PRAXISCENE: BENCHMARKING AGENT SOCIAL AGENCY IN GENERATED OPEN-GOAL INTERACTIVE SCENARIOS

Anonymous authors
Paper under double-blind review

ABSTRACT

As AI agents increasingly operate in social environments, they often face situations with no predefined objective and incomplete decision-relevant information. Recent benchmarks have moved toward interactive social behavior, dynamic information, and complex social reasoning, but typically retain a given task, social goal, or inference target. It remains unclear whether social agents retain their performance when deciding what the situation requires and what evidence to seek becomes part of the interaction itself. We formulate this setting as *open-goal social agency* and introduce PRAXISCENE, a benchmark combining structured story generation, an interactive environment for action-driven branching narratives, a suite of open-goal social scenes, and trajectory-level evaluation. Its 1,248 frozen social situations comprise two story types: one exposes open branching interactions towards different story endings, while the other additionally contain intermediate states with hidden information that reshape subsequent choices. These stories unfold through grounded agent actions and environment responses, producing auditable interaction trajectories. PRAXISCENE evaluates story completion while characterizing how agents adapt their policies to the situation and available information, and what values are expressed along each trajectory. Matched interventions vary Goal, Information, and Value specification, with humans as a behavioral reference. Across six backbones and two agent systems, supplying Information sharply shortens interaction but reduces CR policy agreement by about 11 percentage points; supplying a Goal instead raises agreement by up to 9 points, while Value guidance selectively increases its targeted realization by 10.8 points.

1 INTRODUCTION

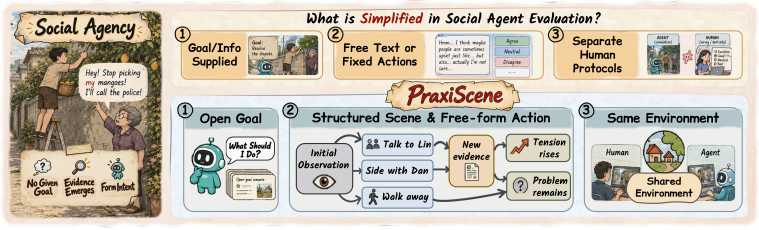

Figure 1: From simplified social evaluation to open-goal social agency.

As language agents move from answering questions toward acting on behalf of users, they increasingly operate in social environments whose demands are not fully specified in advance. A social situation concerns people and their relations with one another: who is involved, what each of them knows, and what each values. The agent acts on partial observations rather than direct access to the underlying state, and must determine what the situation calls for through interaction. We use ***social agency*** to denote the capacity to determine what a social situation calls for, acquire relevant information through interaction, and form, execute, and revise a course of action as the situation unfolds. The boundary dispute in Fig. 1 illustrates these coupled demands.

This motivates a different evaluation problem from asking whether an agent can solve a specified social task. We study this setting as ***open-goal social agency***, which is characterized by three properties. It is *open-goal*: no task-level outcome objective is supplied. It contains *hidden information*:

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

the evidence needed to act is not given up front but obtained through interaction. And its state is *history-dependent*: earlier actions shape subsequent observations and constrain which outcomes remain reachable. The central question is whether competence observed with supplied goals and relevant information persists when the agent must determine for itself what the situation calls for, what evidence to seek, and how to revise its behavior through interaction.

Existing benchmarks cover social commonsense and moral judgment (Sap et al., 2019; Hendrycks et al., 2021), theory of mind and contextual norms (Kim et al., 2023; Ziems et al., 2023), multi-turn social interaction (Zhou et al., 2024; Mou et al., 2025; Zhou et al., 2025), and dynamic social evidence (Xu et al., 2026). Yet most still supply the task, goal, inference target, or decision context. Leaving these decisions to the agent creates a different evaluation problem: trajectories must remain executable and comparable despite greater behavioral freedom, while controlled interventions must vary decision structure without changing the underlying situation. Because open social situations can support multiple defensible resolutions, evaluation must also separate criterion-identifiable capability from variation in the social policies agents choose to realize. Isolating the effect of shifting decision responsibility from evaluator to agent therefore remains difficult.

To study open-goal social agency, we introduce PRAXISCENE, an evaluation framework that combines configurable social-story generation, executable history-dependent interaction, and trajectory-level measurement. First, its structured generator constructs social situations from Scene and Value controls and converts them into executable environments in which free-form behavior is grounded to explicit events and state transitions. Second, we instantiate this framework as a benchmark of 1,248 social situations and evaluate both criterion-referenced capabilities and the social policies realized over complete interaction trajectories, with matched human behavior as a reference. Third, we use matched Goal$\times$Information interventions to isolate how evaluator-supplied decision structure changes the capability profile exposed by social-agent evaluation. Across six backbones and two agent systems, we find that successful termination does not imply a common mode of social agency: agents differ substantially in how they gather information and realize policy, while matched interventions show distinct effects of supplied Information, Goal, and Value on trajectory burden, terminal-policy agreement, and normative orientation.

2 RELATED WORK

From social reasoning to interactive social agents. Social-agent evaluation has progressed from social commonsense, ethical judgment, and theory-of-mind reasoning (Sap et al., 2019; Hendrycks et al., 2021; Emelin et al., 2021; Kim et al., 2023; Ziems et al., 2023) to multi-turn settings with evolving evidence, private information, and goal-directed interaction (Xu et al., 2026; Zhou et al., 2024; Mou et al., 2025; Pham et al., 2026). Yet these settings typically retain an evaluator-supplied task, goal, or inference target. PRAXISCENE instead studies what changes when the agent must determine both what the situation calls for and what evidence to seek through interaction.

Structured executable interaction. Interactive benchmarks trade behavioral freedom against execution control: SOTOPIA supports open-ended language interaction, while STSS and MACHIAVELLI use more explicit action or branching structures (Zhou et al., 2024; Wang et al., 2024a; Pan et al., 2023). Recent work also structures scenario construction through source-derived scenes, branching world trees, or simulation-grounded generation (Mou et al., 2025; Zhou et al., 2025; Chang et al., 2025). PRAXISCENE combines generative story construction with executable state transitions, grounding free-form actions into changes in information and reachable outcomes.

Human behavioral reference. Human data are increasingly used to contextualize social-agent behavior (Zhou et al., 2024; 2025). In PRAXISCENE, humans and agents traverse the same executable scenes under matched interaction semantics, providing a behavioral reference for trajectory and realized Strategy–Value profiles rather than a normative gold standard.

3 THE PRAXISCENE ENGINE

Evaluating open-goal social agency requires an interaction substrate that combines behavioral flexibility with auditable state transitions. We therefore develop the PRAXISCENE engine, shown in Fig. 2. It generates social situations with scene and value, and realizes them into structured story

specifications. Participants interact through free-form language, which the environment grounds into executable semantics to determine state changes and subsequent observations, while a unified logger records the resulting trajectory. This design supports the construction of new evaluation scenarios as needed without sacrificing the structured semantics required for trajectory-level analysis.

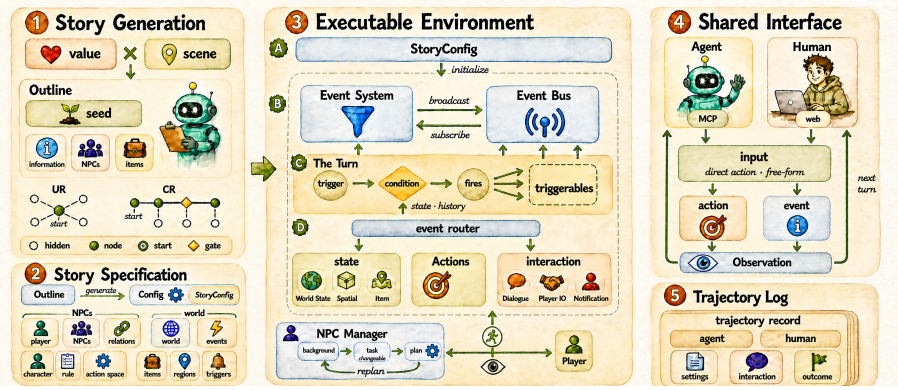

Figure 2: **PraxiScene engine architecture.** Structured social-situation specifications are generated from value and scene axes (1) and written into a `StoryConfig` (2). At execution time (3), an event system resolves triggers by evaluating conditions over both current state and event history, an event router dispatches events to the managers, and the state managers synchronize world state. An NPC manager plans behavior from its background under tasks that remains changeable at runtime. Agents and humans share one interface (4): each may issue a action or free-form input, the latter grounded to either an executable action or an event. Every turn is recorded by the environment in a unified trajectory record (5).

3.1 Story Generation and Specification

Generation controls. PraxiScene generates each situation from two authoring controls drawn from a fixed catalog: a *value* condition from the (Schwartz et al., 2012) value space and a social *scene* from the Bhatia et al. (2026) situation classes (Fig. 2, panels 1–2). Both are design choices rather than facts about the story: the value specifies what is at stake, while the scene specifies the social context where it arises. This design supports customization and helps keep generated situations comparable, while unconstrained prompting does not.

Story structure. Given a scene and value, the generator produces a semantic outline specifying the story's facts: opening situation, actors and their standing/knowledge, background hidden from the player and its disclosure condition, decision points and the transitions between them, and reachable endings. Generation is subject to structural constraints on branching, cast size, the opening state, and the set of reachable endings. The semantic outline is then parsed into a `StoryConfig`, which is the executable configuration that the environment reads. It explicitly names the characters, items, regions, and events the story needs and pairs each decision point with exactly one executable action. That pairing is what makes action grounding well-defined rather than interpretive.

Structural validation. A deterministic validator then checks the configuration for structural consistency and invalid references or transitions, reporting structural defects as diagnostics. These validation results serve as feedback to the generator for configuration update. The loop performs at most two revisions. A configuration is treated as executable only after the validator accepts it.

3.2 Executable Environment

Environment state. The environment loads a `StoryConfig` to start an episode (Fig. 2, panel 3), where it maintains an explicit semantic state x_t, covering the player and NPCs' locations, possessions, and disclosed information, along with a separate event history h_t, recording which events have occurred and when. Together, they define the context from which the story continues to evolve.

Conditional execution semantics. The environment executes by propagating events to advance stories. An event carries a trigger and enabling conditions. A trigger signals that something has happened and it arises through two runtime paths: 1) published by the event system to its subscribers once conditions are satisfied; 2) published by the environment once the defined relevant change in world state occurs. This keeps the environment responsive instead of merely reactive. Once an event is triggered, its conditions are evaluated against the current state x_t, the event history h_t, or both. If the conditions hold, the event fires, applies its state updates, and activates the rules that depend on it. The first ending to fire closes the episode; later endings are discarded while non-ending events continue, so a trajectory terminates in exactly one outcome.

History-dependent consequences. The environment determines how an action changes the state. Executed events can reveal previously unavailable information, elicit an NPC response, enable or disable subsequent actions, or alter which endings remain reachable. Every such effect updates explicit state, thus no free-form input could change the story except through a defined transition. Two trajectories that diverged earlier therefore face different reachable endings, and that divergence follows from the environment's state rather than from the generated text.

NPC execution. NPCs are controlled by a separate manager that plans from character background toward a runtime-adjustable task, with relevant events triggering replanning. NPC behavior can therefore adapt as the story evolves, while information availability, event conditions, and reachable outcomes remain governed by the executable environment. Participants observe only realized NPC behavior rather than the underlying plan, so information is obtained through interaction.

3.3 Shared Interaction and Trajectory Logging

Shared interface. We expose one interface to both agent and human participants though each reaches it through a different client (Fig. 2, panel 4): an agent connects through a tool-calling protocol while a human uses a web interface (Fig. 3). It guarantees that they operate over the same environment state, observations, action semantics, and transition rules.



Figure 3: The human web interface.

Participant input. At each turn, a participant receives a language observation of the perceivable state together with the actions available at that point. Participants may act through either a named executable action with explicit parameters or free-form language. The direct-action vocabulary covers generic behaviors such as movement, attention, manipulation, speech, and posture. A free-form response is expressed in the participant's own words and can cover behavior outside that vocabulary.

Action grounding. Since the execution requires defined transitions, events, and state updates, free-form text input cannot refer to them, which requires grounding to executable semantics. The resolver first attempts to match the input to a defined story event, and otherwise maps it to an executable action. Grounded input then enters the same execution path as a direct action, so the participant may say anything while the environment determines precisely what that saying does. Grounding can fail; when it does, the turn produces no semantic state change, and the failure is recorded explicitly.

Trajectory record. The environment records every turn for both participants in the same schema (Fig. 2, panel 5): the observation presented, the input as given and its grounding, the resulting state and history updates, NPC responses, and reached ending. Agent and human trajectories therefore expose the same execution-level evidence for subsequent trajectory analysis.

4 The PraxiScene Benchmark

Building on the generation and execution substrate, we instantiate PraxiScene as a frozen benchmark for evaluating open-goal social agency in interactive social situations, as shown in Fig. 4.

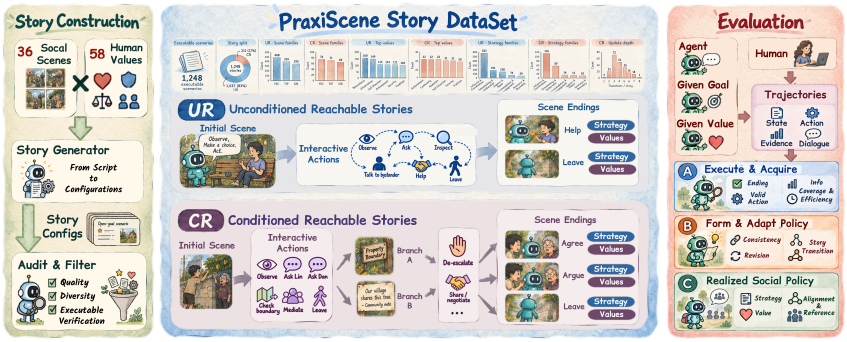

Figure 4: **The PRAXISCENE benchmark.** Stories are generated and audited before inclusion; the frozen corpus contains UR and history-dependent CR stories with Strategy–Value semantics; shared executable trajectories support process-level evaluation and controlled Goal, Value, and human comparisons.

4.1 BENCHMARK CONSTRUCTION, ANNOTATION AND AUDIT

Story construction. Benchmark candidates are generated from the two authoring controls introduced in Sec. 3: a social *Scene* and a *Value*. Scenes are drawn from the 36 social situation classes of Bhatia et al. (2026), organized into three higher-order families: *Family, Education, and Community* (FEC), *Transactional, Institutional, and Professional* (TIP), and *Social, Interpersonal, and Romantic* (SIR), while values use the dimensions from Schwartz et al. (2012). These variables define the context and motivational concern from which a story is constructed; they do not prescribe which action, ending, strategy, or realized value an evaluated participant should produce. There are two types of stories: **Unconditioned Reachable (UR)** stories and **Conditioned Reachable (CR)** stories. Both use the same executable interaction interface. The distinction concerns how story endings become reachable. In UR stories, all endings are reachable from the initial state, while in CR stories, some endings becomes reachable only after the trajectory satisfies an intermediate state. Earlier behavior therefore changes the set of endings available later in CR stories.

Semantic annotations. PraxiScene represents realized social agency along two complementary behavioral semantics: **Strategy** describes *how* a social decision is enacted, whereas **Value** describes the value content expressed by the realized behavior. CR stories additionally receive Strategy annotations at registered intermediate states. Both annotation spaces use two-level taxonomies. The Strategy taxonomy comprises 25 fine-grained strategies grouped into five broad families: *Situation Sensemaking & Deliberation*, *Interpersonal Influence & Coordination*, *Support & Relational Repair*, *Resistance & Boundary Protection*, and *Accommodation & Disengagement*, synthesized from established social interaction strategies (Crick & Dodge, 1994; Yukl et al., 2008; House, 1981; Fransen et al., 2015; Kilmann & Thomas, 1977). The Value taxonomy builds on the Schwartz Value Survey with an additional category *Personal Safety*, yielding 58 fine-grained labels in 11 broad categories (Schwartz, 1992; Schwartz et al., 2012). Construction-side Values are not inherited as realized Value labels; they guide story generation, whereas the latter describe the behavior represented by each ending. Neither Strategy nor Value defines a preferred outcome or competence score.

Strategy and Value labels are produced by a language model judge under a frozen rubric, with the prompt iterated against an accumulating set of test cases until it reproduced them. The same prompt is applied identically across all stories. Value labels are additionally validated against human annotation, reaching 90.4% agreement on endings. We report agreement, unresolved rate, and annotation coverage, together with the full annotation protocol, in Appendix C.2. Items without reliable behavioral labels remain executable benchmark instances but are excluded from label-dependent analyses.

Audit and filtering. The generated candidate pool is reduced to the released benchmark through the Quality, Diversity, and Executable Verification stages summarized in Fig. 4. Candidates pass structural acceptance, static validation of the executable event graph, and a runtime replay that confirms reachability of every candidate ending before inclusion, and are then retained only if their terminal decisions yield reliable Strategy and Value annotations with sufficient label diversity. Failures are excluded rather than repaired, and the retained store is frozen together with its configurations.

The benchmark contains **1,248 executable social situations**, including **1,037 Unconditioned Reachable (UR)** stories and **211 Conditioned Reachable (CR)** stories. Beyond this structural split, the corpus spans all 36 social-situation classes and 58 values.

4.2 Evaluating Social Agency

The canonical setting leaves the social decision problem with the participant: no specified task-level objective, no full information, and free-form actions. Each run therefore exposes not only where the interaction ends, but how the participant gathers information, acts, and changes course as the situation unfolds. We evaluate trajectories directly rather than a single Social Agency scalar score.

Trajectory evidence. We evaluate Social Agency at two complementary resolutions: the trajectory through which a situation is handled, and the behavior ultimately realized at its ending. The trajectory captures how an agent acquires information, forms a policy, and revises that policy as the interactive story proceeds. We first record basic execution and trajectory descriptors. R_{end} represents the rate of trajectories that reach a valid ending, R_{grd} measures the rate of free-form actions successfully resolved to executable actions, and N_{act} describes interaction actions relative to the shortest executable path length. To characterize information use and policy evolution, we define I_C to measure how much information available along the visited states is acquired, and I_E measures unique information gain per active information-seeking action. C_{str} compares the agent's stated action-guiding policy with the Strategy subsequently realized. For CR stories, S_{str} records Strategy changes across registered state updates. We additionally report P_{dep} as a descriptor of how much of the story's conditional structure is traversed. These quantities characterize the decision process rather than impose a preferred trajectory; neither shorter interaction nor more frequent policy revision is assumed to be intrinsically better.

Behavioral realization. The semantic annotations introduced above provide the terminal behavioral readout of each completed trajectory. For every realized ending, we record its annotated **Strategy** and **Value**; for CR stories, intermediate Strategy labels additionally make policy changes across registered state updates observable. We analyze these annotations as distributions over realized behavior rather than as correctness scores. Together with the trajectory measures, they distinguish changes in *how* an agent navigates the interaction from changes in *what behavioral policy* it ultimately realizes.

Controlled specification interventions. PraxiScene supports controlled variants that return selected parts of the social decision problem to the evaluator while preserving the underlying story. A **Goal** specification states what outcome the participant should pursue; an **Information** specification exposes decision-relevant facts that would otherwise need to be acquired through interaction; and a **Value** specification supplies an explicit normative orientation for the response. These controls can be introduced independently, allowing the same executable situation to be evaluated under different allocations of decision authority. They are used diagnostically: the benchmark does not treat any supplied specification as the canonical or normatively preferred condition.

5 Experiments

Our experiments begin from a single allocation of decision authority. In the canonical condition, the evaluator specifies the social situation but leaves three components of its resolution to the agent: the *Goal* to pursue, the decision-relevant *Information* to acquire, and the *Value* that may orient its response. Social Agency may then become visible in the interaction process or in the policy eventually realized. We operationalize this as a 3×3 analysis:

$$\underbrace{\{\textsc{Goal}, \textsc{Information}, \textsc{Value}\}}_{\text{delegated decision component}} \times \underbrace{\left\{ \begin{array}{c} \textsc{Trajectory} \\ \textsc{Terminal Strategy} \\ \textsc{Terminal Value} \end{array} \right\}}_{\text{behavioral readout}}.$$

The rows ask which part of the decision problem is supplied by the evaluator; the columns ask where the resulting behavioral change appears. This design separates a change in *how* an agent navigates a situation from a change in the kind of resolution it ultimately enacts.

Systems and experimental matrix. We evaluate six LLM backbones with a fixed ReAct-style harness on all 1,037 UR and 211 CR stories under the canonical Open Goal + Stepwise In-

Table 1: **Open-goal social agency profiles in PRAXISCENE.** Metrics characterize trajectory enactment, information engagement, and policy coherence and dynamics; † denotes CR-only measures.

	Executability	Trajectory Enactment			Information Engagement		Policy Dynamics	
Agent System	R_{grd}	R_{end}	N_{act}	$P^{\dagger}_{dep}$	I_C	I_E	C_{str}	$S^{\dagger}_{str}$
Deepseek-V4.1-Flash · ReAct	0.852	**0.939**	16.007	0.543	0.422	0.450	0.270	0.726
Qwen3.8-Flash · ReAct	0.860	0.920	12.532	0.507	0.320	0.425	0.267	0.714
GLM 5.3 · ReAct	**0.911**	0.887	12.127	0.472	0.295	0.395	0.288	0.703
GPT-5.6 Terra · ReAct	0.770	0.933	**8.569**	0.559	0.199	0.343	0.254	0.643
Claude Sonnet 5 · ReAct	0.756	0.719	8.647	0.457	0.181	0.322	0.295	0.605
Gemini 3.8 Flash · ReAct	0.855	0.807	10.776	0.550	0.296	0.438	—	0.747
Deepseek-V4.1-Flash · OpenCode	0.852	0.933	16.325	**0.600**	0.418	0.469	0.268	0.734
GPT-5.6 Terra · OpenCode	0.688	0.882	8.870	0.531	0.198	0.355	0.222	0.731

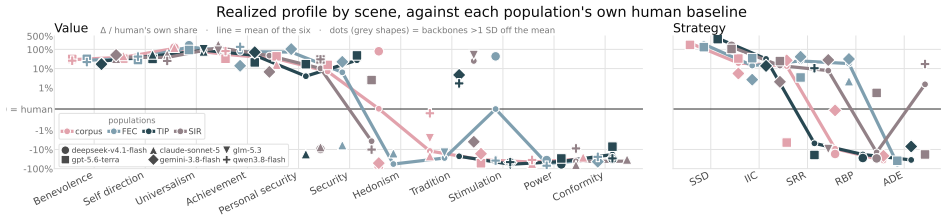

Figure 5: **Open-goal agents exhibit systematic human-relative policy shifts.** Ending Strategy and Value shares are shown relative to matched human baselines. Across backbones, agents consistently favor Sensemaking over Accommodation, with additional scene-dependent variation.

