$.

Temporal Synchronisation Error (TSE). For each event k requires time warping, let t_k^{plan} be the target frame for time warping, and let t_k^* be the original best-contact frame before time warping. After applying the time-warp module, this original frame is moved to a new position \hat{t}_k on the time-

line. TSE measures how far this warped frame is from the planner’s target:

$$\text{TSE} = \frac{1}{K} \sum_{k=1}^K |\tilde{t}_k - t_k^{\text{plan}}|,$$

where K is the number of interaction events. A lower TSE indicates that the warped contact frames are better aligned with the planned target frames.

Temporal Synchronisation Error (TSE). For each event k that requires time warping, let t_k^{plan} denote the target frame planned by the Story Planner, and let t_k^* be the original best-contact frame before time warping. After applying the time-warp module, this frame is moved to a new position \tilde{t}_k on the motion timeline. TSE measures the deviation between the warped frame and the planner’s target:

$$\text{TSE} = \frac{1}{K} \sum_{k=1}^K |\tilde{t}_k - t_k^{\text{plan}}|,$$

where K is the number of interaction events. A lower TSE indicates that synchronised contact frames more closely match the planned timing. Here, t refers to the frame index on the generated motion timeline.

Contact Distance (CD). For an interacting motion pair (i, j) , let $w_i(t)$ and $w_j(t)$ denote the wrist positions of agents i and j at frame t , and let $p(t)$ be the shared interaction target point (either a surface point or a mid-air handover point). The interaction precision is defined as

$$\text{CD} = \min_t (\|w_i(t) - p(t)\|_2 + \|w_j(t) - p(t)\|_2),$$

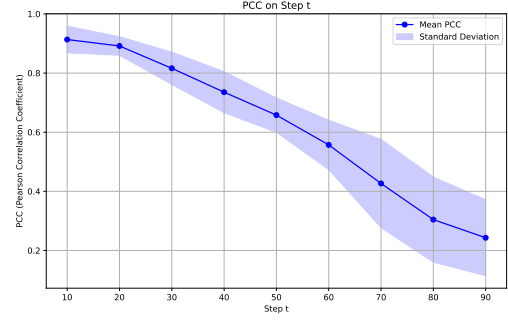
where t iterates over all frames around the interaction window. Lower values indicate higher-quality and more precise physical handovers.

C.2. Results for the Office Scene

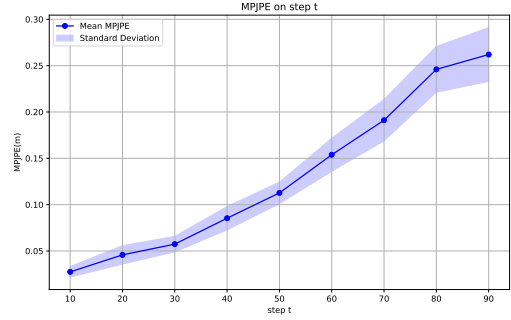
We also provide the detailed results for the *Office* scene, as shown in Fig. 3.

C.3. Results for additional actions

Besides the handover example, we also evaluate another scenario of cheering. The user instruction is “Amir, Benjamin, and Charles pick up the beer. They sit at the table together. They drink the beer together.” The qualitative result is shown in Figure 2. As illustrated in the figure, the three characters pick different beers, sit on different chairs around the same table, and cheer simultaneously. This example also involves dependencies and demonstrates that our planner can correctly resolve resource competition among multiple agents.



(a) PCC with step t



(b) MPJPE with step t

Figure 4. PCC and MPJPE comparisons with different step t.

D. Detail of Low Level Planner

D.1. selection for partial step

To determine the optimal partial sampling step t , we employed the Pearson Correlation Coefficient (PCC) and Mean Per Joint Position Error (MPJPE). These metrics assess the structural alignment (PCC) and positional fidelity (MPJPE) between the preliminary estimation and the final denoised output. As illustrated in Figure 4, both metrics improve as t decreases. We selected $t = 30$ as our partial sampling point, as it satisfies our performance criteria, ensuring a strong correlation ($PCC > 0.8$) and a sufficiently low reconstruction error ($MPJPE < 0.1$).

References

- [1] Mukul Khanna, Yongsan Mao, Hanxiao Jiang, Sanjay Haresh, Brennan Shacklett, Dhruv Batra, Alexander Clegg, Eric Undersander, Angel X Chang, and Manolis Savva. Habitat synthetic scenes dataset (hssd-200): An analysis of 3d scene scale and realism tradeoffs for objectgoal navigation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2024. 2
- [2] Donggeun Lim, Jinseok Bae, Inwoo Hwang, Seungmin Lee, Hwanhee Lee, and Young Min Kim. Event-driven storytelling with multiple lifelike humans in a 3d scene. In *Int. Conf. Comput. Vis.*, 2025. 2, 3