FORMATION + PLAIN VALUE condition. DeepSeek-V4.1-Flash and GPT-5.6-Terra serve as anchor systems for the matched specification experiments. For Goal and Information, we evaluate the full 2×2 crossing of OPEN versus GIVEN GOAL and STEPWISE versus FULL INFORMATION; Value is evaluated through a separate matched intervention. The primary 3×3 analysis compares each specification change with its matched canonical run across trajectory, terminal Strategy, and terminal Value readouts. Architecture variants and protocol-matched human trajectories provide additional checks on whether these effects are specific to the ReAct harness or to model participants.

5.1 CANONICAL OPEN-GOAL BEHAVIOR

We first examine the canonical setting, where Goal, Information, and Value selection remain with the agent. Reaching an executable ending is rarely the main bottleneck: five of six ReAct agents exceed 80% ending rate, with DeepSeek reaching 93.9% (Table 1). The larger variation appears in information use and policy formation. Information coverage ranges from 0.181 to 0.422, while stated–realized Strategy correspondence remains 0.254–0.295 among systems with complete measurements. Interaction length is strongly correlated with information coverage across backbones ($r = 0.95$ in UR; $r = 0.97$ in CR), indicating a consistent coupling between engagement and interaction cost. System ordering depends on the behavioral readout. GLM achieves the highest grounding rate ($R_{grd} = 0.911$); DeepSeek reaches the most endings and the broadest information coverage; GPT reaches 93.3% of endings with roughly half as many actions as DeepSeek (8.57 vs. 16.01). No system dominates execution, information engagement, and policy coherence simultaneously. Terminal success alone therefore obscures substantial differences in how agents resolve the same open-goal situations. Ending profiles differ systematically from human behavior (Fig. 5). Across six backbones, Sensemaking is 214% higher overall and elevated in every scene family (+187% in family, engagement, and community situations (FEC), +385% in transactional, institutional, and professional situations (TIP), +132% in social, interpersonal, and romantic situations (SIR)), while Accommodation is 38% lower overall and approaches parity only in SIR. Influence is also elevated (+19%), whereas Resistance reverses from +18% in FEC to −18% in TIP, indicating a shared deliberative tendency with scene-dependent boundary behavior. Value profiles show the same combination of regularity and context dependence.

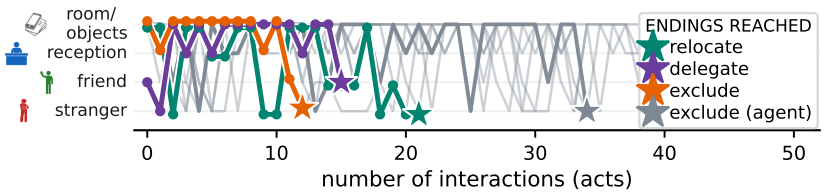

Figure 6: **Humans reach endings where open-goal agent runs keep acting.** Each line is one trajectory. The vertical axis is the action target: the other claimant, the friend, the receptionist, or the room and its objects. Coloured lines are human plays; grey lines the eight agent runs; and stars the action that ended a play. The three human endings are *exclude* (the stranger is asked to leave), *delegate* (the friend is asked to resolve the conflict), and *relocate* (the stranger is moved out to wait).

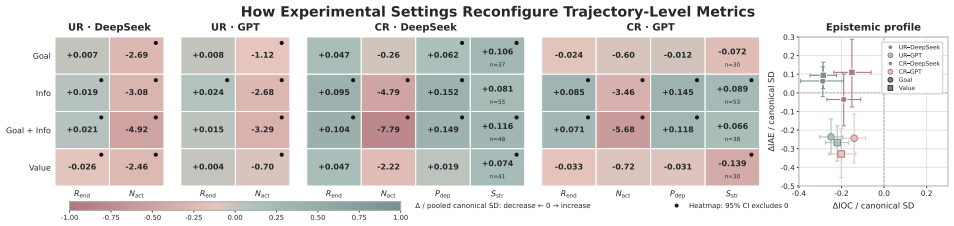

Figure 7: **Information specification most strongly compresses the trajectory.** Heatmaps show matched changes from the canonical condition for UR/CR and both anchor systems; the final panel shows Goal- and Value-induced shifts in the I_C–I_E plane.

Human-agent divergence in one story. Fig. 6 shows one story, in which the player arrives at the booked private room and finds the friend already at the table, with an appointment slip on it. A stranger seated opposite holds an identical slip. All three human plays reached an ending within 22 actions. In comparison, of the eight open-goal agent runs, only seven reached no ending. One reading of this asymmetry is that a terminating action in this story asks one of the other people to act, whereas the agent runs kept gathering information, avoiding a commitment whose consequences would fall on other people while the situation was still uncertain. The strategy profile in Fig. 5 shows this pattern across the corpus, and the panel shows it in one play.

5.2 Distinct Effects of Returned Decision Authority

Finding 1: Information removes search burden but does not align terminal policy. Among single-component interventions, providing decision-relevant Information produces the largest trajectory effect (Fig. 7). Interaction shortens by 2.68–3.08 actions in UR and 3.46–4.79 in CR, while ending rates rise by 1.9–2.4 and 8.5–9.5 percentage points, respectively. In CR, these shorter trajectories also traverse more conditional structure and more often revise Strategy after state updates. Information acquisition is therefore the largest removable interaction burden in the canonical setting. This efficiency gain does not make terminal policies more similar. In CR, Information lowers cross-model Strategy agreement from 0.680 to 0.573 and primary-Value agreement from 0.640 to 0.533 (Fig. 8). Replicated category shifts accompany this change: Resistance decreases while Benevolence increases for both anchor systems. Thus, removing the epistemic burden can improve execution while diversifying the policies ultimately realized.

Finding 2: Goal specification aligns terminal policies more than it reduces interaction cost. Goal specification reduces self-directed information coverage in all four structure×backbone cells, but does not reproduce Information's uniform trajectory compression (Fig. 7). Its clearest effect appears at the ending: in CR, Strategy agreement rises from 0.680 to 0.760 and primary-Value agreement from 0.640 to 0.733 (Fig. 8). Supplying an objective therefore makes models converge more strongly on terminal policies without removing the same search burden. The joint Goal+Information condition sharpens this distinction. In CR, both systems shift toward Sensemaking and away from Resistance, whereas the same Sensemaking shift reverses in UR. Goal specification therefore increases within-context agreement without imposing a single policy across story structures.

Finding 3: Value guidance selectively steers realized orientation. Value specification also reduces information coverage, but its execution effects are smaller and backbone-dependent (Fig. 7).

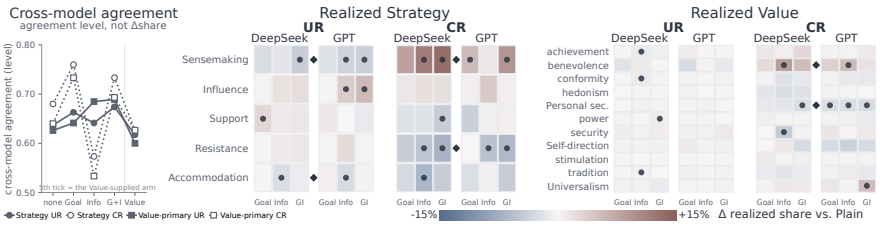

Figure 8: **Goal specification aligns terminal policy, whereas Information can diversify it.** Left: cross-model agreement across specification conditions. Center and right: matched Strategy and Value shifts by structure and backbone.

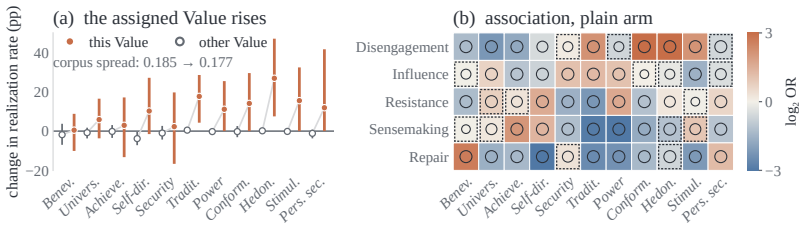

Figure 9: **Value guidance selectively changes realized Value.** (a) Matched changes for assigned Values and non-target controls. (b) Canonical Strategy–Value associations across six backbones.

Its strongest effect is instead selective realization. Across 22 UR backbone×target estimates, the assigned Value increases by 10.8 percentage points on average, compared with -0.8 points for Values not assigned to the story (Fig. 9). Value guidance therefore changes the requested normative content rather than inducing a generic shift in terminal behavior. The steering follows structure already present in the canonical policy space. For every Value, the strongest associated Strategy has the same sign across all six backbones, and for 10 of 11 Values across all scene families. After removing global Strategy and Value shifts, Value-specific Strategy changes remain aligned with these associations in UR ($r = 0.300$, 95% CI $[0.051, 0.400]$), while the CR interval includes zero. Value guidance therefore steers behavior along recurring Strategy–Value modes rather than replacing the underlying policy structure.

6 Conclusion

We introduced PraxiScene to evaluate open-goal social agency through 1,248 executable UR and CR situations with grounded, history-dependent interaction and terminal Strategy–Value semantics. The canonical setting leaves Goal, Information, and Value selection to the participant, while matched controls return each decision component to the evaluator. These interventions produce distinct behavioral signatures. Information removes the largest interaction burden but can reduce terminal-policy agreement; Goal increases agreement without producing the same trajectory compression; and Value selectively shifts realized normative orientation. Performance under supplied structure therefore depends not only on how well an agent executes a policy, but also on which decisions the evaluator has already made for it.

AI use statement

Required disclosure. We used generative AI to generate the synthetic story dataset, to support qualitative and thematic annotation of stories and trajectories, and to implement methods, including the benchmark harness, the experiment pipeline, and trajectory recording. We have not used generative AI for the conceptual framework, the properties of the setting, the experimental design, or the interpretation of results.

Recommended disclosure. We used generative AI to create figures from human-authored drafts, to identify relevant literature, and to edit the manuscript for readability.

We reviewed all AI-assisted work: generated stories were inspected for structural validity, the language-model annotations were validated against human annotation, and LLM-assisted code was tested through the reported workflows. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

The human study in this work is **E5** (Section 5), which collects human trajectories through the same executable free-form interface used for agents. This study was reviewed within our research department, which determined that formal ethics-board approval was not required. Participation was voluntary, and participants were informed that their interaction trajectories would be recorded and analyzed and were compensated for their time. No identifying information is collected or linked to any trajectory, all reported results are aggregate, and human trajectories are not released.

REPRODUCIBILITY STATEMENT

PRAXISCENE consists of a story generator, a structural validator, an executable environment, and a shared interface for agents and human participants. We release the framework implementation, the frozen benchmark specifications, the annotation and metric implementations, and the analysis scripts that reproduce the reported tables and figures at `https://anonymous.4open.science/r/PraxiScene-FB5F/`.

Appendix B.2 and Appendix B.3 provide the story specification schema, the runtime state and event semantics, and the trajectory record format; Appendix B.4 documents the action-grounding procedure; and Appendix C.1 documents benchmark construction, filtering, and annotation. Appendix F lists the released artifacts with version identifiers, and Appendix F states the model and runtime configuration needed to reproduce the experiments and the known reproducibility constraints.

REFERENCES

Sudeep Bhatia, Andrew Yang, and Taya R. Cohen. The structure of social situations: Insights from the large-scale automated coding of text. *Psychological Science*, 37(3):180–205, 2026. doi: 10.1177/09567976261418946.

Matthew Chang, Gunjan Chhablani, Alexander Clegg, Mikael Dallaire Cote, Ruta Desai, Michal Hlavac, Vladimir Karashchuk, Jacob Krantz, Roozbeh Mottaghi, Priyam Parashar, Siddharth Patki, Ishita Prasad, Xavier Puig, Akshara Rai, Ram Ramrakhya, Daniel Tran, Joanne Truong, John M. Turner, Eric Undersander, and Tsung-Yen Yang. PARTNR: A benchmark for planning and reasoning in embodied multi-agent tasks. In *International Conference on Learning Representations*, 2025. URL `https://proceedings.iclr.cc/paper_files/paper/2025/hash/a3cf318fbeec1126da21e9185ae9908c-Abstract-Conference.html`.

Hongzhan Chen, Hehong Chen, Ming Yan, Wenshen Xu, Gao Xing, Weizhou Shen, Xiaojun Quan, Chenliang Li, Ji Zhang, and Fei Huang. SocialBench: Sociality evaluation of role-playing conversational agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 2108–2126. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.125.

Nicki R. Crick and Kenneth A. Dodge. A review and reformulation of social information-processing mechanisms in children's social adjustment. *Psychological Bulletin*, 115(1):74–101, 1994. doi: 10.1037/0033-2909.115.1.74.

Denis Emelin, Ronan Le Bras, Jena D. Hwang, Maxwell Forbes, and Yejin Choi. Moral stories: Situated reasoning about norms, intents, actions, and their consequences. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 698–718. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.emnlp-main.54.

Marieke L. Fransen, Edith G. Smit, and Peeter W. J. Verlegh. Strategies and motives for resistance to persuasion: An integrative framework. *Frontiers in Psychology*, 6:1201, 2015.

Hitesh Goel and Hao Zhu. LIFELONG SOTOPIA: Evaluating social intelligence of language agents over lifelong social interactions. *arXiv preprint arXiv:2506.12666*, 2025. doi: 10.48550/arXiv.2506.12666.

Dan Hendrycks, Collin Burns, Steven Basart, Andrew Critch, Jerry Li, Dawn Song, and Jacob Steinhardt. Aligning AI with shared human values. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=dNy_RKzJacY.

James S. House. *Work Stress and Social Support*. Addison-Wesley, Reading, MA, 1981. ISBN 0201031019.

Tiancheng Hu, Joachim Baumann, Lorenzo Lupo, Nigel Collier, Dirk Hovy, and Paul Röttger. SimBench: Benchmarking the ability of large language models to simulate human behaviors. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/2fc4d6a6400addda7493b9db258c34e0-Abstract-Conference.html.

Yizhe Huang, Xingbo Wang, Hao Liu, Fanqi Kong, Aoyang Qin, Min Tang, Song-Chun Zhu, Mingjie Bi, Siyuan Qi, and Xue Feng. AdaSociety: An adaptive environment with social structures for multi-agent decision-making. In *Advances in Neural Information Processing Systems*, volume 37, pp. 35388–35413. Curran Associates, Inc., 2024. doi: 10.52202/079017-1115. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/3e4d8407cb468850f2f8f4a949e64bf0-Abstract-Datasets_and_Benchmarks_Track.html.

Ralph H. Kilmann and Kenneth W. Thomas. Developing a forced-choice measure of conflict-handling behavior: The "mode" instrument. *Educational and Psychological Measurement*, 37 (2):309–325, 1977. doi: 10.1177/001316447703700204.

Hyunwoo Kim, Melanie Sclar, Xuhui Zhou, Ronan Le Bras, Gunhee Kim, Yejin Choi, and Maarten Sap. FANToM: A benchmark for stress-testing machine theory of mind in interactions. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 14397–14413. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.890.

Huayi Lai, Shichao Song, Simin Niu, Hanyu Wang, Jiawei Yang, Zhouxing Wang, Zhiqiang Yin, and Xun Liang. RoleCDE: Benchmarking and mitigating role–alignment trade-offs in role-playing agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 2226–2248. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.106. URL https://aclanthology.org/2026.findings-acl.106/.

Yuxuan Lu, Jing Huang, Yan Han, Bingsheng Yao, Sisong Bei, Yaochen Xie, Yisi Sang, Qi He, and Dakuo Wang. Can LLM agents simulate multi-turn human behavior? evidence from real online customer behavior data. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 43961–43977. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.2034. URL https://aclanthology.org/2026.acl-long.2034/.

Xinyi Mou, Jingcong Liang, Jiayu Lin, Xinnong Zhang, Xiawei Liu, Shiyue Yang, Rong Ye, Lei Chen, Haoyu Kuang, Xuanjing Huang, and Zhongyu Wei. AgentSense: Benchmarking social intelligence of language agents through interactive scenarios. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 4975–5001. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-long.257.

Seyed Mahed Mousavi, Edoardo Cecchinato, Lucia Horníková, and Giuseppe Riccardi. Garbage in, reasoning out? why benchmark scores are unreliable and what to do about it. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 1747–1759. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-eacl.89. URL https://aclanthology.org/2026.findings-eacl.89/.

Alexander Pan, Jun Shern Chan, Andy Zou, Nathaniel Li, Steven Basart, Thomas Woodside, Hanlin Zhang, Scott Emmons, and Dan Hendrycks. Do the rewards justify the means? measuring trade-offs between rewards and ethical behavior in the MACHIAVELLI benchmark. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 26837–26867. PMLR, 2023. URL https://proceedings.mlr.press/v202/pan23a.html.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22. ACM, 2023. doi: 10.1145/3586183.3606763.

Viet Thanh Pham, Lizhen Qu, Thuy-Trang Vu, Gholamreza Haffari, and Dinh Phung. LiveCultureBench: a multi-agent, multi-cultural benchmark for large language models in dynamic social simulations. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27076–27105. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.1247.

Jinghua Piao, Yuwei Yan, Jun Zhang, Nian Li, Junbo Yan, Xiaochong Lan, Zhihong Lu, Zhiheng Zheng, Jing Yi Wang, Di Zhou, Chen Gao, Fengli Xu, Fang Zhang, Ke Rong, Jun Su, and Yong Li. AgentSociety: Large-scale simulation of LLM-driven generative agents advances understanding of human behaviors and society. *iFuture*, 2026. doi: 10.26599/IF.2026.9710004. URL https://doi.org/10.26599/IF.2026.9710004. Just Accepted.

Giorgio Piatti, Zhijing Jin, Max Kleiman-Weiner, Bernhard Schölkopf, Mrinmaya Sachan, and Rada Mihalcea. Cooperate or collapse: Emergence of sustainable cooperation in a society of LLM agents. In *Advances in Neural Information Processing Systems*, volume 37, pp. 111715–111759. Curran Associates, Inc., 2024. doi: 10.52202/079017-3548. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/ca9567d8ef6b2ea2da0d7eed57b933ee-Abstract-Conference.html.

Siyuan Qi, Shuo Chen, Yexin Li, Xiangyu Kong, Junqi Wang, Bangcheng Yang, Pring Wong, Yifan Zhong, Xiaoyuan Zhang, Zhaowei Zhang, Nian Liu, Yaodong Yang, and Song-Chun Zhu. CivRealm: A learning and reasoning odyssey in civilization for decision-making agents. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/76d2f8e328e1081c22a77ca0fa330ca5-Abstract-Conference.html.

Yuanyi Ren, Haoran Ye, Hanjun Fang, Xin Zhang, and Guojie Song. ValueBench: Towards comprehensively evaluating value orientations and understanding of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2015–2040. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.111.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social IQa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4463–4473. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1454.

Shalom H. Schwartz. Universals in the content and structure of values: Theoretical advances and empirical tests in 20 countries. In *Advances in Experimental Social Psychology*, volume 25, pp. 1–65. 1992. doi: 10.1016/S0065-2601(08)60281-6.

Shalom H. Schwartz, Jan Cieciuch, Michele Vecchione, Eldad Davidov, Ronald Fischer, Constanze Beierlein, Alice Ramos, Markku Verkasalo, Jan-Erik Lönnqvist, Kursad Demirutku, Ozlem Dirilen-Gumus, and Mark Konty. Refining the theory of basic individual values. *Journal of Personality and Social Psychology*, 103(4):663–688, 2012. doi: 10.1037/a0029393.

Alexander Sasha Vezhnevets, John P. Agapiou, Avia Aharon, Ron Ziv, Jayd Matyas, Edgar A. Duéñez-Guzmán, William A. Cunningham, Simon Osindero, Danny Karmon, and Joel Z. Leibo.

Generative agent-based modeling with actions grounded in physical, social, or digital space using Concordia. *arXiv preprint arXiv:2312.03664*, 2023. doi: 10.48550/arXiv.2312.03664. URL https://arxiv.org/abs/2312.03664.

Chenxu Wang, Bin Dai, Huaping Liu, and Baoyuan Wang. Towards objectively benchmarking social intelligence of language agents at the action level. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 8885–8897. Association for Computational Linguistics, 2024a. doi: 10.18653/v1/2024.findings-acl.526.

Ruiyi Wang, Haofei Yu, Wenxin Zhang, Zhengyang Qi, Maarten Sap, Yonatan Bisk, Graham Neubig, and Hao Zhu. SOTOPIA-π: Interactive learning of socially intelligent language agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12912–12940. Association for Computational Linguistics, 2024b. doi: 10.18653/v1/2024.acl-long.698.

Zixiang Xu, Yanbo Wang, Yue Huang, Haomin Zhuang, Yujun Zhou, Jiayi Ye, Sixian Li, Zirui Song, Lang Gao, Chenxi Wang, Zhaorun Chen, Wang Pan, Yue Zhao, Jieyu Zhao, Xiangliang Zhang, and Xiuying Chen. SocialMaze: A benchmark for evaluating and enhancing social reasoning in large language models in complex social environments. In *Findings of the Association for Computational Linguistics: EMNLP 2026*, Budapest, Hungary, October 2026. Association for Computational Linguistics. To appear.

Yashwanth YS, Ruichen Wang, Shihua Zeng, Xuhui Zhou, Koichi Onoue, Vasudha Varadarajan, and Maarten Sap. SOTOPIA-TOM: Evaluating information management in multi-agent interaction with theory of mind. *arXiv preprint arXiv:2605.02307*, 2026. doi: 10.48550/arXiv.2605.02307. URL https://arxiv.org/abs/2605.02307.

Gary Yukl, Charles F. Seifert, and Carolyn Chavez. Validation of the extended influence behavior questionnaire. *The Leadership Quarterly*, 19(5):609–621, 2008. doi: 10.1016/j.leaqua.2008.07.006.

Jun Zhang, Yuwei Yan, Junbo Yan, Zhiheng Zheng, Jinghua Piao, Depeng Jin, and Yong Li. A parallelized framework for simulating large-scale LLM agents with realistic environments and interactions. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pp. 1339–1349. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-industry.94.

Jinfeng Zhou, Yuxuan Chen, Yihan Shi, Xuanming Zhang, Leqi Lei, Yi Feng, Zexuan Xiong, Miao Yan, Xunzhi Wang, Yaru Cao, Jianing Yin, Shuai Wang, Quanyu Dai, Zhenhua Dong, Hongning Wang, and Minlie Huang. SocialEval: Evaluating social intelligence of large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 30958–31012. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1496.

Xuhui Zhou, Hao Zhu, Leena Mathur, Ruohong Zhang, Haofei Yu, Zhengyang Qi, Louis-Philippe Morency, Yonatan Bisk, Daniel Fried, Graham Neubig, and Maarten Sap. SOTOPIA: Interactive evaluation for social intelligence in language agents. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/b3075b88e583a0e98d8b24338a613060-Abstract-Conference.html.

Ziheng Zhou, Huacong Tang, Mingjie Bi, Wanying He, Fang Sun, Yizhou Sun, Ying Nian Wu, Demetri Terzopoulos, Yipeng Kang, and Fangwei Zhong. Why are we moral? an LLM-based agent simulation approach to the study of moral evolution. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28993–29046. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.1339. URL https://aclanthology.org/2026.acl-long.1339/.

Caleb Ziems, Jane Dwivedi-Yu, Yi-Chia Wang, Alon Halevy, and Diyi Yang. NormBank: A knowledge bank of situational social norms. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7756–7776. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-long.429.

A Detailed Related Work

Research relevant to PRAXISCENE spans several traditions that place different parts of a social decision problem inside the evaluation. Static and constituent benchmarks typically supply the situation and ask models to infer social knowledge, mental states, norms, or values. Sequential benchmarks add branching action trajectories but usually retain game rewards or other explicit objectives. Interactive social-agent benchmarks move evaluation into live, multi-turn exchanges, while generally providing an organizing social or task goal. A parallel line of work develops persistent or executable multi-agent worlds for studying coordination, cooperation, and social dynamics. PRAXISCENE builds on all four directions, but focuses on a different allocation of decision responsibility: the agent must determine what the situation calls for and what information to obtain as part of the interaction itself.

Social knowledge, norms, theory of mind, and values. Early social-intelligence benchmarks primarily isolate constituent reasoning capabilities. SOCIALIQA evaluates commonsense reasoning about people's motivations, emotions, and likely consequences in everyday social situations (Sap et al., 2019). ETHICS evaluates predictions of human ethical judgments across commonsense morality, justice, deontology, virtue ethics, and utilitarianism (Hendrycks et al., 2021). MORAL STORIES introduces structured branching narratives linking intentions, norms, actions, and consequences, including generation tasks in which actions must achieve predefined goals under normative constraints (Emelin et al., 2021).

Theory-of-mind and norm-centered resources make the supplied context more structured. FANTOM stress-tests theory of mind through question answering over information-asymmetric conversations, using multiple probes that require reasoning about the same underlying interaction (Kim et al., 2023). NORMBANK contains 155k situational norms grounded in multivalent sociocultural frames, including setting, roles, attributes, and contextual constraints, and explicitly studies nonmonotonicity under changes to those frames (Ziems et al., 2023). Values have likewise become direct evaluation objects: VALUEBENCH aggregates 44 psychometric inventories covering 453 value dimensions to evaluate both value orientation and value understanding (Ren et al., 2024), while ROLECDE evaluates role-playing agents under structured conflicts between role-specific values and alignment-oriented constraints (Lai et al., 2026). SOCIALBENCH evaluates role-playing agents at both individual and group levels and reports that strength at one level does not imply strength at the other (Chen et al., 2024).

These benchmarks provide increasingly rich descriptions of the beliefs, norms, and values that can support social behavior, but their evaluation object remains largely a judgment, prediction, ranking, or response conditioned on a supplied question or frame. PRAXISCENE instead uses situation and value structure to construct and characterize executable interactions, while observing how an agent recruits information and realizes behavior over a trajectory.

Sequential and interactive social-agent benchmarks. A key sequential precedent is MACHIAVELLI, a benchmark of 134 Choose-Your-Own-Adventure games with more than half a million social-decision scenarios (Pan et al., 2023). Agents act through branching narratives, while scenarios are annotated for power-seeking, disutility, and ethical violations, enabling analysis of trade-offs between game reward and harmful behavior. This is structurally relevant to PRAXISCENE: both expose socially consequential trajectories rather than isolated responses. The evaluation target differs, however. MACHIAVELLI studies competent and ethical reward pursuit inside existing game structures; PRAXISCENE's canonical condition removes a task-level social objective and treats realized Strategy and Value as trajectory semantics rather than a global reward or ethical penalty.

The closest literature places agents inside live social interaction. SOTOPIA provides open-ended role-play in which agents coordinate, collaborate, exchange, and compete while pursuing supplied social goals, and uses SOTOPIA-Eval together with human–agent role-play to evaluate the resulting interactions (Zhou et al., 2024). SOTOPIA-π further shows that the environment can support interactive learning and reports that LLM-based social evaluators can overestimate agents trained specifically for social interaction (Wang et al., 2024b). AGENTSENSE constructs 1,225 script-derived multi-turn scenarios informed by Dramaturgical Theory and evaluates both goal completion and implicit reasoning, including reasoning over private information (Mou et al., 2025). SOCIALEVAL uses manually constructed bilingual "world trees" to combine outcome-oriented social-goal achievement

with process-oriented interpersonal-ability evaluation (Zhou et al., 2025). These works establish increasingly structured and process-sensitive forms of social-agent evaluation, while retaining an organizing goal or outcome target.

Other benchmarks strengthen execution and information structure. STSS grounds social tasks in a multi-agent sandbox and evaluates agents through objective action-level goal achievement in simulator state (Wang et al., 2024a). LIVECULTUREBENCH places a focal resident with a supplied daily goal inside a dynamic simulated town and jointly tracks task progress, socio-cultural norm adherence, and verifier uncertainty (Pham et al., 2026). SOTOPIA-TOM directly targets information management in multi-party interaction: scenarios provide primary objectives, partitioned private knowledge, sharing constraints, and public and private channels, and evaluation measures information sharing and acquisition, coordination, efficiency, and privacy (YS et al., 2026). It is therefore particularly close to PRAXISCENE's information-acquisition dimension. The distinction is that PRAXISCENE couples information seeking to an unspecified task-level social objective and experimentally compares Stepwise against Provided Information on matched stories.

Longer temporal dependence is studied by LIFELONG-SOTOPIA, which links multiple goal-directed social episodes and evaluates whether agents retain and use information accumulated across interactions (Goel & Zhu, 2025). PRAXISCENE's history dependence is instead intra-episode: earlier actions can reveal information, update state, and change subsequent opportunities within one executable story. Taken together, existing interactive benchmarks already cover open-ended dialogue, process evaluation, private information, simulator-grounded action, culture, and long-range history. The remaining distinction for our study is whether the organizing objective itself is supplied or must be constructed by the agent; PRAXISCENE makes this boundary an experimental variable through matched Open-Goal and Given-Goal conditions.

Executable social and multi-agent simulation environments. A broader literature develops persistent social worlds rather than fixed benchmark episodes. Generative Agents introduced memory, reflection, and planning mechanisms for believable behavior in an interactive small-town sandbox (Park et al., 2023). CONCORDIA generalizes this idea into a framework for generative agent-based modeling across physical, social, and digital settings: agents express actions in natural language and a Game Master resolves their plausibility and grounded consequences (Vezhnevets et al., 2023). This separation between expressive agent behavior and environment-mediated consequences is an important methodological precedent for executable language-agent interaction.

Other environments emphasize richer strategic structure. ADASOCIETY combines expanding state and action spaces with explicit, alterable social structures that affect multi-agent tasks and access to information and rewards (Huang et al., 2024). CIVREALM presents an imperfect-information general-sum environment with changing players and action spaces and explicit demands for diplomacy and negotiation (Qi et al., 2024). GOVSIM studies sustainable cooperation by requiring societies of LLM agents to manage shared resources over time (Piatti et al., 2024), while Zhou et al. (2026) manipulate agents' moral dispositions, observability, and communication bandwidth to study moral evolution in a simulated society. AGENTSOCIETY extends this line toward large-scale computational social science with realistic societal environments and scalable LLM-agent simulation (Zhang et al., 2025; Piao et al., 2026). These platforms emphasize open-ended social dynamics or social-science experimentation; PRAXISCENE uses an executable environment as a standardized measurement substrate with frozen states, legal transitions, information opportunities, and terminal outcomes.

The embodied-agent literature provides a complementary construction precedent. PARTNR uses a semi-automated generation pipeline with simulation-in-the-loop grounding and verification to construct 100,000 human–robot collaborative tasks (Chang et al., 2025). Although those tasks have explicit objectives and simulator-defined success functions, the methodological lesson is directly relevant: language-generated benchmark items become reliable evaluation units only after they are grounded and verified against an executable substrate.

Human behavior as reference rather than gold standard. Human data play different roles across social-agent research. SOTOPIA includes human–agent role-play, and SOCIALEVAL explicitly compares model and human social-intelligence behavior (Zhou et al., 2024; 2025). A separate literature treats fidelity to human behavior as the main target. SIMBENCH standardizes 20 behavioral

datasets, spanning tasks from moral decisions to economic choice, to measure population-level simulation fidelity and reports substantial difficulty in reproducing heterogeneous demographic groups (Hu et al., 2026). At the trajectory level, Lu et al. (2026) use 31,865 real online-shopping sessions to evaluate next-action generation and show a large gap between qualitative believability and accurate step-by-step behavioral reproduction.

PRAXISCENE uses human data differently. Human participants traverse the same executable stories and free-form interface as agents, but their behavior is not treated as a normative gold standard. Human trajectories instead provide a matched behavioral reference for interaction depth, information acquisition, policy revision, and realized Strategy/Value distributions. This makes human–agent differences observable without assuming that the majority human trajectory is uniquely correct.

Benchmark and evaluator validity. Social-agent evaluation also depends on the validity of its measurement instrument. SOTOPIA-π shows that LLM-based social evaluators can overestimate agents optimized for the same social-interaction setting (Wang et al., 2024b), while LIVECULTUREBENCH explicitly quantifies verifier uncertainty and studies when human oversight is needed (Pham et al., 2026). Validity problems can also enter during benchmark construction. Mousavi et al. (2026) audit SOCIALIQA, FauxPas-EAI, and ToMi and identify structural, semantic, and pragmatic problems in benchmark items and scoring procedures; their analyses further show sensitivity to context availability, wording, and output format.

These findings motivate two design choices in PRAXISCENE. First, runtime measures such as execution and information acquisition are grounded in explicit environment state rather than a single holistic LLM-judge score. Second, annotation-dependent analyses and Goal/Information/Value interventions are restricted to stories that satisfy their corresponding validation and eligibility criteria. Properties without a defensible directional criterion are reported as behavioral profiles rather than collapsed into an aggregate social utility.

Situation and value structure as coverage priors. The benchmark's coverage design draws on empirical descriptions of social situations rather than scenario categories defined only by generation prompts. Bhatia et al. (2026) analyze more than 20,000 descriptions of dyadic social interactions from participant-generated stories, published fiction, blogs, and autobiographical narratives, and derive 36 data-driven situational classes, further relating them to higher-level properties such as conflict, power, and duty. PRAXISCENE uses these classes as a coverage prior over social contexts, not as an exhaustive ontology of social life.

Value structure is informed by Schwartz's refined theory of basic individual values, which distinguishes 19 values arranged along a circular motivational continuum with systematic relations of compatibility and conflict (Schwartz et al., 2012). In PRAXISCENE, these values provide a construction and analysis vocabulary. Construction-side values do not determine which value an ending must realize, and realized values are not given a universal competence ordering. The Value intervention instead asks a narrower question: when a story can express a target value, how does explicit conditioning shift the policy realized by the same system?

Comparison with closely related benchmarks. Table 2 summarizes the benchmarks that most directly motivate PRAXISCENE's design. Rather than treating "interactive" as a binary attribute, the comparison asks which parts of the decision problem are supplied by the evaluator and which must be resolved through the agent's own interaction.

Positioning summary. Across these literatures, the unit of social-agent evaluation has expanded from judgments over supplied situations to sequential decisions, multi-turn social interaction, executable action, active information management, and persistent social worlds. PRAXISCENE does not claim that these components are absent from prior work. Its distinction is the allocation of decision responsibility within the episode: the canonical condition withholds a task-level social objective, relevant evidence may need to be acquired through interaction, and history-dependent stories allow earlier behavior to change later state and opportunities. Goal specification and information access are then manipulated on matched stories rather than treated as fixed benchmark properties.

This distinction also determines what can be scored directionally. Execution, information acquisition, and locally identifiable policy correspondence support criterion-referenced capability mea-

Table 2: Selected sequential and interactive social-agent benchmarks most closely related to PRAXISCENE. The comparison emphasizes the allocation of decision responsibility: whether an objective is supplied, how interaction and information are structured, and what the evaluation measures.

Benchmark	Episode objective	Interaction / state	Information structure	Evaluation / reference
MACHIAVELLI (Pan et al., 2023)	Game rewards / objectives	Branching CYOA game trajectories	Scenario information revealed through game progression	Reward plus power-seeking, disutility, and ethical-violation annotations
SOTOPIA (Zhou et al., 2024)	Social goals supplied	Open-ended multi-turn role-play	Role context and information emerging through dialogue	SOTOPIA-Eval; human–agent role-play
STSS (Wang et al., 2024a)	Social tasks supplied	Multi-agent sandbox actions with simulator state	Simulator observations and task state	Objective action-level goal achievement
AGENTSENSE (Mou et al., 2025)	Social goals supplied	Script-derived multi-turn scenarios	Implicit and private-information reasoning within scenarios	Goal completion and implicit reasoning
SOCIALEVAL (Zhou et al., 2025)	Social-goal achievement evaluated	Structured bilingual narrative world trees	Narrative state unfolds along script branches	Outcome goal plus interpersonal process; human comparison
LIVECULTURE BENCH (Pham et al., 2026)	Daily goal supplied	Dynamic multi-agent town	Dynamic social and cultural context	Task progress, norm adherence, and verifier uncertainty
SOTOPIA-TOM (YS et al., 2026)	Group and agent objectives supplied	Multi-party public/private interaction	Partitioned private knowledge and sharing constraints	Information sharing/seeking, coordination, efficiency, and privacy
PRAXISCENE	**No task-level goal canonically; matched Given-Goal control**	**Free-form actions grounded into executable branching stories**	**Stepwise information opportunities; Provided-Information control; history-dependent CR state**	**Criterion-identifiable capabilities plus Strategy/Value profiles; matched human trajectories**

sures. Interaction depth, contingent policy dynamics, and realized Strategy/Value distributions instead characterize how a policy is expressed without assuming that one socially plausible trajectory is universally preferable. In this sense, PRAXISCENE treats open-goal interaction as a controlled evaluation condition rather than as a synonym for unconstrained social simulation.

B PRAXISCENE ENVIRONMENT: IMPLEMENTATION DETAILS

This appendix provides the technical counterpart to the environment overview in Sec. 3. We describe PRAXISCENE as an executable social-interaction environment rather than as a collection of prompts or a benchmark server. The environment spans two stages. At authoring time, social situations are converted into structured story specifications and then compiled into executable story configurations. At runtime, participant inputs are resolved into registered actions or semantic events, event conditions are evaluated against explicit state and interaction history, NPC behavior responds to the evolving story, and the resulting execution is recorded as a trajectory.

The distinction between the *environment* and the *benchmark* is important. This appendix specifies the reusable mechanisms by which stories are generated, represented, executed, observed, and logged. Appendix C specifies how those mechanisms are instantiated into the frozen PRAXISCENE benchmark, including the UR/CR taxonomy, corpus composition, benchmark-side filtering and annotation, and Strategy/Value labels. In particular, the runtime determines which configured semantic events and endings actually occur; benchmark labels are attached later and do not control the execution itself.

B.1 ENVIRONMENT ARCHITECTURE AND DESIGN CONTRACT

At a high level, PRAXISCENE connects an authoring pipeline to an event-driven runtime. A story generator produces a structured social scenario, which is compiled into a story configuration containing the entities, semantic events, conditions, state changes, NPC-side information, and terminal events needed by the runtime. During interaction, participant inputs enter through a shared environment interface. Free-form text is grounded into executable semantics before it can affect the story; structured actions are dispatched through the action interface directly. Runtime events then mediate information revelation, state change, NPC-side updates, and ending resolution. A trajectory recorder preserves the resulting execution evidence for later benchmark analysis.

Three design contracts are central to the environment. First, free-form language does not directly mutate world state: it must first be resolved into a registered semantic event or an executable action. Second, story consequences are mediated by explicit events, conditions, transitions, and terminal objects rather than inferred post hoc from narrative text. Third, runtime execution is kept separate from benchmark interpretation. In particular, the environment resolves the terminal `End` event reached by a trajectory, whereas Strategy and Value labels are attached later by the benchmark layer.

B.2 Story Generation and Compilation

Structured Story Specification and Generation A machine-readable story specification fixes the initial situation, the participants and their relations, the information that is visible and the information that is latent, the semantic events available in the situation, the state-update rules, the conditions that guard later transitions, and the terminal outcomes. The generator must produce a specification that satisfies this schema before any graph is built, and the specification is what the graph is compiled from. Construction controls are authoring inputs, not benchmark ground truth: the value assigned during construction guides generation and is not a claim about what the resulting interaction realizes. Generation is subject to structural constraints on branching, cast size, the opening state, and the set of reachable endings, so that a candidate is rejected before it can enter the benchmark if it violates the authoring contract. The field definitions, the required and optional fields, and the serialization format are part of the released implementation (`https://anonymous.4open.science/r/PraxiScene-FB5F/`).

Executable Story Graph and Compilation An executable story is a directed graph whose nodes are scenes and whose edges are player choices. A scene node carries the visible situation, the entities present, and the choices available from it; a choice node carries the player- facing wording, the entities it addresses, and the event it resolves to; and an ending node carries the terminal description. Edges are the routing from a choice to its destination, and each non-root node has exactly one incoming edge: branches cannot merge, endings cannot be shared, and cycles cannot occur. Every interactive scene additionally reserves one final slot for an engine-owned exit, so that a participant who does not choose a story-authored option can still leave the scene. Information, NPC tasks, and state updates are attached to nodes rather than encoded in the prose, which is what makes the graph executable rather than descriptive: an event fires because its trigger and condition are satisfied, not because a narrative passage implies it. The graph is a bounded program over the authoring space, not a symbolic model of the social world, and no claim is made that it represents all of a situation's social structure. The node and edge schemas and the routing invariants are part of the released implementation (`https://anonymous.4open.science/r/PraxiScene-FB5F/`).

Structural Validation Checks that are purely structural are decided by deterministic code, not by a language model. The configuration linter builds a unified event map from the top-level events, the NPC transitions, the world transitions, and the system and debug triggers, then runs a registry of static diagnostic handlers over it and reports diagnostics by rule code. The registry covers thirteen rules: broken references from triggers and condition trees to undefined events, duplicate event names, dangling events that no trigger or condition names, condition operators applied to the wrong number of operands, conditions that reference a signal which no triggerable object publishes, transitions that apply no state change, self-dependent task transitions, transitions that assign a task value the task does not declare, item references that no configured item resolves, region centers that fall outside the scene bounds, mismatched approach-trigger semantics, and information payloads whose length violates the exchange contract. A cycle in the triggered-object expansion is also rejected. The linter emits errors and warnings and refuses a configuration that carries an error; the same code runs before admission and as a static check on the released artifacts. The exact rule codes, messages, and severities are part of the released implementation (`https://anonymous.4open.science/r/PraxiScene-FB5F/`).

B.3 Conditional Execution Mechanisms

Runtime State and Interaction History Execution carries a state x_t, an interaction history h_t, and an observation o_t. The state is the world state together with the NPC state: world state holds the item set and the scene identity, and each NPC carries its character description, behavioral rules,

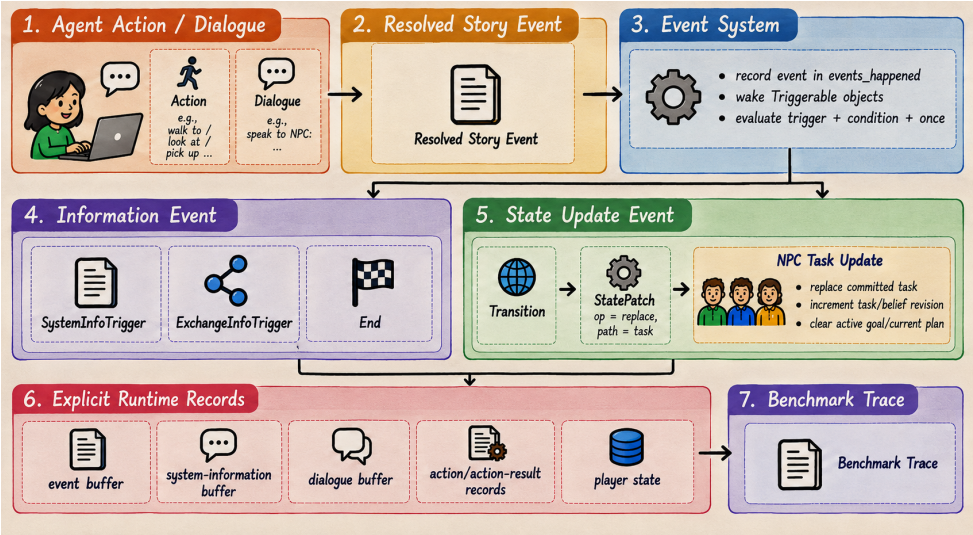

Figure 10: **Information revelation and state-update flow.** After an agent action or dialogue turn is resolved into a story event, the event system records the event, evaluates registered triggerable objects, and produces either information-level consequences or explicit state updates. Information-level records include revealed system information, detected information exchange, and terminal events. State-changing consequences are represented through transitions and, in the CR stories, NPC-facing task patches. The resulting runtime artifacts are exposed through player buffers and copied into the benchmark trace.

available actions, and its own transition set. History is the ordered record of events that have happened; every executed event is recorded, appended to the timeline, and published to the topic subscribers registered for it. The observation o_t is not the state itself but the subset of it that the participant interface serializes, and it is constructed by the same procedure for a human participant and for an agent. We give the schema, the observation construction, and the trajectory record in Appendix B.4 and in the released repository (`https://anonymous.4open.science/r/PraxiScene-FB5F/`); the remainder of this appendix describes the mechanism rather than reproducing the field definitions.

Guarded Transitions and Reachability This subsection should explain the generic mechanism by which previously established state, information, events, or history can enable or disable later transitions. A compact formalization such as $\text{enabled}_t(e) = C_e(x_t, h_t)$ may be used if it matches the implementation. The purpose here is only to describe runtime capability; the benchmark-side UR/CR taxonomy is defined later in Appendix C.

Information Revelation and State Updates After an agent action or dialogue turn has been resolved into a story event, its effects are mediated by the runtime event system. The details of input resolution are described separately in Appendix B.4.1; here we focus on how a resolved event produces explicit information and state changes. When an event is published, the runtime records it in `events_happened`, appends it to the event timeline, broadcasts it to the relevant topics, and wakes the `Triggerable` objects registered on that event. Each triggered object then checks whether its condition over the event history is satisfied and whether a trigger marked `once=True` has already fired. Figure 10 summarizes this runtime path.

Story effects are represented in two main forms. The first form is information revelation through explicit event objects. A `SystemInfoTrigger` exposes configured information to the player, an `ExchangeInfoTrigger` records an information transfer between participants, and an `End` event records story termination. These effects therefore appear as concrete runtime events rather than being inferred from narrative text.

The second form is a state-changing transition. A `Transition` is activated only after its triggering event and condition are satisfied, and its result is expressed as one or more `StatePatch` objects. Each patch specifies an operation, a target path, and a value. In the current CR benchmark stories, the principal NPC-facing update uses `op=replace` with `path=task`. Applying this patch replaces the NPC's committed task, increments the task revision and the knowledge-base belief revision, and clears the active goal and current plan so that subsequent NPC behavior is generated from the updated task. In contrast, the inspected UR story exposes its consequences primarily through information and ending events and does not rely on NPC task patches.

Both information events and state-changing consequences are reflected in explicit runtime records. The environment maintains separate buffers for engine events, revealed system information, dialogue, resolved actions, action results, and the current player state. The benchmark server copies the event, system-information, dialogue, action, and action-result buffers into the trajectory record. This allows later evaluation to identify which events occurred, what information was revealed, and which observable state changes were produced without reconstructing them from an unconstrained narrative summary. The detailed trajectory schema is described separately in Appendix B.5.

Terminal Conditions and Ending Resolution Story termination is represented by explicit `End` events in the story configuration. An `End` is a `Triggerable` object with a `trigger`, an optional condition, a `name`, and a `description`. Thus, a story does not terminate because the agent or an NPC produces an unconstrained textual ending; it terminates only when a configured terminal event is activated by the runtime event system.

Ending activation follows the same guarded-execution mechanism as other triggerable events. The event system waits for the event named in the `trigger` field, evaluates the ending condition if one is provided, and respects the `once=True` setting. In UR-style stories, terminal events are often triggered directly by semantic player-input or information-exchange events. In CR-style stories, terminal events may additionally require that branch-specific history events have already occurred, for example through an `events_happened` condition that checks whether the relevant choice-result event is present in the runtime event history.

When an `End` event fires, the event system records it as the run's stop reason. If multiple terminal events become eligible in the same interaction, the runtime uses a first-ending-wins rule: the first `End` stored in the stop reason is retained, and later terminal events are ignored or suppressed rather than replacing it. The environment also routes the ending event to the player-facing event buffer and then shuts down the active environment managers. The engine result serializes the ending as its `stop_reason`, preserving the terminal event's `name` and `description` for downstream consumers.

Runtime ending resolution is separate from benchmark annotation. The environment determines only which `End` event was reached. Strategy and value labels are attached later by the benchmark layer, which maps the resolved ending name to precomputed ending-to-strategy and ending-to-value annotations. If an engine stops without a real `End` event, the benchmark layer uses the synthetic `unknown_ending` outcome and treats the story as having no valid ending, rather than counting it as a successful terminal state.

B.4 Shared Participant Interface and Action Grounding

Shared execution semantics. Human participants and automated agents use different clients but the same underlying server-authoritative story state, transition rules, and ending resolution. Agent runs use a session-scoped tool interface; human runs use the browser client. This client distinction is intended to change presentation, not the semantics of the executable story.

Both clients observe the same world: the regions the player can be in, the NPCs present, and the items present, together with the action results and the events published to the participant. Both select from the same player-facing actions, and a free-form utterance on either client is resolved through the same grounding procedure onto that action set. What differs is presentation: the browser client renders the situation and the available actions visually, whereas the agent client receives a serialized observation and returns tool calls. No player-facing action, observation field, or ending resolution is exclusive to either client, so the two clients are equivalent with respect to what they can observe and

what they can do, and trajectories from matched human and agent runs are compared on the same measurement surface.

Free-Form Agent Interface Agents interact with the environment through a player-facing MCP interface, rather than by directly calling internal engine methods. The interface exposes the same interaction loop used by the benchmark runner: the agent first observes the current player-visible state, then either sends free-form natural language input or invokes one of the structured player actions. The environment serializes these calls with a tool lock, so each agent turn is processed against a single ordered stream of flushed observation buffers and action results.

The observation is obtained through `get_world_state`. Internally, this tool combines the current player state with the player-facing buffers maintained by the environment. The returned text may include the current input-resolution match, action results, revealed story context, local observation, recent dialogue, engine events, and terminal-state information when the story has ended.

Observation template.

```
Input match:
<latest player-input grounding result, if any>

Action results:
<resolved action results since the previous observation>

Story context:
<new system information revealed to the player>

Local observation:
Location: <current player location>
Inventory: <player inventory>
Nearby agents: <visible nearby agents>
Nearby items: <visible nearby items>
Regions: <known nearby regions>

Recent dialogue:
<dialogue turns visible to the player>

Events:
<engine events visible through the player buffer>

STORY ENDED.
Ending: <ending name>
Labels: <ending labels>
Description: <ending description>
```

The fields are omitted when the corresponding buffer is empty. This means the agent does not receive a hidden global world state; it receives only the player-visible state and the information that has been explicitly exposed through the runtime interface.

Agent-facing tools. The first tool is a pure observation call. It takes no arguments and returns the current player-facing world text.

```
Tool: get_world_state

Input schema:
{
  "type": "object",
  "properties": {},
  "required": []
}
```

The second tool accepts free-form player input. This is the main natural-language interface used by agents. The environment-side resolver later decides whether the input corresponds to an executable player action or to a configured player-input trigger. For speech, the interface instructs the

agent to include both the target and the spoken content, for example using the form `speak to <character>: <content>`.

```
Tool: send_player_input

Input schema:
{
  "type": "object",
  "properties": {
    "text": {
      "type": "string",
    }
  },
  "required": ["text"]
}
```

The third tool bypasses the free-form resolver and asks the environment to execute a structured player action directly. The action name must be one of the registered public action names, and the parameter object supplies the arguments needed by that action.

```
Tool: send_player_action

Input schema:
{
  "type": "object",
  "properties": {
    "action": {
      "type": "string",
      "enum": [
        "walk_to",
        "look_at",
        "pick_up",
        "follow",
        "speak",
        "read"
      ],
      "description": "The action to perform."
    },
    "parameters": {
      "type": "object",
      "description": "Key-value parameters for the action.",
      "properties": {
        "location": {
          "type": "string",
          "description": "Target location or region."
        },
        "target": {
          "type": "string",
          "description": "Target character or entity."
        },
        "content": {
          "type": "string",
          "description": "Utterance content for speech."
        },
        "object_name": {
          "type": "string",
          "description": "Target object name."
        },
        "time": {
          "type": "number",
          "description": "Optional duration."
        }
      }
```

```
    }
  },
  "required": ["action"]
}
```

The structured-action interface is intentionally narrow. If the agent does not know the exact action name or object name, it is expected to use `send_player_input` instead, allowing the environment-side resolver to ground the utterance using the current local context.

Turn and budget conventions. Each benchmark episode is executed as an ordered sequence of agent-environment turns. On a turn, the agent may call `get_world_state` to observe the latest player-visible state and then submit either a free-form input or a structured action. The server waits for the corresponding environment-side processing to finish, collects the newly visible buffers, and returns an updated world-state text. These buffers are flush-on-read, so the serialized MCP tool loop also defines the observation boundary used by the benchmark trajectory.

```
Runtime interaction constants:
POLL_WAIT = 1 second
SPEAK_REPLY_TIMEOUT = 10 seconds
DEFAULT_REQUEST_TIMEOUT = 10.0 seconds
PLAYER_REQUEST_TIMEOUT = 60.0 seconds
ACTION_RESULT_TIMEOUT = 15.0 seconds
```

These constants bound how long the MCP layer waits for request completion, action results, and NPC replies. They do not create hidden story progress or synthetic success states. If an action or free-form input cannot be resolved within the interface semantics, the failure is returned through the public result channel and becomes part of the observable interaction trace.

Action Grounding and Resolution Free-form player input is grounded by the environment-side player-input resolver before it can affect the story. The resolver receives the raw input string, the current local world context, and the configured player-input triggers. It does not execute the text directly. Instead, it first returns a structured resolution result, which the command layer then interprets as either a player action or a player-input event. In the current implementation, this resolver is a structured-output language-model call with temperature zero. The default resolver model is `gpt-4.1`.

The resolver distinguishes two resolution types. If the input matches a configured player-input trigger, `resolution_type` is `event` and the result identifies the matched trigger name. If the input describes an executable environment action, `resolution_type` is `action` and the result contains the resolved action name and parameter dictionary. The resolver is explicitly instructed to check configured player-input triggers before executable actions. This ordering matters because many benchmark choices are represented as semantic story events rather than low-level physical operations. When such a trigger is matched, the runtime publishes the corresponding event instead of treating the same text as an ordinary action.

Grounding is constrained by the current local world context. The resolver prompt includes nearby agents, nearby items, and known regions. Entity and region parameters must refer to names present in this context. Speech grounding also uses a context-dependent rule: if exactly one agent is nearby, an unnamed speech input may be matched to that agent; if multiple agents are nearby, the input must explicitly name the target and include the literal words to say; if no agents are nearby, the input cannot be resolved as speech.

When the resolved type is `event`, the command layer looks up the named trigger in the player-input trigger table and publishes that event to the runtime event system. When the resolved type is `action`, the command layer dispatches the resolved action through the environment action manager and stores the resulting action task name in the returned `action_id`. This identifier is later used by the MCP layer to associate the eventual action result with the input that created it.

Structured actions submitted through `send_player_action` bypass the natural-language resolver. The action name is looked up in the environment action registry, and the supplied parameters are validated against the corresponding action schema before execution. Thus, free-form inputs are

grounded through the resolver, while structured action calls are grounded by registry lookup and schema validation. The current grounding procedure does not use a numeric confidence threshold. Ambiguity, missing entities, or mismatched player-input triggers are represented through the structured success flag and are handled by the interface-failure rules in Appendix B.4.3.

Resolver configuration. The current player-input resolver uses the following configuration.

```
PlayerInputEvaluatorConfig:
{
  "model": "gpt-4.1",
  "temperature": 0.0
}
```

Resolver system prompt.

```
You classify player input into actions or triggers.

Your reason must match your output:
- if reason says no match, set success=false.

Do not assign agents or actions unless the input explicitly names them.

A trigger names a specific scenario. Match only that same scenario:
- the same action
- the same reason
- the same outcome
- the same timing

Do not match a similar topic.
```

Resolver user prompt template.

```
You are resolving player input, in Chinese or English, into either:
- an available action
- a matching event trigger

Player Input:
"{input_content}"

World Context:
{world_context}

Available Actions:
Use these exact param names regardless of language.

- walk_to:
  location: entity or region name

- follow:
  target: agent name

- speak:
  target: agent name
  content: actual speech text, extracted after the target

- pick_up:
  object_name: item name

- place:
  object_name: item name

- drop:
  object_name: item name

- look_at:
  location: entity name to look at

- sit_down:
  no params

- sleep_on:
  no params

Available Event Triggers:
{input_triggers}
```

```
{speak_rule}

Instructions:
- Handle input in any language, including mixed Chinese and English.

- Check event triggers before actions. If the input matches an event
  trigger, resolve to that trigger. Only resolve to an action when no
  trigger matches.

- Reason and output fields MUST agree. Set success=true only if the
  reason concludes a match. If the reason says no match, success MUST
  be false.

- Entity and region names for parameters MUST exist in World Context.
  If not found, return success=false.

- Region names may be hierarchical. The last segment after / is the
  short name.

- walk_to requires a destination that the input names explicitly.
  If the input describes leaving without naming a destination, it is
  NOT walk_to.

- Match a trigger ONLY if the input states the same specific scenario:
  the same action, reason, resulting outcome, and timing.

- If the input's reason, outcome, or timing differs from the trigger,
  it is a different scenario, even if the act looks similar.

- Do not match on a shared keyword alone. If a match needs inference,
  set success=false. Do not substitute or generalize.

- Content for speak must be literal words, not an abstract topic.

Return a structured result with:
- resolution_type:
  "action" if the input maps to an action
  "event" if the input maps to a trigger

- action_name:
  the matched action name

- action_params:
  the params for the matched action

- trigger_name:
  the matched trigger name

- success:
  true if a match was found, false otherwise

- reason:
  a brief explanation of the decision
```

World-context template. At resolution time, the resolver receives a text context derived from the player's current local region.

```
Nearby Agents:
  - <agent name> (<script name, if present>)
  ...

Nearby Items:
  - <item name> (<script name, if present>):
    <look_at_description, if present>
  ...

Regions:
  - <region name>
  ...
```

Speech grounding rules. The resolver inserts one of the following speech rules depending on the number of nearby agents.

```
If exactly one agent is nearby:

SPEAK RULE:
There is exactly 1 agent nearby (<agent>).
You MAY match input as speak to that agent without the agent name
in the text.

If multiple agents are nearby:

CRITICAL SPEAK RULE:
<n> agent(s) nearby: <agent names>.
Speak requires:
- a communication verb
- an explicitly named agent
- exact words to say, not a topic

If agent is not named, it is not speak.
If content is a topic rather than words, it is not speak.

If no agents are nearby:

SPEAK RULE:
No agents nearby. NEVER match speak.
```

Structured resolver output. The internal structured output from the language model has the following schema.

```
PlayerInputResolutionResult:
{
  "reason": "<brief reason for the resolution decision>",
  "resolution_type": "action" | "event",
  "action_name": "<matched action name>",
  "action_params": {
    "location": "<target location, for walk_to or look_at>",
    "target": "<target agent, for speak or follow>",
    "content": "<literal speech content, for speak>",
    "object_name": "<item name, for item actions>",
    "time": "<duration in seconds>"
  },
  "trigger_name": "<matched trigger name>",
  "success": true | false
}
```

Public turn result. The public result returned by `send_player_input` is represented as `PlayerInputResult`. This is the object exposed to the MCP layer after the input has been grounded and, when applicable, dispatched.

```
PlayerInputResult:
{
  "content": "<original player input>",
  "success": true | false,
  "resolution_type": "action" | "event",
  "resolution_content": {
    "...": "resolved action parameters or trigger name"
  },
  "action_id": "<engine action id, empty unless action dispatched>"
}
```

Interface Failure Handling Interface failures are represented explicitly rather than repaired silently. For free-form input, grounding failure is encoded in the returned `PlayerInputResult`. If the resolver cannot match the input to either a configured player-input trigger or an executable action, it returns `success=false` with empty resolution content. The command layer then returns this structured resolution to the caller without publishing an event and without dispatching an action. Exceptions raised during the resolver call are handled in the same way: the environment logs the error and returns an unsuccessful `PlayerInputResult`, rather than substituting a default action.

Structured action calls fail at a different boundary. A `send_player_action` request bypasses the language resolver, so the provided action name and parameter dictionary must be accepted by the action registry and the corresponding action schema. If the action name is unavailable or the parameters do not validate, command execution fails and the benchmark endpoint reports an error instead of inventing a replacement action. If the action is successfully dispatched but fails during

execution, the failure is reported as an `ActionResult` with a failure message. These action results are exposed through the player action-result buffer and are copied into the trajectory log.

The current interface does not run a separate semantic repair loop for failed agent inputs. It does not ask the model to rewrite an ambiguous command, does not infer missing entities that are absent from the world context, and does not promote an unmatched free-form input into a story event. Ambiguity, missing grounding targets, mismatched trigger semantics, invalid action names, invalid parameters, and action-execution failures remain observable as unsuccessful resolutions, command errors, or failed action results.

The MCP layer bounds waiting time for every interface operation. Ordinary observation reads use the default short request timeout, while player-turn requests use the longer player-request timeout because they may involve input resolution or action execution. If a free-form input resolves to an action, the MCP server polls the player-info endpoint for the resulting action id for a bounded window. When an action id is available, the poller waits for that exact id; otherwise it falls back to the newest same-named action observed on the first buffer read. Speech turns use a separate bounded polling window so that an NPC reply can be included in the returned observation when it arrives. If a polling deadline expires, the MCP server returns the world state and any runtime buffers already collected rather than waiting indefinitely or reporting a synthetic success.

Interface failure does not by itself terminate the story. A run reaches a valid terminal state only through the runtime `End` mechanism described in Appendix B.2.3. If an engine stops without a real `End` event, the benchmark layer records the synthetic `unknown_ending` outcome and treats the story as having no valid ending. Thus, grounding failures, invalid actions, action timeouts, and missing action results are kept separate from story-ending resolution.

Failure-state summary. The implemented interface exposes the following failure states.

```
Free-form input cannot be grounded:

PlayerInputResult:
{
  "content": "<original player input>",
  "success": false,
  "resolution_type": "event",
  "resolution_content": {},
  "action_id": ""
}


Structured action cannot be validated or executed as a command:

CommandResult:
{
  "command_id": "<command id>",
  "success": false,
  "message": "<command or validation error>",
  "payload": null
}


Dispatched action fails during execution:

ActionResult:
{
  "action_id": "<engine action id>",
  "action_name": "<action name>",
  "success": false,
  "message": "<execution failure message>",
  "error_code": "<optional error code>",
  "error_animation": "<optional error animation>"
}


Story stops without a real End event:

{
  "name": "unknown_ending",
  "description": "Story ended."
}
```

Timeout conventions. The current MCP runtime uses the following request and polling budgets.

```
DEFAULT_REQUEST_TIMEOUT = 10.0 seconds
PLAYER_REQUEST_TIMEOUT  = 60.0 seconds
ACTION_RESULT_TIMEOUT   = 15.0 seconds
SPEAK_REPLY_TIMEOUT     = 10.0 seconds
POLL_WAIT               = 1.0 second
```

B.5 Trajectory Logging and Replay

Unified Trajectory Log This subsection should define what is recorded for every interaction step: observations, raw agent outputs, grounded actions, revealed information, state/flag changes, semantic events, and ending status. The log is the bridge between environment execution and benchmark scoring.

The environment-to-benchmark boundary is recorded as a story-level trajectory rather than as an unconstrained narrative summary. During execution, the MCP interface and the benchmark recorder collect both agent-side interaction data and engine-side runtime events. The raw trace remains the source of truth for agent- or provider-specific payloads, while a normalized story trace is derived from it for cross-agent analysis and scoring.

For agent runs, the normalized trace uses the `agent_trace_v1` schema. It records the story identity, engine identity, story structure, agent and model labels, terminal outcome, normalized agent action trace, engine event trace, and usage statistics. The action trace stores the observable sequence of agent-facing events: the initial player input prompt, model reasoning or text, MCP tool calls, tool inputs and outputs, tool errors if present, and usage records. The engine-event trace stores events emitted by the story runtime when they are available through the event collector, including semantic events such as information revelation, transitions, and ending events.

Normalized agent trace schema.

```
AgentTrace:
{
  "schema_version": "agent_trace_v1",
  "story_id": "<story id>",
  "engine_id": "<engine id>",
  "agent_kind": "<agent type>",
  "model_label": "<model label>",
  "structure": "UR" | "CR",
  "server": "<benchmark server>",
  "server_log_path": "<optional server log path>",
  "engine_log_path": "<optional engine log path>",
  "outcome": {
    "status": "done" | "failed",
    "stop_reason": "end" | "timeout" | "agent_exit" |
                   "engine_error" | "component_error" |
                   "cancelled" | "unknown",
    "ending_name": "<ending name, if stop_reason is end>",
    "failure_reason": "<failure reason, if any>"
  },
  "action_traces": [
    "<player_input | reasoning | tool_use | usage events>"
  ],
  "engine_events": [
    "<engine-side runtime events>"
  ],
  "usage": {
    "<story-level token usage>"
  }
}
```

Each normalized action-trace event is typed. A `player_input` event records the prompt or user text sent to the agent. A `reasoning` event records model text or reasoning emitted before or around tool calls. A `tool_use` event records one MCP benchmark tool call together with its input and returned state. A `usage` event records token-usage statistics for a model step.

Action-trace event schema.

```
TraceEvent =
  PlayerInput | Reasoning | ToolUse | Usage

PlayerInput:
{
  "type": "player_input",
  "timestamp_ms": "<optional timestamp>",
  "time": {"start": "<ms>", "end": "<ms>"},
  "content": "<prompt or user text>"
}

Reasoning:
{
  "type": "reasoning",
  "timestamp_ms": "<optional timestamp>",
  "time": {"start": "<ms>", "end": "<ms>"},
  "content": "<model text or reasoning>"
}

ToolUse:
{
  "type": "tool_use",
  "timestamp_ms": "<optional timestamp>",
  "time": {"start": "<ms>", "end": "<ms>"},
  "mcp_tool": "<benchmark tool name>",
  "input": "<tool input>",
  "state": {
    "status": "completed" | "error" | "pending" | "unknown",
    "input": "<normalized tool input>",
    "output": "<tool output>",
    "error": "<tool error, if any>",
    "metadata": {},
    "title": "<optional title>",
    "time": {"start": "<ms>", "end": "<ms>"}
  }
}

Usage:
{
  "type": "usage",
  "step_index": "<optional model-step index>",
  "prompt_tokens": 0,
  "completion_tokens": 0,
  "total_tokens": 0,
  "reasoning_tokens": 0,
  "cached_input_tokens": 0,
  "cache_write_tokens": 0,
  "raw": {}
}
```

Engine-side events are stored separately from agent-side tool events. They are the runtime events published by the story engine and copied into the trace when available. The normalized engine-event record keeps the event type, event name, and timestamp, while allowing event-specific fields from the underlying runtime payload. This accommodates different event classes such as `system_info`, `exchange_info`, `transition`, and `end` without forcing them into a single fixed payload shape.

Engine-event schema.

```
EngineEvent:
{
  "event_type": "<runtime event type>",
  "event_name": "<runtime event name>",
  "timestamp_ms": "<optional publish timestamp>",

  "...": "<event-specific payload fields>"
}
```

Human-facing runs use the same run-record layout but a human-specific raw trace schema. A human story trace records story identity, participant/session metadata, timing, terminal outcome, ending annotations shown to the participant, and an ordered event stream. The event stream includes observations, player inputs, structured player actions, dialogue, revealed system information, engine events, resolved actions, action results, and story-end records. These human records are validated against pydantic schemas with closed record-level fields, while engine-shaped payloads allow event-specific fields.

Human raw-trace event types.

```
HumanTraceEvent =
  story_start |
  player_input |
  player_action |
  dialogue |
  system_info |
  event |
  observation |
  resolved_action |
  action_result |
  story_end

HumanStoryTrace:
{
  "schema_version": "human_story_raw_trace_v1",
  "run_id": "<run id>",
  "benchmark_session_id": "<benchmark session id>",
  "agent_kind": "human",
  "subject_type": "human",
  "participant_id": "<participant id>",
  "story_id": "<story id>",
  "engine_id": "<engine id>",
  "structure": "UR" | "CR",
  "started_at": "<timestamp>",
  "ended_at": "<timestamp>",
  "duration_seconds": "<seconds>",
  "status": "done" | "failed" | "abandoned",
  "outcome": "success" | "timeout" | "no_ending" | "skipped",
  "failure_reason": "<failure reason, if any>",
  "ending": "<ending payload, if any>",
  "ending_value_ids": ["<value id>", "..."],
  "ending_strategy_ids": ["<strategy id>", "..."],
  "labels_shown": ["<displayed label>", "..."],
  "action_count": 0,
  "reveal_count": 0,
  "actions": ["<action events>"],
  "reveals": ["<reveal events>"],
  "events": ["<ordered event stream>"]
}
```

The trajectory log is deliberately separated from the benchmark's scoring yardsticks. The trace records what happened during execution: what the agent or human saw, what was output, which tools or actions were invoked, which runtime events occurred, what information was revealed, and how the story terminated. Metrics such as shortest valid path, ending strategy/value labels, and information inventories are computed later by combining the trace with the frozen story configuration and benchmark metadata. This separation prevents scores from depending on post hoc narrative reconstruction while also keeping story annotations outside the execution log itself.

C Benchmark Construction, Structure, and Annotation

This appendix specifies how the reusable environment is instantiated as the frozen PraxiScene benchmark. The release manifest, executable `StoryConfig`s, annotation sidecars, and versioned scorer are authoritative.

C.1 Construction Space and Story Families

Each candidate is generated from one social-situation record and one construction value; construction values are authoring controls, not Values an evaluated agent must realize. The frozen release reports its final construction grid, candidate and accepted counts, generator version, and the manifest mapping each retained story to its source run. Fig. 11 reports the scene composition of the retained corpus.

Unconditional-Reachability Stories UR is a structural rather than a difficulty distinction: a UR story may be multi-step, branching, and information-seeking, but no trajectory-established state change may gate a later executable event or ending. In the frozen UR configuration, `world.transitions` and `npc.transitions` are empty and all executable conditions are

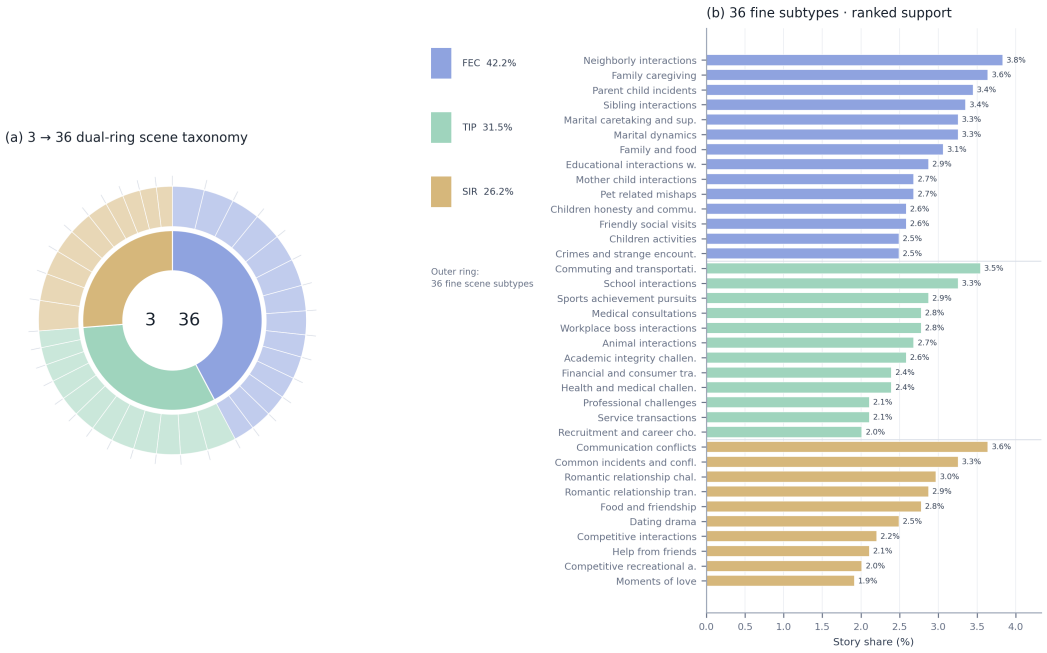

Figure 11: **Scene composition of the frozen benchmark.** (a) The three broad scene families (FEC, TIP, SIR) and the 36 fine-grained scene subtypes that instantiate them. (b) Story share of each fine subtype, ranked within family. Every retained story is generated from one catalog record of this taxonomy.

null; information may be hidden and revealed, provided that revelation does not unlock an event or ending.

The UR pipeline comprises free-form scenario ideation, structured conversion, missing-option augmentation, deterministic `BenchmarkStory` assembly, constrained `StoryConfig` completion, and bounded static repair; dynamic reachability testing and release admission are separate stages. Structuring removes unsupported claims about the player's internal state, separates information-seeking dialogue from terminal decisions, and normalizes identifiers; augmentation adds plausible omitted dialogue or terminal options without rewriting the scenario. Assembly instantiates regions, player, NPCs, and items, and a fourth model call generates the event and NPC-task components of an executable `StoryConfig`, with the remaining world and initial-state fields supplied deterministically. The resulting configuration satisfies the UR contract: each terminal decision maps to an ending reachable without a state-changing transition or a condition-gated event.

The frozen UR templates are reproduced verbatim below; no word, placeholder, order, or rule has been changed.

Frozen UR prompt templates (verbatim)

```
description: "Generate StoryBen benchmark story JSON before config generation."
prompts:
  step1_generate_story:
    - - system
      - |-
        You generate simple first-person interactive narrative scenarios. The player should
        ↪  understand the situation through observation and dialogue, then make an
        ↪  autonomous final choice.

        Always follow these rules:
        - Do not assign the player a personality, value preference, attitude, thoughts, or
        ↪  subjective feelings. Describe only objective facts and perceptible information.
        - Do not give the player an identity that depends on professional knowledge,
        ↪  professional credentials, or professional ethics.
        - Do not decide for the player, and do not imply that one option is the only
        ↪  correct answer.
```

```
        - Keep the story simple. Avoid unnecessary characters, backstory, hidden
        ↪  relationships, or twists.
        - NPC behavior and speech must fit their identity and characterization. If an NPC
        ↪  is a negative character, do not soften or beautify them.
        - The central conflict must be an objective fact that has already happened or is
        ↪  currently happening, and it must directly affect the player.
        - The player must be drawn into the conflict through an objective identity,
        ↪  relationship, responsibility, resource, limitation, or prior arrangement.
        - NPCs, items, and locations must all serve the current conflict; they must not be
        ↪  decorative filler.

    - - human
      - |-
        Generate a complete interactive narrative scenario from the information below:

        Main Value: {main_value}
        Main Value Definition: {main_value_definition}

        Target Sub-Value: {sub_value}
        Sub-Value Definition: {sub_value_definition}

        Scene Type: {scene_type}
        Scene Subtype: {scene_subtype}
        Scene Definition: {scene_definition}

        The main_value gives the broader value context.
        The sub_value is the actual target value this scenario must express. At least one
        ↪  possible decision must clearly express this sub_value.

        Interpret the social situation from scene_subtype and scene_definition, and design
        ↪  the story around the concrete meaning of the sub_value rather than
        ↪  free-associating from its name.

        Prefer an indoor location.

        [Opening Scene]
        Describe the objective facts and minimal necessary background the player can
        ↪  perceive just after entering the story.
        Every piece of information must have a clear source: what the player can currently
        ↪  see, hear, has been told, or already has arranged.
        Make clear what is currently happening and what kind of choice the player faces,
        ↪  but do not include the player's mental activity, emotion, value judgment, or
        ↪  action tendency.
        The conflict should already have happened or be happening, and it must objectively
        ↪  affect the player.
        Keep the opening short. Include only the essential setting, characters, conflict,
        ↪  and pressure. Do not dump lists, numbers, or item details here; leave those for
        ↪  the sandbox, items, or dialogue.

        [Sandbox]
        Describe what the player can observe about NPCs, items, or the environment before
        ↪  talking to NPCs.
        These observations should help the player understand the situation.
        NPCs, items, and locations should carry relationships, pressure, evidence,
        ↪  responsibility, or observable information tied to the current conflict.

        [Possible Dialogue]
        Describe possible one-turn exchanges between the player and NPCs. These exchanges
        ↪  should add event history, NPC positions, or decision-relevant information.
        Each entry must contain only one brief exchange, not an extended multi-turn
        ↪  conversation.
        Generate 3-4 one-turn exchanges that cover different information angles.
        Do not directly tell the player what they should choose.

        [Possible Decisions]
        Generate about 5-7 representative final choices.
        At least one decision must express the Target Sub-Value.
        Each decision must contain one atomic action only. Do not include sequential
        ↪  actions such as "first... then..." or "after that...".
        Do not split essentially identical actions only by amount, quantity, duration, or
        ↪  degree.
        The decisions must not form a moral ranking. They should represent conflicts among
        ↪  different values, interests, or emotions. Each choice should be defensible from
        ↪  some perspective and should require giving up something else.

        [NPC]
        Generate about 1-3 NPCs.
        Each NPC must have a concrete English name or form of address, such as "Ava", "Mr.
        ↪  Chen", or "Ms. Brooks". Do not refer to an NPC only by a role description such
        ↪  as "male passenger" or "the filming man".
```

```
        Use the same name for the same NPC consistently across the opening scene, sandbox,
        ↪  possible dialogue, possible decisions, and NPC section.
        For each NPC, define stable characterization based on the story: identity,
        ↪  motivation, interests, current attitude, and any position they may hide or
        ↪  insist on.
        The characterization must be strong enough to constrain how the NPC speaks and
        ↪  behaves later.
        If an NPC is a negative character, preserve that characterization and do not soften
        ↪  it.

        Output strictly in the following format, in English:

        Value
        {sub_value}

        Scene Subtype
        {scene_subtype}

        Opening Scene:
        ...

        sandbox:
        ...

        Possible Dialogue:
        ...

        Possible Decisions:
        ...

        NPC:
        ...

  step2_postprocess:
    - - system
      - |-
        You review and repair an already generated interactive narrative scenario, then
        ↪  convert it into the required JSON format.

        Preserve the original scenario as much as possible. Only fix actual problems;
        ↪  rewrite locally when necessary.
        Do not introduce new characters, items, or locations that do not exist in the
        ↪  original scenario.
        All story prose, scene descriptions, dialogue, decision text, NPC characterization,
        ↪  task text, and display names must be in English.
    - - human
      - |-
        Main Value: {main_value}
        Main Value Definition: {main_value_definition}

        Target Sub-Value: {sub_value}
        Sub-Value Definition: {sub_value_definition}

        Scene Type: {scene_type}
        Scene Subtype: {scene_subtype}
        Scene Definition: {scene_definition}

        The main_value gives the broader value context.
        The sub_value is the target value to check.

        Original scenario:
        {story}

        Review and repair the following areas:

        [scene_description]
        - Use first-person English to describe objective information the player has already
        ↪  seen, heard, been told, or arranged in the current scene.
        - Include the key objective pressure that supports the target value conflict.
        - Keep sentences grounded in external facts, observable states, and verifiable
        ↪  information. Do not write mental activity, emotion, value judgment, or action
        ↪  tendency.
        - Use only information explicitly present in the original scenario as the source.
        ↪  If details are missing, summarize instead of adding complex hidden
        ↪  relationships.
        - Keep only the core information the player must know on entry. Put concrete
        ↪  evidence, lists, quantities, and item states in
        ↪  `item_updates[].look_at_description`,
        ↪  `npc_updates[].initial_state.look_at_description`, or dialogue_options.
        - Do not pack every sandbox observation into scene_description.
```

```
[Possible Dialogue]
- Check whether each exchange is a single-turn exchange.
- Each exchange should supplement event history, NPC stance, or decision-relevant
↪  information rather than telling the player what to choose.
- Generate 3-4 dialogue_options. If there are too few, add information-seeking
↪  questions based on the original scenario.
- `dialogue_options[].id` must be short, semantic ascii_snake_case. Do not use pure
↪  numbering such as dialogue_1.

[Possible Decisions]
- At least one decision_option.player_action must clearly express the definition of
↪  Target {sub_value}.
- Each decision must be one atomic action only, without sequential actions.
- Do not split essentially identical actions only by amount, quantity, duration, or
↪  degree.
- Generate 5-7 decision_options. Each `label` is a short English choice label. Each
↪  `player_action` is a first-person English action beginning with "I".
- decision_options must be semantically distinct. If two options are nearly
↪  identical, merge them.
- `decision_options[].id` must be short, semantic ascii_snake_case. Do not use pure
↪  numbering such as decision_1.

[NPC]
- Check whether each NPC's characterization is consistent with the story and
↪  dialogue.
- If an NPC is a negative character, do not soften them.
- Do not change existing NPC, item, or region entity design while repairing.
- Generate one `npc_updates` entry for each important NPC in the original scenario.
- `npc_updates[].name` is a lowercase English internal ID using only a-z, 0-9, and
↪  underscores, such as `ava` or `teacher_wang`.
- `npc_updates[].script_name` is the English display name, such as `Ava` or `Ms.
↪  Brooks`.
- `npc_updates[].gender` is `male` or `female`, based on the NPC.
- `npc_updates[].figure` is `adult` or `child`, based on the NPC.
- `npc_updates[].height` is a numeric height in cm, estimated reasonably.
- `npc_updates[].region` is the NPC's initial location and must match one top-level
↪  `regions[].name`.
- `npc_updates[].character` should contain only stable identity, relationship, core
↪  motivation, long-term interest, and stable position. Do not include daily
↪  action strategy, dialogue answer scripts, itemized answerable facts, or long
↪  scene summaries.
- character must be written from the NPC's own prompt perspective. Use "you" to
↪  refer to the NPC themself. Do not use third-person self-description such as
↪  "he" or "she". When mentioning another NPC, cite their `npc.name` in backticks.
- `npc_updates[].rule` must be an English string array with 3-6 entries. Include
↪  stable speaking boundaries, prohibitions, value/interest constraints, and
↪  characterization limits. Do not include exact answers for dialogue_options.
- `npc_updates[].initial_state.look_at_description` must describe what the player
↪  objectively observes when looking at the NPC, using a third-person observer
↪  perspective.
- Do not generate `npc_updates[].initial_state.task`; the config stage will
↪  generate NPC daily tasks from the final dialogue_options.
- Each item in `npc_updates[].initial_state.properties` must contain `name` and
↪  `description`.
- property descriptions also use "you" to refer to the NPC themself. Do not use
↪  third-person self-description. When mentioning another NPC, cite their
↪  `npc.name` in backticks.
- Each NPC must have at least 1-2 properties describing current physical,
↪  psychological, social, or stable stance states. Do not put dialogue_options
↪  answers into properties.

[Item]
- Generate one `item_updates` entry for each important item in the original
↪  scenario.
- `item_updates[].name` is a lowercase English internal ID using only a-z, 0-9, and
↪  underscores, such as `phone` or `booking_form`.
- `item_updates[].script_name` is the English display name, such as `phone` or
↪  `registration form`.
- `item_updates[].region` is the item's initial location and must match one
↪  top-level `regions[].name`.
- `item_updates[].look_at_description` must be concrete observable English that
↪  helps the player understand the current conflict.

[Locations]
- Generate 1-3 `regions` for the scenario locations.
- `regions[].name` is a lowercase English internal ID using only a-z, 0-9, and
↪  underscores, such as `living_room` or `police_station`.
- `regions[].script_name` is the English display name, such as `living room` or
↪  `police station`.
- The first region must be where the player starts.
```

```
Fix problems directly while preserving the original content as much as possible.
Output only the specified JSON format.
Natural-language content in JSON must be English; technical identifiers such as
↪  `id` and `name` must be English ascii_snake_case.

JSON format:
{{
  "title": "English title",
  "scene_description": "120-180 words or fewer, in the player's first-person
  ↪  objective entry perspective",
  "regions": [
    {{
      "name": "ascii_region_name",
      "script_name": "English location display name"
    }}
  ],
  "npc_updates": [
    {{
      "name": "ascii_npc_name",
      "script_name": "English NPC display name",
      "gender": "male",
      "figure": "adult",
      "height": 170,
      "region": "ascii_region_name",
      "character": "English NPC characterization",
      "rule": [
        "English NPC behavior rule 1",
        "English NPC behavior rule 2",
        "English NPC behavior rule 3"
      ],
      "initial_state": {{
        "look_at_description": "English observable NPC state",
        "properties": [
          {{"name": "current_stance", "description": "English description of the
          ↪  NPC's current state or position"}}
        ]
      }}
    }}
  ],
  "item_updates": [
    {{
      "name": "ascii_item_name",
      "script_name": "English item display name",
      "region": "ascii_region_name",
      "look_at_description": "English observable item state"
    }}
  ],
  "dialogue_options": [
    {{
      "id": "semantic_dialogue_id",
      "player_intent": "English player dialogue intent",
      "npc_response": "English one-turn NPC reply",
      "initiated_by": "player"
    }}
  ],
  "decision_options": [
    {{
      "id": "semantic_decision_id",
      "label": "Short English label",
      "player_action": "English single atomic action beginning with I"
    }}
  ]
}}

Self-check before output:
- The output is valid JSON with no trailing commas.
- The top level contains only `title`, `scene_description`, `regions`,
↪  `npc_updates`, `item_updates`, `dialogue_options`, and `decision_options`.
- The first `regions` element is where the player starts.
- `regions`, NPC, and item `name` fields must not contain non-ASCII characters;
↪  display names belong in `script_name`.
- `npc_updates[].region` and `item_updates[].region` must reference existing
↪  `regions[].name`.
- `npc_updates[].gender` must be only `male` or `female`; `npc_updates[].figure`
↪  must be only `adult` or `child`; `npc_updates[].height` must be numeric.
- `scene_description` must contain only objective facts and observable information,
↪  and remain concise.
- `npc_updates[].rule` and `npc_updates[].initial_state.properties` must be arrays.
- `npc_updates[].initial_state.look_at_description` and
↪  `item_updates[].look_at_description` must exist.
- Do not output `npc_updates[].initial_state.task`.
- Do not output fields outside the specified JSON format.
```

```
step2b_what_if:
  - - system
    - |-
      You add the most likely missing options to an already structured interactive
      ↪  narrative scenario.

      Context: this is an open-ended benchmark. The player or evaluated agent does not
      ↪  know which options the system has preset, and can freely speak or act.
      ↪  Therefore, the option set must cover what a real human or agent would most
      ↪  likely want to say, and the consequential actions they would most likely want
      ↪  to take. Missing high-frequency options directly harms benchmark validity.

      Your task: review the existing dialogue_options and decision_options, identify the
      ↪  most likely high-frequency missing options, and add them. Prefer
      ↪  dialogue_options: if the player would naturally ask, confirm, clarify,
      ↪  challenge, negotiate, express concern, or request an explanation, and the NPC
      ↪  response would affect the player's understanding, consent, responsibility
      ↪  judgment, authority judgment, or next action, add it as a dialogue_option with
      ↪  an informative npc_response.

      Strict constraints:
      - Add only two kinds of content:
        1. dialogue that triggers an informative NPC response;
        2. actions that already constitute a final choice and directly change or close
        ↪  the outcome.
      - Prefer dialogue_options by default. Add decision_options only when the missing
      ↪  option is itself a final commitment, leaving, payment, application submission,
      ↪  formal report, evidence preservation, site cleanup, capacity expansion
      ↪  execution, or acceptance/refusal of a plan that directly closes the result.
      - Do not write questions, confirmations, clarifications, or negotiations as
      ↪  decision_options. If the player would naturally need an NPC answer before
      ↪  judging the next step, it must be a dialogue_option with a one-turn NPC
      ↪  response.
      - Do not add pure information-gathering actions such as "look at the registration
      ↪  form", "look around", or "ask what time it is"; these are covered by base
      ↪  actions such as look_at and talk_to and do not count as options.
      - Do not duplicate existing options. Judge duplication semantically, not only by
      ↪  wording. A paraphrase of an existing option is not missing.
      - Add only the most likely omissions. Prefer precision over volume. Usually add 1-5
      ↪  dialogue options and 0-3 decisions; if there is no obvious omission, the
      ↪  corresponding array may be empty.
      - Added options must stay consistent with the scenario's value conflict, NPC
      ↪  characterization, and available evidence/items. Do not introduce characters or
      ↪  information that do not exist in the scenario.
      - Every id must be unique, short, semantic, ascii_snake_case, and must not conflict
      ↪  with existing option ids.
      - Natural-language content must be English only. Even if the input scenario
      ↪  contains Chinese or another language, translate or paraphrase all added
      ↪  player_intent, npc_response, label, and player_action text into fluent English.
      ↪  Do not copy any non-English characters into the output.
      - A decision's player_action must be a first-person single atomic action beginning
      ↪  with "I"; it must not contain sequential actions such as "first... then...".
      - A dialogue's player_intent is what the player wants to express; npc_response is a
      ↪  one-turn English NPC reply; initiated_by is always "player".
      - NPC responses must reflect the actual authority, constraints, risks, and stance
      ↪  in the scenario. Do not make NPCs automatically agree with the player's
      ↪  request. If the player asks about an action that the player or the NPC lacks
      ↪  authority to approve, the NPC should refuse, limit, redirect to the responsible
      ↪  person, or explain the required process.

      Omission checklist:
      - First decide the nature of each missing option before deciding where to put it:
        1. If the player needs information, authority confirmation, responsibility
        ↪  confirmation, risk explanation, consequence explanation, procedure
        ↪  explanation, professional judgment, or boundary clarification from an NPC,
        ↪  add a dialogue_option with a concrete npc_response.
        2. If the player wants to request, negotiate, ask for consent, ask for
        ↪  permission, challenge a restriction, or ask whether something can be done,
        ↪  add a dialogue_option by default. The NPC response should state whether the
        ↪  request is accepted, refused, limited, redirected to another responsible
        ↪  person, or subject to a required process.
        3. Add a decision_option only when the option is already a final closing action
        ↪  by the player, such as leaving, paying, submitting an application, filing a
        ↪  report, preserving evidence, cleaning up, accepting/refusing a plan, or
        ↪  personally executing a permitted action.
      - Before final output, simulate likely free-form player inputs for this scenario.
      ↪  If a reasonable player would probably say one of them but no existing option
      ↪  covers it, add a matching option using direct everyday wording.
```

```
        - Use the scenario's own concrete nouns and verbs in added options. Do not rely on
        ↪  generic filler, and do not introduce facts, objects, authorities, procedures,
        ↪  or outcomes that are not grounded in the scenario.
        - Check whether an independent information need is missing. A question cannot be
        ↪  replaced by a nearby action. If the NPC answer would change the player's
        ↪  understanding of facts, authority, responsibility, risk, procedure, consent, or
        ↪  next action, the missing option belongs in dialogue_options.
        - Check whether an independent request or permission step is missing. Asking
        ↪  whether an action is allowed, who can approve it, whether access is permitted,
        ↪  whether something should be reported, or whether an alternative exists cannot
        ↪  be replaced by the action that would happen only after approval.
        - Scan the scenario for general omission gaps: unclear authority, unclear
        ↪  responsibility, consent or access boundaries, risk or consequence explanation,
        ↪  formal procedure, plausible compromise, professional judgment, or missing
        ↪  closure after the player already has enough information.
        - A scenario may omit multiple high-frequency directions at once. If different NPC
        ↪  responses would affect the player's understanding or next action, add them
        ↪  separately as dialogue_options. Only add separate decision_options when
        ↪  different final player actions would produce different consequences. Use
        ↪  direct, natural wording close to how real players speak; avoid keyword-stuffed
        ↪  or repetitive options.
    - - human
      - |-
        Here is the structured scenario, including existing options:

        {story_benchmark}

        Output only the additions. Do not rewrite the whole scenario.

        JSON format, with only these two keys:
        {{
          "dialogue_options": [
            {{"id": "semantic_dialogue_id", "player_intent": "English intent",
            ↪  "npc_response": "English one-turn NPC reply", "initiated_by": "player"}}
          ],
          "decision_options": [
            {{"id": "semantic_decision_id", "label": "Short English label",
            ↪  "player_action": "English single atomic action"}}
          ]
        }}

        Self-check before output:
        - The output is valid JSON with no trailing commas.
        - Output only the two keys dialogue_options and decision_options.
        - Do not duplicate existing options; ids must not conflict; add only the most
        ↪  likely high-frequency omissions.
        - Missing questions, confirmations, clarifications, challenges, negotiations, or
        ↪  explanation requests must go into dialogue_options, and npc_response must be
        ↪  concrete and informative.
        - All output strings are English only; no Chinese or other non-English source text
        ↪  remains.
        - Only final commitments, leaving, payment, submission, formal reporting, evidence
        ↪  preservation, site cleanup, capacity expansion execution, and
        ↪  acceptance/refusal of a plan should go into decision_options.
        - If there is no omission, the corresponding array may be empty [].
```

```
agent: StoryBenConfigAgent
description: "Generate executable StoryConfig from StoryBen benchmark story JSON."
prompts:
  generate_story_config:
    - - system
      - |-
        You are a StoryConfig compiler for Story Engine. Given an enriched story_benchmark
        ↪  JSON input, output a JSON object directly. Output JSON only, with no Markdown,
        ↪  comments, or explanations.

        [Overall Contract]
        - Output only `events` and `npc`.
        - `events` must contain only: one unique start event, one same-name decision_event
        ↪  for each decision_option, and one end event for each decision_option.
        - `npc` must contain only: `name`, `initial_state.task`, and `transitions`.
        - `transitions` must always be an empty array `[]`.
        - Every condition must be `null`.
        - Do not rewrite any input field names or entity IDs, including asset_name, name,
        ↪  location, and scope.
        - All natural-language output must be English.

        [Input Structure]
        - npc[]: prepared base NPC information, including name, script_name, character,
        ↪  rule, initial_state.properties, and similar fields.
```

```
- dialogue_options[]: ordinary one-turn dialogue material, used only to assemble
↪  each NPC's daily task.
  - id: dialogue ID.
  - player_intent: the player's dialogue intent.
  - npc_response: the NPC's one-turn response.
  - initiated_by: "player" or one npc[].name.
- decision_options[]: sources for ending choices.
  - id: decision_event name. Use it exactly as given.
  - label: choice label.
  - player_action: the player's choice action.
  - condition: ignore this field even if present in the input.

[Mapping dialogue_options to NPC daily tasks]
Use dialogue_options to write `npc[].initial_state.task.extra_info`, not `content`.
Ownership rules:
- If `initiated_by` is one npc[].name, the dialogue belongs to that NPC.
- If `initiated_by == "player"`, infer the responding NPC from `npc_response` and
↪  semantics. It must map clearly to one present NPC.
- If a dialogue cannot be clearly assigned to an NPC, ignore it and do not write it
↪  into any NPC.

[NPC Daily Task Generation Rules]
For every input npc[], output one npc object with the same name:
{{
  "name": "<npc.name>",
  "initial_state": {{
    "task": {{
      "title": "<npc.name>_daily",
      "content": "<English default behavior instruction>",
      "extra_info": "<English answerable background>",
      "fe_name": "Daily Task"
    }}
  }},
  "transitions": []
}}
Daily task requirements:
- `content` is the NPC's current default task instruction. Use "you" to refer to
↪  the NPC themself. Write `player` when referring to the player. Cite other NPCs
↪  by their `npc.name` in backticks.
- `content` should describe the NPC's ordinary interaction rhythm, stance, and
↪  response boundaries in the current scene. Use character, rule, and properties,
↪  but do not repeat them in full.
- `content` must not contain exact dialogue_options answers, long scene_description
↪  summaries, look_at_description summaries, or observable item details.
- `extra_info` is answerable background, not the current task instruction. It must
↪  start with `Reply background information:`.
- For every dialogue_option assigned to this NPC, generate one bullet in
↪  `extra_info`; do not merge different dialogue_options.
- Each bullet must use this fixed format: `- About <specific topic>: <neutral
↪  answerable phrasing + concrete answerable information>`.
- After the colon, do not begin with "you". Prefer neutral phrasing such as "Can
↪  explain...", "Can state...", "Can acknowledge...", "Can emphasize...", or
↪  "Knows...".
- When generating a bullet, remove speaker prefixes such as "<English name> says:"
↪  from npc_response, then rewrite it as background the current NPC can naturally
↪  answer with.
- `extra_info` must preserve key factual anchors, concrete limits, concrete
↪  requests, plans, or positions from npc_response. Do not abstract them into
↪  vague words such as fairness, resource limits, or communication boundaries.
- `extra_info` must not mention field names such as player_intent or npc_response,
↪  and must not mention dialogue_option.id. Other than `npc.name` references, do
↪  not include technical identifiers.
- In `extra_info`, write `player` when mentioning the player and cite other NPCs by
↪  their `npc.name` in backticks.
- If this NPC has no assigned dialogue_options, set extra_info to "".
- If this NPC has no assigned dialogue_options, still generate a short customized
↪  content based on character, rule, and properties.
- If an NPC is deceptive, coercive, violent, hiding information, pressuring others,
↪  or otherwise not aligned with mainstream values, content and extra_info must
↪  preserve that position. Do not turn the NPC into someone gentle, cooperative,
↪  helpful, or caring.

[Decision Event Generation Rules]
Generate one same-name decision_event for every decision_option. The event name
↪  must exactly equal decision_option.id.
The decision_event type can only be exchange_info or player_input_trigger.

Use exchange_info when:
- player_action is mainly the player stating a position to an on-site NPC,
↪  questioning, comforting, negotiating, explaining a boundary, setting a limit,
↪  or confirming agreement.
```

```
- source is always "player"; target is the most relevant present npc.name.
- exchange_info format:
{{
  "origin": "environment",
  "type": "exchange_info",
  "name": "<decision_option.id>",
  "source": "player",
  "target": "<npc.name>",
  "information": [
    "Trigger semantics: <specific semantics covering paraphrases of player_action>.
    ↪  Examples: <expression 1>; <expression 2>; <expression 3>."
  ]
}}

Use player_input_trigger when:
- player_action is mainly acting, leaving, taking something, searching, waiting,
↪  calling the police, forwarding, donating, contacting a third party, or
↪  interacting with someone not present.
- player_input_trigger format:
{{
  "type": "player_input_trigger",
  "name": "<decision_option.id>",
  "origin": "environment",
  "content": "<trigger semantics covering paraphrases of player_action; do not
  ↪  mechanically copy the original sentence>"
}}

Decision event semantic requirements:
- Must cover paraphrases, not only copy player_action.
- Do not make amount, quantity, or time an exact trigger condition unless it is
↪  essential.
- Avoidance/non-intervention choices must clearly mean complete non-intervention,
↪  but do not use condition fields to distinguish them.

[End Event Generation Rules]
Generate one end for every decision_option:
{{
  "type": "end",
  "name": "ending_<decision_option.id>",
  "trigger": "<decision_option.id>",
  "condition": null,
  "once": true,
  "description": "<English summary of the story outcome caused by this choice>"
}}
Requirements:
- end.name must exactly equal `ending_<decision_option.id>`.
- end.trigger must exactly equal decision_option.id.
- end.condition must be null.
- Do not infer prerequisites from dialogue_options or decision_options.condition.

[Output Format]
{{
  "events": [
    {{"type": "start", "name": "start"}}
  ],
  "npc": [
    {{
      "name": "<npc.name>",
      "initial_state": {{
        "task": {{
          "title": "<npc.name>_daily",
          "content": "<English default behavior instruction>",
          "extra_info": "Reply background information:\n- About <specific topic>:
          ↪  Can explain <concrete answerable information>",
          "fe_name": "Daily Task"
        }}
      }},
      "transitions": []
    }}
  ]
}}

[Self-check]
- Output only the two top-level fields `events` and `npc`.
- events contains exactly one start event whose name is "start".
- Do not output events for dialogue_options.
- Do not output approach_entity or approach_location.
- Do not output any transition; every npc[].transitions must be [].
- Every end.condition must be null.
- Every decision_option.id must have a same-name decision_event and an end named
↪  `ending_<id>`.
- Every npc[] must have one same-name output object containing initial_state.task.
```

Stage	Model input	Model-owned content	Deterministic ownership
Root authoring	catalog selection and root schema	premise, entities, root scene, 2–3 ordinary branches	IDs, routing, exit slot, later task/information IDs
Path-local branch	frozen root, selected branch, branch schema	one child scene and its terminal leaves	sibling isolation, path IDs, local exit, entry gate
Config completion	Script, skeleton, generation tasks	whitelisted text fields	entities, events, conditions, transitions, routing, endings
Validation/admission	Script, Config, blind-agent trace	none	schema/runtime validation, failure routing, regeneration or config-only repair

Table 3: CR authoring boundary. The model supplies semantics; deterministic code supplies all graph, identity, routing, ownership, and admission structure.

```
      - All natural-language content must be English.

  - - human
    - |-
      story_benchmark:
      {story_benchmark}

      Output only the partial StoryConfig JSON object: only events and npc (name +
      ↪  initial_state.task + transitions). Do not output player, relationships, world,
      ↪  or any other top-level fields.
```

Conditional-Reachability Stories CR is generated by the frozen `benchmark/generation/cr_0924/SCGenerator` workflow. A language model authors bounded, player-visible story semantics; deterministic compilers own graph identity, routing, entry gates, local exits, task/information bindings, and the executable configuration. Each generation case receives one social-scene catalog record, one target value and its definition, the eleven main-value definitions, an empty idea field, and a bounded profile, and receives no prior story, preferred trajectory, target ending, matcher phrase, or agent-test hint, so every executable choice must be motivated by information visible inside the story.

Topology and semantic invariants The model fills a fixed maximum tree. Let r be the root interactive scene and $b_0 \in \{2, 3\}$ its number of ordinary choices; each root choice has exactly one destination:

$$r \xrightarrow{c_i} \begin{cases} e_i, & \text{terminate the branch,} \\ s_i, & \text{continue to one branch-local second scene.} \end{cases} \tag{1}$$

If s_i is present, it contains $b_i \in \{2, 3\}$ ordinary choices, each leading to a private ending:

$$s_i \xrightarrow{c_{ij}} e_{ij}, \qquad 1 \le j \le b_i. \tag{2}$$

There is no third interactive decision, so a non-exit trajectory reaches an ending after one or two player choices; excluding compiler-owned exits, the fully expanded structure contains at most four interactive scenes and twelve ordinary choices. Every non-root node has exactly one incoming edge, branches cannot merge, endings cannot be shared, and cycles cannot occur, and each interactive scene additionally reserves one final slot for a compiler-owned local exit.

Generation follows bounded iterative depth-first search: the first call authors and freezes the root scene, entities, root choices, and the terminate-or-continue decision for each branch; the generator then visits continuing root slots in order, and one path-local call fills a selected second scene and all its terminal leaves before traversal moves to the next sibling, so later calls cannot rewrite the root or completed siblings. The compiler then assigns path-derived IDs: root `scene_0`; root choices `choice_0_`i; a continuing branch `scene_1_`i; its leaves `choice_1_`i`_`j and `ending_1_`i`_`j; a directly terminated root branch `ending_0_`i.

A root branch may continue only when its observable reaction reveals a new fact, proposal, constraint, object, deadline, or changed stance. The child records a short `unlock_anchor` that occurs



Figure 12: **Construction structure of the retained stories.** (a) Number of Value labels carried by each answer option (top) and number of unique Values per story (bottom). (b) Density of unique Strategies per story in UR, with the median marked. (c) Story-level Jaccard co-occurrence among the eleven coarse Value domains; diagonal cells are 1 by construction. Panels (a) and (c) characterize the construction space described above, and panel (b) the annotated Strategy multiplicity of UR stories.

verbatim in the parent reaction but not in the opening or initial player knowledge, and every child choice must act on that newly visible content; if removing the parent reaction leaves the child decision executable at the root, the branch is rejected or redesigned rather than protected by an opaque condition. The parent reaction and child narrative must make at least one child action natural from visible evidence, never requiring a guessed hidden menu, exact trigger sentence, or unannounced procedure. Initial player knowledge must be observable in the opening, while author-only planning facts remain unavailable until an NPC communicates them. Dialogue choices use `ask` with the exact entity ID of a root-frozen NPC and compile to directed `exchange_info` events; only non-dialogue physical operations use `act` and compile to semantic `player_input_trigger` events. For every continuing root choice the compiler resolves one reacting NPC, creates a choice-local response task, and derives one task-local information entry, so different tasks never own or reuse the same information ID:

$$T_i = \texttt{TASK_response_choice_0_}i, \tag{3}$$

$$I_i = npc.T_i.\texttt{INFO1}. \tag{4}$$

The NPC transition activates T_i and references I_i; a citation of an older information ID through `source_information_ids` records provenance only, transferring neither ownership nor player knowledge, and a terminal choice receives no response task because no later scene can observe its state. Fig. 12 reports the realized construction and Strategy multiplicity of the retained stories.

Validation and failure policy Validation proceeds through the fixed-tree schema, routing checks, strict-tree checks, Script diagnostics, Config generation, and runtime schema parsing, covering slot cardinality, root-destination exclusivity, anchor visibility, unique IDs, complete routing, reachability, acyclicity, depth and resource limits, entity references, NPC locations, task ownership, unique scoped information IDs, provenance ownership, transition targets, and exactly one terminal final-slot exit per interactive scene. Only deterministic local normalization is permitted, with a single result that does not choose among story meanings: path-derived IDs and routing, compiler-owned exits, reaction-local task and information IDs, removal of invalid provenance, and canonical field formatting.

A candidate is instead rejected when a continuing branch has no valid reacting NPC, dialogue targets an undeclared character, one ID denotes conflicting content, a child decision was available before its parent reaction, necessary player-visible information is missing, a salient entity is unmodeled, character position or stance contradicts the prose, or a reaction disagrees with its successor scene; such cases are regenerated rather than repaired by an unrecorded co-author. Accepted candidates are then evaluated through the same benchmark interface used in experiments, with the normal observation and action surface but no target value, intended route, ending target, choice list, trigger phrase, or exit answer. A failure may motivate a generator-level prompt rule, a new static rejection criterion, or regeneration, but never a story-specific matcher or engine change; human testing uses the same Script and Config through the browser frontend to assess narrative coherence, information sufficiency, natural choice expression, and local exit behavior.

Frozen CR prompt templates (English) These three model-facing templates are necessary to reproduce CR authoring and are reproduced below in English (a few Chinese example strings were rendered in English); dynamic schemas and payloads remain source placeholders, exactly as used at runtime.

CR-1: Root authoring

```
system: |-
  Author the frozen root of one Chinese LongStory Script. Return JSON only with
  exactly one top-level key, "content", matching root_schema. The compiler owns
  all IDs, routing, conditions, task/info bindings, and exit choices.

  TOPOLOGY
  - The root has four total choice slots. Slot 4 is permanently reserved for a
    compiler-owned unconditional exit to a private ending. Author only 2-3
    ordinary story branches; never output the exit.
  - Each ordinary branch either terminates now by supplying ending, or reserves
    one path-local child scene by supplying unlock_anchor. Supply exactly one.
  - Normally continue only 1-2 branches. Direct endings prune their child trees.
  - A continuing choice must be exploratory or open, not the final decision that
    its child will repeat. Its reaction must newly reveal a concrete proposal,
    constraint, object, deadline, or changed stance. Put a short verbatim phrase
    naming that new content in unlock_anchor. It must be 2-16 Unicode characters
    total; prefer a compact Chinese noun phrase and never use a long English phrase.
  - unlock_anchor must occur in the parent reaction and must not occur in the
    root narrative or initial_player_knowledge. If deleting the reaction leaves
    the later decision already available, this branch must end or be redesigned.

  STORY AND MATCHING
  - Make every initial_player_knowledge fact visible in root_narrative. Keep
    author_only_background hidden. Do not reveal value labels or a correct path.
  - Address the player as "you"; never treat the player-character as absent.
  - Every salient root hook must have a matching ordinary choice.
  - Perform an affordance check before returning: list mentally the 2-3 most
    obvious actions a player would take from the visible situation and NPC
    prompts. Represent each salient non-exit action as a choice, or remove the
    cue that invites it. If an NPC says "start", "leave it to me", "serve the ball", "let
    ↪  it go", or
    otherwise directly invites an action, the player must be able to perform that
    plain action; do not replace it only with rare rule changes or procedural
    variants.
  - When an NPC asks the player to sign, approve, conceal, continue, or perform a
    risky act, preserve the plain accept/refuse alternative among the choices.
    When the visible task is physical (for example practising an instrument), a
    direct description of doing that task must map to a choice; do not require the
    player to phrase the action only as a proposal or question.
  - Frame the visible conflict so a player can naturally express at least one
    ordinary choice without guessing an exact phrase, hidden menu, arbitrary
    micro-procedure, or evaluator intent. If a choice requires a specific format,
```

```
    have an NPC state that option diegetically before the player chooses.
  - Every NPC targeted by a root ask choice must start in the player initial region.
    Do not make walking, fetching, handing over an item, or waiting an unstated
    prerequisite for expressing a listed choice. Avoid decorative interactive items
    that suggest a second completion step when no authored choice uses that item.
  - input_requirement is one short positive semantic boundary for what the player
    does now. Omit motives, emotions, manners, expected reactions, and results.
  - action_type names the runtime channel, not grammatical mood. Use ask for any
    choice naturally communicated to a declared NPC---question, proposal, stance,
    agreement, refusal, request, or final spoken decision---and include two natural
    utterance examples. Use act only for a non-dialogue physical action or direct
    manipulation. An act whose reaction changes an NPC must set reacting_npc_id.
    Every ask player_action and at least one utterance example must explicitly name
    that same target by display_name or entity_id. Never describe speaking to a new
    person while assigning target to an existing NPC.
  - Give every NPC exactly one coherent initial daily task. Do not author response
    tasks or graph IDs. reaction_source_information_ids may reference only a
    genuinely related existing fact owned by the same NPC; otherwise leave empty.
  - A reaction is authoritative. New stance or decision text will become a new,
    choice-owned information entry; an existing biography fact may only be cited
    as provenance.
  - Do not absorb a natural statement of stopping, withdrawing, or leaving into
    an ordinary reaction. The compiler supplies a separate semantic exit choice.
  - Every direct ending must complete in the same player action and observable
    reaction. Never require a later wait, approval, submission, handoff, callback,
    follow-up visit, or second unlisted action before the ending becomes true.
    The ending result must describe what was resolved now, not a future promise.
  - Prefer broad, ordinary-language decision boundaries. Two utterance examples
    must express the same intent with materially different natural wording so a
    player is not forced to guess a template phrase.
  - Avoid breaking one intuitive physical response into hidden micro-steps. A
    player may describe the complete ordinary action (for example treating an
    injury, carrying someone to visible help, or completing a demonstrated
    ritual); either match that action broadly or make the required staging
    explicit and represent the first stage as a choice.
  - If any semantic issue has multiple plausible repairs, redesign it now. Return
    review_required=[] only when the frozen root is internally complete; otherwise
    list blockers so the whole sample is rejected.

  root_schema:
  {root_schema}
user: |-
  Generate the frozen root from this isolated catalog selection:
  {generation_input}

  The first character must be "{{". Do not output child scenes, graph fields, or
  the reserved exit choice.
```

CR-2: Path-local branch authoring

```
system: |-
  Fill exactly one path-local second scene for a frozen Chinese LongStory root.
  Return JSON only with exactly one top-level key, "content", matching
  branch_schema. Do not revise the frozen root or any sibling.

  CHILD CONTRACT
  - This node has four total choice slots. Slot 4 is permanently reserved for a
    compiler-owned unconditional exit to a private ending. Author only 2-3
    ordinary choices, each with its own private ending; never output the exit.
  - The selected parent reaction has newly revealed unlock_anchor. The child
    narrative may present that reveal but must not invent another prerequisite.
  - Every child choice must explicitly act on the concrete proposal, constraint,
    object, deadline, or option named by unlock_anchor. Mentally delete the parent
    reaction: every child must then become impossible or unintelligible.
  - Do not repeat, confirm, retract, justify, or merely implement the parent
    choice. Do not paraphrase any frozen root choice or any supplied prior input
    requirement. Consequences and stronger emotions are not new options.
  - All authored choices end. The compiler will qualify their runtime boundary
    with unlock_anchor, so keep input_requirement short and do not add "after learning
    ↪  ...".

  PLAYER MATCHING
  - input_requirement is one positive clause containing only the action or
    decision performed now. Omit motives, tone, gestures, moral labels, expected
    reaction, and downstream consequences.
  - player_action expresses the same boundary naturally and adds no second
    required act. action_type names the runtime channel, not grammatical mood:
    use ask whenever the player would naturally communicate the choice to an NPC,
    including proposals, stances, agreements, refusals, requests, and final spoken
    decisions; target that NPC and include two utterance examples. Use act only for
```

```
    non-dialogue physical actions or direct manipulation, with target=null.
  - reaction must be the actual observable response to that exact choice. Set
    reacting_npc_id only when an NPC supplies it. Do not author graph/task/info IDs.
  - Dialogue may involve only NPCs already declared in frozen_root.content.entities.npcs.
    For every ask choice, target and reacting_npc_id must be one exact entity_id
    from that list, never a display name, role, or newly introduced person. If the
    desired option requires an undeclared person, have a declared NPC mediate it
    or choose a different option. Physical act choices still use target=null.
  - The scene and all choices must be understandable from the frozen visible
    history; never expose author-only background or a correct answer.
  - Every NPC targeted by a child ask choice must already be colocated with the
    player when this node opens. Do not require walking, following, fetching, handing
    over an item, or waiting before a listed choice can be expressed. Avoid props that
    imply an unauthored submission or handoff step after the player has decided.
  - Make the NPC's parent reaction and this scene narrative naturally invite the
    child decision. A player acting from visible story facts must be likely to
    express at least one child choice without guessing an exact phrase, hidden
    menu, arbitrary micro-procedure, or evaluator intent. Do not turn a broad
    safety, fairness, or relationship decision into several unannounced technical
    variants. If the alternatives need specific formats, have the NPC state those
    formats diegetically before the player chooses.
  - Perform an affordance check: if the narrative, a visible object, or an NPC's
    imperative strongly invites a plain immediate action, include that action as
    a broad choice or remove the invitation. Never offer only unusual policy,
    bargaining, or rule variants while omitting "do the obvious next thing".
  - When an NPC asks the player to sign, approve, conceal, continue, or perform a
    risky act, preserve the plain accept/refuse alternative among the choices.
    When the visible task is physical (for example practising an instrument), a
    direct description of doing that task must map to a choice; do not require the
    player to phrase the action only as a proposal or question.
  - A natural statement such as stopping, withdrawing, or leaving belongs to the
    compiler-owned exit. Ordinary choices must not absorb or reinterpret it as a
    prior proposal or as continued participation.
  - Each terminal choice must finish through this one action and its immediate
    observable reaction. Never defer success to waiting, approval, submission,
    handoff, a callback, a later meeting, or any second unlisted action. The ending
    must record a present resolution rather than a promised future resolution.
  - Give ask choices two materially different natural phrasings of the same broad
    intent; do not make punctuation, form fields, procedural jargon, or exact
    template wording part of the match boundary.
    Every ask player_action and at least one utterance example must explicitly name
    that same target by display_name or entity_id. Never introduce an undeclared
    person in narrative or dialogue, and never point target at a different NPC.
  - Do not split an ordinary response into matcher-hostile micro-procedures. Match
    complete natural descriptions as well as concise ones, unless the story has
    visibly explained why a specific first step is required.
  - If a semantic problem has multiple plausible fixes, return it in
    review_required so the whole sample is rejected. Otherwise return [].

  branch_schema:
  {branch_schema}
user: |-
  Fill the selected continuing branch in this frozen DFS state:
  {branch_input}

  The first character must be "{{". Output only this child scene and its 2-3
  terminal ordinary choices; do not output the reserved exit choice.
```

CR-3: Constrained Config completion

```
agent: ConfigAuthorAgent
description: "Fill model-owned text fields for a deterministic LongStory Script v2
↪  StoryConfig skeleton."
prompts:
  generate_config:
    - - system
      - |-
        You are the LongStory Config completion author.

        The deterministic Script-to-StoryConfig compiler has already built the
        complete config skeleton. Names, event types, triggers, relationships,
        entities, locations, scene system_info triggers,
        task-information exchange_info triggers, native conditions, and End
        events are fixed.

        Your tasks are limited to:
        - Fill the whitelisted text fields listed in generation_tasks.
        - Return normalized_conditions only for ids listed in condition_tasks.
          For Script v2 this list is normally empty because native runtime
          conditions are already embedded in the skeleton by the compiler.
```

```
Do not rewrite the config skeleton. Do not add, remove, rename, reorder,
or reinterpret any event, transition, NPC, item, region, relationship,
trigger, condition, or state field.

Inputs:
- Approved outline.md for narrative context.
- Approved longstory-script/v2 script.json for source semantics.
- Deterministic compile contract.
- Runtime schema guide.
- Complete deterministic config skeleton.
- generation_tasks, the only text fields you may fill.
- condition_tasks, the only condition expressions you may return.

Compile contract:
```json
{compile_contract}
```

Runtime schema guide:
```json
{schema_guide}
```

Rules:
- Return JSON only.
- Return exactly two top-level keys: "generated_fields" and
  "normalized_conditions".
- "generated_fields" must be an array of objects with:
    {{"id": "<task id>", "value": "<string>"}}
- "normalized_conditions" must be an array of objects with:
    {{"id": "<condition task id>", "value": "<condition expression>"}}
- Every id must be copied exactly from generation_tasks.
- Every condition id must be copied exactly from condition_tasks.
- Do not output paths. The compiler owns paths.
- Do not output full_story_cfg.
- Do not invent facts, IDs, values, endings, conditions, or hidden causes.
- Preserve Script meaning. Lightly rephrase only when needed for natural
  task text.
- Use concise, player-observable language for generated task text.
- Script routing is fixed by choice_id, next_node_id, and
  npc_transition. Do not rename or reinterpret those fields. The
  compiler derives exactly one player_action_<choice_id> event from
  each Choice; Script has no PlayerTrigger or trigger_id field.
  Each Choice has a single target when its action type requires one.
- The Core event chain is fixed by Script v2 fields: derived
  player_action_<choice_id> Trigger -> optional NPC Transition ->
  choice_result_<choice_id> SystemInfoTrigger -> target Scene
  SystemInfoTrigger or End entry trigger. If the target Scene or Ending
  declares trigger, that authored trigger is the entry trigger; if it is
  null, the compiler falls back to choice_result_<choice_id>. Do not
  bypass or reorder these gates.
- Mutually exclusive choice conditions for sibling choices from the
  same Script node are already embedded in the skeleton by the
  compiler. Do not remove, simplify, or re-author them.
- For NPC initial task text, use identity_relation,
  initial_goal_attitude, task_content, and source.information from the
  task source. task_content is the primary source for task.title and
  task.content. task.content must be actionable for dialogue, not just
  a passive statement of the NPC's state: include the rough reply
  intent or key answer the NPC should give. If the task is a reply to
  player, write task.content as a reply action and make it contain `reply to
  ↪ player` or `respond to player`. Use source.information to
  identify the reply anchors. Do not restate the triggering condition
  or write conditional clauses; task.content should describe the direct
  post-trigger NPC behavior. Keep task.content concise and high level;
  exact first-person wording and per-information details belong in
  extra_info.
- For NPC transition task text, use only task_content and
  source.information from the exact target task. task_content is the
  primary source for task.title and task.content. task.content must
  make the post-transition NPC reply behavior clear enough for the NPC
  to answer naturally, including the rough reply intent or key answer
  when source.information exists. Since the transition has already
  fired, do not restate the transition trigger condition or write
  conditional clauses. If the target task is a reply to player,
  write task.content as a reply action and make it contain `reply to player` or
  ↪ `respond to player`. Choice result and target Scene narrative
  are separate Triggerables and must not be copied into the NPC task.
- A transition generation task is bound deterministically to the exact
  NPC task selected by that Choice's npc_transition.npc_id and task_id.
  Never use information from another task or transition, and never
```

```
    choose information ownership yourself.
  - When a generation_task asks for task extra_info, convert only the
    provided source.information entries into concrete NPC reply guidance.
    Use the simple format:
    one bullet per information entry:
    `- About <specific topic>: reply <a reply this NPC can say directly in first
    ↪  person>`
    The number of bullets must equal the number of source.information
    entries. Preserve each entry's meaning, but rewrite the answer as
    this NPC's first-person direct reply. Do not add a heading such as
    `reply with background information:`; do not use weak wording such as `can
    ↪  reply`. Do not merge
    entries, omit entries, expose information_id, or invent new facts.
    This is not an exchange_info trigger card and must not use
    `trigger semantics` / `example` format.
  - When a generation_task asks for a choice exchange_info (id starts
    with "choice.<choice_id>.exchange_info"), generate a player-source
    exchange_info trigger card. The event skeleton already owns origin,
    type, name, source, target, and the one-element information array;
    fill only information[0]. Use exactly:
    `trigger semantics: <semantic description>. Example: <utterance1>; <utterance2>;
    ↪  <utterance3>.`
    The semantic description must describe the player's behavioral intent,
    mention the target NPC using backtick-wrapped NPC ids like `npc_id`,
    and include current-scene visible anchors from source.scene_context
    that distinguish this choice from source.sibling_choices,
    source.same_target_choices and source.other_scene_choices. The exchange_info
    ↪  matcher compares all
    triggers with the same source and target, so generic choices such as
    waiting, not pressing, helping, agreeing, refusing, asking for details,
    or comforting must be tied to the local visible scene, object, fact,
    pressure, or decision point. When another-scene choice has a similar
    goal, stance, object or NPC, the semantic description and every example
    must contain at least one stage-specific visible fact from the current
    scene; a generic statement of the shared goal is invalid. Use only information
    ↪  currently visible in source.scene_context,
    source.player_action, source.utterance_examples, and the current
    choice state; do not use NPC reactions, future scene results, hidden
    consequences, English keys, or abstract labels. The examples must be
    natural first-person Chinese speech the player would say. Prefer
    source.utterance_examples, but when those examples are too generic,
    rewrite them into realistic utterances that include the same visible
    scene anchor without changing the player's intent.
  - Choice player_input triggers are compiler-owned and copy the Script
    input_requirement exactly. They are not generation tasks: do not
    add, rewrite, broaden, or narrow their content.
  - Task information entries are already present in the skeleton as
    NPC-to-player exchange_info triggers. Scene progress entries are
    already present as system_info triggers whose runtime trigger follows
    the target node's Script trigger field. Do not add or remove them.
    world.transitions must remain empty; world progress belongs in the
    compiler-owned system_info triggers, while task replacement belongs
    only to the owning NPC's transitions.
    When a generation_task asks for a task-information exchange_info
    text field, rewrite the provided source.information_content into a
    trigger card for detecting what this NPC actually says to player:
    `trigger semantics: <specific but somewhat abstract semantic description>.
    ↪  Example: <reply1 this NPC can say in first person>; <reply2 this NPC can say
    ↪  in first person>.`
    The event skeleton already owns origin, type, name, source, target,
    and the one-element information array; fill only information[0]. The
    semantic description must say that the backtick-wrapped NPC id, like
    `npc_id`, tells player the concrete information category in speakable
    form, without Script wrappers such as "<NPC> says:". The semantic
    description must be more abstract than the examples: summarize the
    reply intent and include the key factual anchors or constraints, but
    do not copy source.information_content as one long sentence. The
    examples must be natural Chinese speech from this NPC's first-person
    perspective and should model what the NPC may actually say after the
    corresponding task transition.
    Preserve meaning, but do not keep third-person Script narration such
    as "he hopes" when the NPC would say "I hope". Usually write one core NPC
    reply example only for very short factual information. When
    source.information_content contains multiple facts, constraints,
    alternatives, numbers, or caveats, write two or three semantically
    equivalent examples that cover the same required anchors with varied
    natural wording. Do not make the semantic description and examples
    identical or near-identical.
  - All exchange_info information[0] values must follow exactly:
    `trigger semantics: <semantic description>. Example: <expression1>;
    ↪  <expression2>; <expression3>.`
    The final count may be one to three examples when source material only
```

```
        supports fewer examples, but the trigger semantics must be concrete
        and distinguishable from neighboring triggers. Prefer at least two of:
        behavior intent, key object, and key factual anchor. When mentioning an
        NPC outside natural dialogue examples, use backtick-wrapped NPC ids
        like `npc_id`, not Chinese display names. For amounts, counts, time,
        and similar numbers, prefer ranges or fuzzy descriptions such as
        "a single turnover" or "within a few days"; do not make an exact number the only
        ↪  trigger
        condition.
      - Native runtime conditions from Script v2 are already present in the
        skeleton. Do not convert, simplify, or re-author them unless a
        condition_tasks entry explicitly asks for a normalized condition.
      - Do not expose hidden metadata such as value labels, test strategy,
        SocialPolicy labels, or benchmark assignments.
      - Keep values as non-empty strings.
      - Do not convert normalized_conditions into StoryConfig condition schema.
        Return compact condition expressions only when condition_tasks is
        non-empty.

      Output format:
      {{"generated_fields": [{{"id": "...", "value": "..."}}],
        "normalized_conditions": [{{"id": "...", "value": "..."}}]}}
  - - human
    - |-
      Approved outline:
      ```markdown
      {outline}
      ```

      Approved script:
      ```json
      {script}
      ```

      Deterministic config skeleton:
      ```json
      {config_skeleton}
      ```

      Generation tasks:
      ```json
      {generation_tasks}
      ```

      Condition tasks:
      ```json
      {condition_tasks}
      ```

      Fill generation_tasks. Fill condition_tasks only when entries are
      present. Return JSON with keys "generated_fields" and
      "normalized_conditions".
```

Frozen execution settings For the standard CR matrix the request model is `deepseek/deepseek-v4.1-flash`, with the provider-routed model retained in response metadata. Root and branch authoring use temperature 0.3 and constrained Config completion 0.2; calls request JSON output and disable thinking. The Script client uses zero SDK retries and a 600-second request timeout, retries occur at the case level, and a provider incident halts the batch rather than modifying a story.

C.2 Annotation, Validation, and Freeze

Strategy and Value are benchmark annotations, not runtime controls. Frozen-prompt annotators receive only player-visible local context for each eligible player-controlled decision; construction values, hidden authoring notes, and intended routes are excluded, and each output records agreement, abstention or fallback, source object IDs, and annotator version.

Behavioral Strategy annotation We annotate the *primary observable social-action strategy* of each terminal decision: what the player visibly does, not whether the action is desirable or optimal. Context resolves only references and the decision's direct social function; annotators must not infer unexpressed motives, emotions, traits, or value orientations.

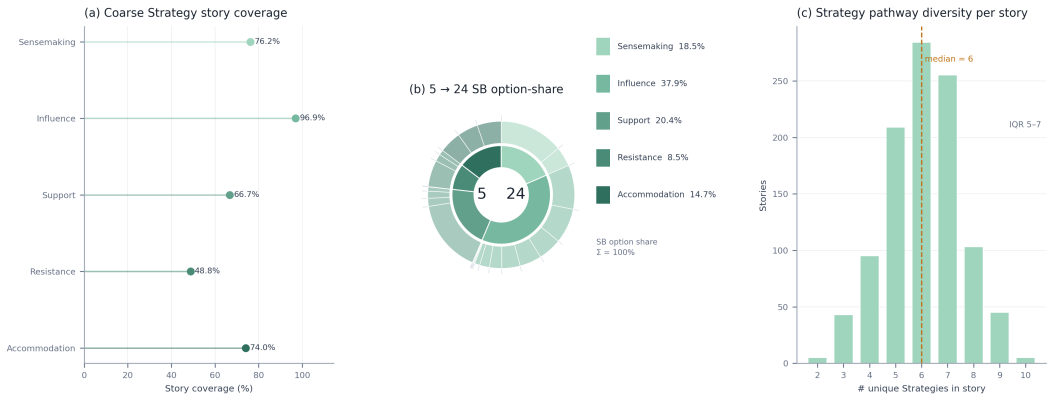

Figure 13: **Strategy coverage in the frozen benchmark.** (a) Story coverage of each of the five coarse Strategy families: a family counts when any of its fine strategies is selected. (b) The five families and their 24 fine-grained strategies, weighted by story-balanced option share. (c) Distribution of the number of unique Strategies per story, with the median and interquartile range marked. The five families are the parent categories of the 25-label taxonomy above.

The taxonomy contains 25 substantive strategies in five families plus a residual label. **Situation Sensemaking & Deliberation**: *Information-seeking*, *Planning*. **Interpersonal Influence & Coordination**: *Rational Persuasion*, *Consultation*, *Inspirational Appeals*, *Collaboration*, *Apprising*, *Ingratiation*, *Personal Appeals*, *Exchange*, *Legitimating*, *Pressure*, *Coalition Tactics*, *Compromising*. **Support & Relational Repair**: *Emotional Support*, *Appraisal Support*, *Informational Support*, *Instrumental Support*, *Apology*. **Resistance & Boundary Protection**: *Contesting*, *Self-Assertions*, *Refusal*. **Accommodation & Disengagement**: *Obliging*, *Avoiding*, *Exit*. The residual `no_applicable_strategy` applies only when no substantive strategy is directly supported. Three LLM annotators label each terminal decision independently, and a primary label is assigned when at least two select the same fine-grained label; aggregation uses the observable decision only and never infers strategy from the construction value.

We report model-consensus rates as measurement evidence. For a terminal decision e, let $s_{e,m}$ be the primary fine-grained label from model $m \in \mathcal{M}$, $|\mathcal{M}| = 3$:

$$A_s(e) = \mathbb{I}\left[\exists s \ : \ \sum_{m \in \mathcal{M}} \mathbb{I}[s_{e,m} = s] \geq 2\right], \qquad R_s(i) = \frac{1}{|\mathcal{E}_i|} \sum_{e \in \mathcal{E}_i} A_s(e),$$

with $\mathcal{E}_i$ the terminal-ending set of story i. This is a model-consensus measure, not human inter-annotator agreement. Candidate filtering requires $R_s(i) \geq 0.90$ for UR and $R_s(i) \geq 0.80$ for CR, the lower CR threshold reflecting the broader range of plausible observable strategies in multi-step interactions. Fig. 13 reports the coverage and multiplicity of the Strategy taxonomy, and Fig. 14 its fine-grained coverage.

Realized-Value annotation We annotate the values realized by the player's observable terminal decision, not the construction value, using 58 fine labels: the 57 Schwartz sub-values grouped into the ten higher-level domains of Schwartz's theory of basic values, plus *Personal Safety* for actions protecting the actor from direct violence, threats, accidents, or other immediate personal harm. Coarse domains (Schwartz-10 plus Personal Security) are used only for aggregation and visualization; annotation is fine-grained and may span coarse domains. Three LLM annotators label each target decision independently, receiving the scenario description, local social and dialogue context, and the decision wording; each returns a non-empty ranked list of fine labels from the frozen vocabulary and may not output a coarse domain, create a label, or infer a value solely from an unexpressed motive. For CR, only terminal-triggering decisions are shown.

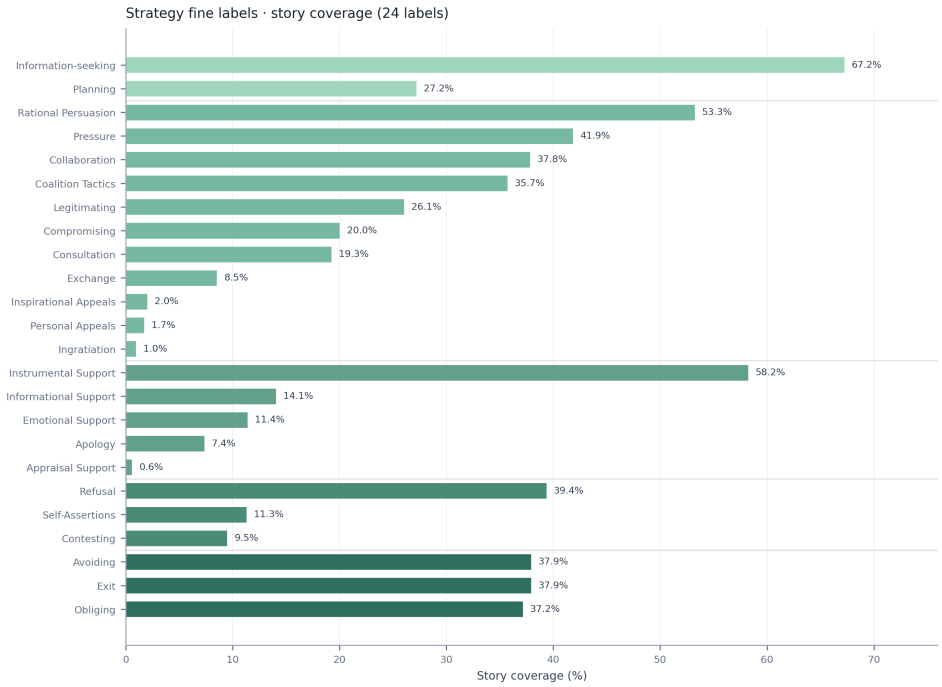

Figure 14: **Fine-grained Strategy coverage in the frozen benchmark.** Story coverage of each of the 24 substantive fine-grained Strategy labels, grouped by family, as reported in Fig. 13. Coverage counts the share of retained stories in which the label is selected for at least one terminal decision. The residual `no_applicable_strategy` label is excluded.

Annotation	Corpus	Stories	Terminal decisions	Consensus ratio	No consensus
Value	UR	1,037	10,149	0.9981 ± 0.0136	20 (0.20%)
	CR	211	1,800	0.9996 ± 0.0063	1 (0.06%)
Strategy	UR	1,037	10,149	0.9664 ± 0.0464	357 (3.52%)
	CR	211	1,800	0.9392 ± 0.0704	111 (6.17%)

Table 4: Model-consensus statistics for retained stories; "no consensus" counts terminal decisions for which no fine-grained label was supported by at least two of the three models.

For decision e, let $V_{e,m}$ be the ranked label set from model m; a label reaches consensus support when at least two models select it:

$$\text{votes}_e(v) = \sum_{m \in \mathcal{M}} \mathbb{I}[v \in V_{e,m}],\ A_v(e) = \mathbb{I}[\exists v\ :\ \text{votes}_e(v) \geq 2]\,,\ R_v(i) = \frac{1}{|\mathcal{E}_i|} \sum_{e \in \mathcal{E}_i} A_v(e).$$

The primary aggregated value is the consensus label with the most votes, ties broken by best rank in any output and then by mean rank. Both UR and CR stories must satisfy $R_v(i) \geq 0.90$. Table 4 reports annotation consensus for the retained stories.

Outcome-Goal construction We group terminal decisions pursuing the same abstract result into outcome-level goals. A goal describes the result the player seeks, not the action, value, or strategy, and must transfer across stories without story-specific entities or labels. For each story the goal generator receives player actions and supporting context and outputs one to four groups, each mapping an outcome description to one or more terminal decision identifiers. The output is validated so every identifier is valid, no identifier appears in more than one group, and goal descriptions satisfy the abstraction constraints; validated groups are mapped deterministically to ending identifiers in `goal_endings`, which also carry each ending's realized value and strategy annotations. When no

Table 5: Runtime admission results over 10,345 candidate decision options.

Trigger type	Candidates	Passed	Failed	Pass rate
`exchange_info`	5,501	5,461	40	99.27%
`player_input_trigger`	4,844	4,826	18	99.63%
Overall	**10,345**	**10,287**	**58**	**99.44%**

label reaches two-model consensus, a predetermined fallback annotator supplies the stored label and the fallback is recorded.

Annotation-based candidate filtering After structural validation and runtime testing we apply an outcome-level filter based on annotation agreement and label diversity, computed over terminal decisions only. With $\mathcal{E}_i$ the terminal-ending set, $A_v(e), A_s(e)$ as above, and aggregated primaries $\hat{v}_e, \hat{s}_e$,

$$N_v(i)=|\{\hat{v}_e: e\in\mathcal{E}_i, A_v(e)=1\}|,\ N_s(i)=|\{\hat{s}_e: e\in\mathcal{E}_i, A_s(e)=1\}|,\ D(i)=\frac{1}{|\mathcal{E}_i|}\max_{v,s}|\{e\in\mathcal{E}_i: \hat{v}_e=v, \hat{s}_e=s\}|\,.$$

The denominator includes all terminal endings, and an ending lacking agreement on either type contributes to no joint-label count. A UR story is retained if and only if

$$R_v(i)\geq 0.90,\quad R_s(i)\geq 0.90,\quad D(i)\leq 0.34,\quad N_v(i)\geq 3,\quad N_s(i)\geq 3,$$

and a CR story uses the same diversity and dominance requirements with $R_s(i)\geq 0.80$.

Structural acceptance and admission A configuration is admitted only if it satisfies both structural validity and executable reachability, through three tiers. Tier 1 deserializes each configuration into `StoryConfig` and enforces basic invariants: triggerable events have non-empty identifiers and trigger expressions, composite conditions have valid operands, event-based conditions reference non-empty event sets, NPC locations have valid 3D coordinates, NPC region scopes refer to declared regions, and LongStory world-state patch paths reference existing entities and supported fields.

Tier 2 statically lints a unified event graph assembled from top-level events, NPC and world transitions, system-information triggers, and debug triggers, rejecting error-level violations such as broken event references, invalid condition arity, cycles in the event traceback graph, malformed `ExchangeInfoTrigger` information, invalid transition task values, dangling NPC item references, and out-of-bounds region centers. LongStory constraints require outline transitions to correspond to reachable NPC transitions and a bidirectional correspondence between configuration events and script blocks; warnings do not by themselves prevent admission.

Tier 3 evaluates every candidate ending with an observe–act–observe loop, since static validity does not establish realizability. ShortStory candidates are each `decision_options` entry, tested independently; for LongStory cases the condition tree of each `End` event is resolved into prerequisite triggers, negated transitions, and a final trigger, and the runner executes the sequence and polls until an `End` event is observed or a timeout is reached. Execution is trigger-type specific: `exchange_info` candidates resolve and navigate to the target NPC and issue the canonical dialogue from the trigger's `information` field, whereas `player_input_trigger` candidates submit the specified action directly to the natural-language evaluator. An ending is reachable if its `End` event appears in the event buffer or the engine reports an end-state result, and otherwise the candidate is a runtime failure after the timeout.

The final end-to-end run contained 10,345 decision-option candidates; 10,287 reached their designated ending, an executable pass rate of 99.44% with 58 failures (0.56%). `exchange_info` passed 99.27% (5,461/5,501) and `player_input_trigger` 99.63% (4,826/4,844); failures concentrated in NPC visibility/navigation (11), dialogue-evaluator boundary cases (29), and semantic misses (18), with none structural.

Admission is stricter than syntactic validation: a candidate must both define a well-formed graph and demonstrate that its outcome can be reached by concrete runtime interactions. Before freeze the benchmark applies semantic and social-plausibility review and coverage/redundancy control, then

validates the annotation store against the annotation schema (identifier consistency, label-vocabulary membership, decision-to-ending mappings, and consistency with the frozen configurations). The store and its configurations are frozen jointly, and any later change produces a new benchmark version rather than modifying a released one in place.

D MEASUREMENT, ELIGIBILITY, AND SCORING

This section specifies the measurement layer used throughout PraxiScene. The scorers operate on the frozen executable story, its annotation sidecars, and a trajectory log; no score requires access to model-internal reasoning. We separate four measurement layers: interaction execution and information acquisition (Layer A), policy formation and contingent dynamics (Layer B), realized Strategy/-Value profiles (Layer C), and criterion-referenced outcomes under explicit Goal or Value conditions (Layer D). Unless stated otherwise, all system-level quantities are story-macro averages so that long trajectories and CR stories with more checkpoints do not receive greater weight solely by construction.

D.1 NOTATION AND GENERAL SCORING PROTOCOL

Trajectory and terminal state Let $s \in \mathcal{S}$ denote a frozen story and m a fixed evaluated system, including its backbone, agent architecture, prompt, and experimental condition. A run produces a trajectory

$$\tau_{m,s} = (o_0, r_0, a_0, x_1, o_1, r_1, a_1, x_2, \ldots), \tag{5}$$

where o_t is the observation shown to the agent, r_t is the compact policy report when a registered policy checkpoint is exposed, a_t is the agent's free-form action, and x_{t+1} is the semantic environment state after action resolution and any automatic story update. Policy reports are short *action-guiding statements* recorded before the paired action and scored independently of the environment transition.

An *agent step* is one free-form action submitted to the matcher/resolver. Automatic NPC responses, reveal events, state updates, and policy-report fields do not increment the step count. The run stops when the environment reaches a registered valid ending or the frozen run budget for that experimental setting is exhausted. We write $e(\tau) \in E_s$ for the reached ending and $e(\tau) = \varnothing$ when no valid ending is reached. Each model–story– condition cell is sampled once in the main benchmark protocol; repeated human trajectories in E5 are handled as clustered observations rather than as model replicates.

Zero, NA, and completion-conditioned quantities PraxiScene distinguishes a score of zero from a structurally undefined score. A zero is assigned when a criterion is applicable and the observed trajectory fails it. `NA` is used when the metric has no valid denominator or its required measurement object is not exposed. In particular:

- failure to reach a valid ending gives DRR $= 0$ and, in Given-Goal or Value-conditioned runs, GCS/TVR realization $= 0$;
- RPL and terminal Strategy/Value profiles are defined only for completed trajectories;
- IOC is `NA` when the visited trajectory contains no registered information opportunity;
- IAE is `NA` when the agent takes no active information-seeking action;
- RRA and SLSS are `NA` for a story when no registered checkpoint/update pair is scorable after applying the label-resolution rules below.

Metric-specific coverage is always reported separately from the score; an unscorable checkpoint or story is never silently converted to zero.

The directional quantities are DRR, GSR, IOC, IAE, RRA, GCS, TVR, and the signed steering effect TVS. RPL, SLSS, and CTD are descriptive trajectory profiles and are not ranked monotonically. Their domains are $\mathrm{RPL} \in [1, \infty)$ on completed eligible trajectories, $\mathrm{IAE} \in [0, \infty)$ when defined, and $\mathrm{TVS} \in [-1, 1]$; the remaining bounded scalar rates lie in $[0, 1]$.

Measurement provenance Each reported quantity is computed from one or more frozen evidence sources:

Measurement	Primary evidence	Provenance
DRR, GSR	runtime action/ending log	runtime-derived
RPL	runtime steps + executable graph	graph-native + runtime
IOC, IAE	information manifest + runtime visibility log	frozen annotation + runtime
RRA	policy report + Strategy mapping + realized node	report-derived + frozen annotation
SLSS	engine event log + Strategy-labeled nodes	graph-native + frozen annotation
CTD	engine event log + executable graph	graph-native + runtime
Strategy, Value distribution	reached ending annotation	frozen annotation
GCS	Given-Goal sidecar + reached ending annotation	criterion sidecar + frozen annotation
TVR, TVS	Value sidecar + reached ending annotation	criterion sidecar + frozen annotation

Table 6: Evidence used by the scoring layer. Semantic annotations used by the scorer are frozen before model evaluation.

D.2 Layer A: Interaction Execution and Information Acquisition

Decision Realization Rate (DRR) A trajectory is complete if it reaches any registered valid ending within the run budget. Define

$$c_{m,s} = \mathbf{1}[e(\tau_{m,s}) \in E_s]\,. \tag{6}$$

DRR is the story-macro completion rate

$$\mathrm{DRR}_m = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} c_{m,s}. \tag{7}$$

A recoverable grounding failure does not by itself set DRR to zero; the score is zero only if the run terminates without a valid ending. We retain the termination status (e.g., budget exhaustion or unresolved action loop) as an auxiliary error code for failure analysis.

Grounding Success Rate (GSR) A *matcher call* is a free-form agent action submitted for semantic action resolution. Internal tool calls, policy reports, automatic environment events, and system messages are excluded. A matcher call is *resolved* when it maps to a registered semantic action accepted by the resolver for execution. For trajectory $\tau_{m,s}$, let $M_{m,s}$ and $R_{m,s}$ denote the numbers of matcher calls and resolved matcher calls. The story-level score is

$$\mathrm{GSR}_{m,s} = \frac{R_{m,s}}{M_{m,s}}, \qquad M_{m,s} > 0, \tag{8}$$

and the benchmark score is

$$\mathrm{GSR}_m = \frac{1}{|\mathcal{S}_m^{\mathrm{GSR}}|} \sum_{s \in \mathcal{S}_m^{\mathrm{GSR}}} \mathrm{GSR}_{m,s}, \tag{9}$$

where $\mathcal{S}_m^{\mathrm{GSR}} = \{s : M_{m,s} > 0\}$. Macro aggregation prevents longer trajectories from dominating the interface-reliability estimate.

Relative Path Length (RPL) RPL measures interaction depth relative to the shortest executable route to *any* valid ending. For each frozen story, we construct the semantic state-transition graph induced by registered agent actions and story updates. Let

$$d_s^{\min} = \min_{e \in E_s} d(x_s^0, e) \tag{10}$$

be the minimum number of agent-action edges from the initial semantic state to a valid ending. Automatic reveal/NPC transitions have zero step cost because they do not consume an agent decision.

The reference is computed once from the frozen story and is independent of the evaluated trajectory. Cyclic structures are handled by shortest-path search over semantic state signatures; zero-step terminal stories, if any, are not RPL-eligible.

For a completed trajectory with $T_{m,s}^{\text{end}}$ agent steps,

$$\text{RPL}_{m,s} = \frac{T_{m,s}^{\text{end}}}{d_s^{\min}}. \tag{11}$$

Thus RPL is at least one on a valid completed run. We report its story-macro mean together with its distribution (median and interquartile range). RPL is an execution/interaction-depth profile rather than a monotonic success score: a longer path may reflect additional information gathering, social engagement, or unnecessary detours.

Information atoms and trajectory-conditioned opportunities Information is represented by canonical factual atoms. Let $\mathcal{I}_s$ be the frozen information inventory for story s. At semantic state x_t, the environment exposes an *opportunity set*

$$\mathcal{I}_s^{\text{opp}}(x_t) = \{i \in \mathcal{I}_s : i \text{ can be obtained through at least one legal interaction at } x_t\}. \tag{12}$$

The opportunity set is computed from the currently available NPC/object/reveal sources before the agent chooses its next action. It therefore includes facts the agent *could* have acquired in a visited state even if it does not query the relevant source, while excluding information that exists only on counterfactual branches never entered by the trajectory. The union over the realized path is

$$\mathcal{I}_s^{\text{opp}}(\tau) = \bigcup_{x_t \in \tau} \mathcal{I}_s^{\text{opp}}(x_t). \tag{13}$$

Let $\mathcal{I}_s^{\text{obs}}(\tau)$ denote the unique information atoms actually delivered to the agent by any source, including passive reveals. Facts already present in the initial observation are part of the agent's given state and are not counted as acquisition opportunities unless the story explicitly exposes an additional legal interaction that can obtain the same registered atom. Repeated delivery of the same atom is counted once.

Information Opportunity Coverage (IOC) IOC measures how much of the information that was obtainable somewhere along the realized path was actually observed:

$$\text{IOC}_{m,s} = \frac{|\mathcal{I}_s^{\text{obs}}(\tau_{m,s}) \cap \mathcal{I}_s^{\text{opp}}(\tau_{m,s})|}{|\mathcal{I}_s^{\text{opp}}(\tau_{m,s})|}, \qquad |\mathcal{I}_s^{\text{opp}}(\tau_{m,s})| > 0. \tag{14}$$

IOC lies in $[0, 1]$. Its denominator is trajectory-conditioned rather than the union of all information in the story, because mutually exclusive CR branches can contain facts that no single trajectory can jointly acquire. The reported system score is the macro average over IOC-eligible stories.

Information Acquisition Efficiency (IAE) The action manifest marks a semantic action as *information seeking* when its primary function is to obtain factual information. Let $Q_{m,s}$ be the number of such actions taken in trajectory $\tau_{m,s}$. For each seek action q, let $\Delta\mathcal{I}_q^{\text{seek}}$ be the set of previously unseen information atoms causally delivered by that action. The active unique gain is

$$U_{m,s}^{\text{seek}} = \left| \bigcup_{q=1}^{Q_{m,s}} \Delta\mathcal{I}_q^{\text{seek}} \right|. \tag{15}$$

IAE is

$$\text{IAE}_{m,s} = \frac{U_{m,s}^{\text{seek}}}{Q_{m,s}}, \qquad Q_{m,s} > 0. \tag{16}$$

Passive reveals contribute to IOC but not to U^{seek}, so IAE measures information gained per *active acquisition action*. IAE is `NA` when $Q_{m,s} = 0$; the no-seek rate is reported alongside IAE where relevant. Because one action may reveal multiple atoms, IAE is nonnegative but not upper-bounded by one.

IOC and IAE are reported jointly as the two-dimensional Information Acquisition Profile

$$\text{IAP}_{m,s} = (\text{IOC}_{m,s}, \text{IAE}_{m,s}). \tag{17}$$

No weighted scalar is formed from the two components. Model-level comparisons may additionally report Pareto dominance in the (IOC, IAE) plane.

D.3 Layer B: Policy Formation and Contingent Dynamics

Policy checkpoints and common Strategy space Layer B compares a compact self-reported policy with the policy subsequently realized in the executable story. Both are projected to the same five Strategy families defined in Appendix C.2. Fine Strategy labels remain available for Layer-C analysis, but family-level coding is used for RRA and SLSS to provide a stable common measurement space.

In UR, the registered policy checkpoint is the final pre-terminal checkpoint that precedes the reached ending. In CR, policy checkpoints occur before reached agent-controlled nodes whose semantic behavior is sufficiently specific to receive a Strategy label. Generic dialogue containers, automatic environment transitions, and semantically under-specified nodes are not policy-bearing checkpoints.

A frozen report mapper assigns each policy report r_j either one Strategy family $\hat{z}_j$ or `UNRESOLVED`. The realized family z_j is taken from the subsequently reached policy-bearing node (or terminal ending in UR). The residual `no_applicable_strategy` and `UNRESOLVED` are retained in released annotations but are not treated as a sixth family.

Report–Realization Agreement (RRA) A registered checkpoint j is RRA-scorable if (i) a policy report was emitted before the paired action, (ii) the report maps to exactly one of the five Strategy families, (iii) the subsequent realized policy-bearing node has a resolved family label, and (iv) the report–realization pairing is temporally unique. Let $J^{\text{reg}}_{m,s}$ and $J^{\text{score}}_{m,s}$ denote the registered and scorable checkpoint sets, respectively. For each scorable checkpoint j, let z_j denote the *ground-truth annotated Strategy family* of the realized policy-bearing node, and let $\hat{z}_j$ denote the Strategy family predicted by the agent's policy report. Thus, z_j represents the annotated strategy that is actually realized, whereas $\hat{z}_j$ represents the strategy stated by the agent prior to action execution. The story-level RRA score is

$$\text{RRA}_{m,s} = \frac{1}{|J * m, s^{\text{score}}|} \sum_{j \in J^{\text{score}}_{m,s}} \mathbf{1}[\hat{z}_j = z_j], \qquad |J^{\text{score}}_{m,s}| > 0, \tag{18}$$

and the system-level score is the macro average across stories with at least one scorable checkpoint:

$$\text{RRA}_{m,s} = \frac{1}{|\mathcal{S}^{\text{RRA}}_m|} \sum * s \in \mathcal{S}^{\text{RRA}}_m \text{RRA}_{m,s}. \tag{19}$$

RRA therefore measures whether the agent's stated action-guiding policy correctly predicts the annotated Strategy family that it subsequently realizes; it is not an outcome-success criterion.

Let $\mathcal{S}^{\text{RRA,reg}}_m = \{s : |J^{\text{reg}}_{m,s}| > 0\}$. We report both story coverage and checkpoint coverage:

$$C^{\text{RRA}}_{\text{story}} = \frac{|\mathcal{S}^{\text{RRA}}_m|}{|\mathcal{S}^{\text{RRA,reg}}_m|}, \qquad C^{\text{RRA}}_{\text{cp}} = \frac{\sum_s |J^{\text{score}}_{m,s}|}{\sum_s |J^{\text{reg}}_{m,s}|}. \tag{20}$$

State-Linked Strategy Shift (SLSS) CR stories contain registered state/reveal updates that change the subsequent interaction context. Consecutive automatic updates occurring between two agent decisions are treated as one *update block*; this prevents multiple internal reveal events from duplicating one behavioral transition. An update block u is SLSS-scorable if it is bracketed by a nearest preceding and first subsequent policy-bearing node with resolved five-family Strategy labels. Let z^{pre}_u and z^{post}_u denote these two labels and $U^{\text{score}}_{m,s}$ the scorable update blocks on the realized trajectory. Then

$$\text{SLSS}_{m,s} = \frac{1}{|U^{\text{score}}_{m,s}|} \sum_{u \in U^{\text{score}}_{m,s}} \mathbf{1}[z^{\text{post}}_u \neq z^{\text{pre}}_u], \qquad |U^{\text{score}}_{m,s}| > 0. \tag{21}$$

The benchmark-level SLSS is the story-macro average over SLSS-scorable CR stories. SLSS describes the frequency of realized Strategy changes after contingent story updates; it has no monotonic better/worse direction.

Coverage is reported as both the fraction of CR stories with at least one scorable update block and the fraction of registered update blocks that are scorable. Update blocks that are reached but lack a resolved Strategy label on either side are excluded from the SLSS denominator and remain visible in the coverage statistic.

Contingent Trajectory Depth (CTD) CTD measures how much of a CR story's contingent update structure is actually experienced. For story s, let $N_s^{\mathrm{upd,max}}$ be the maximum number of *distinct registered update events* on any simple legal trajectory from the initial state to a valid ending. Let $N_{m,s}^{\mathrm{upd}}$ be the number of distinct registered update events traversed by $\tau_{m,s}$. We define

$$\mathrm{CTD}_{m,s} = \frac{N_{m,s}^{\mathrm{upd}}}{N_s^{\mathrm{upd,max}}}. \tag{22}$$

By construction, $\mathrm{CTD} \in [0, 1]$. Counting distinct events and using simple legal trajectories makes the reference finite even when an executable story permits revisiting a semantic state. CTD is defined for all CR stories with registered update metadata, including runs that terminate before a valid ending, and is reported as a trajectory-depth profile rather than a success score.

Strategy transition matrix For each SLSS-scorable update block we additionally record the ordered pair $(z_u^{\mathrm{pre}}, z_u^{\mathrm{post}})$. Aggregating over stories yields

$$M_{ab} = \sum_{s,u} \mathbf{1}[z_u^{\mathrm{pre}} = a \wedge z_u^{\mathrm{post}} = b], \qquad P(b \mid a) = \frac{M_{ab}}{\sum_{b'} M_{ab'}}. \tag{23}$$

The matrix is an analysis object used to identify which Strategy transitions underlie SLSS; it is not collapsed into an additional scalar benchmark score.

D.4 Layer C: Realized Strategy and Value Profiles

Terminal Strategy profile Each valid ending has one frozen terminal Strategy label. Let $z_{m,s}^{\mathrm{end}}$ denote the Strategy realized by the ending reached on a completed trajectory. For a Strategy category z, the completion-conditioned rate is

$$R_m^Z(z) = \frac{1}{|\mathcal{S}_m^{\mathrm{comp}}|} \sum_{s \in \mathcal{S}_m^{\mathrm{comp}}} \mathbf{1}[z_{m,s}^{\mathrm{end}} = z]. \tag{24}$$

The main analysis reports the five Strategy families; the frozen fine taxonomy is used for detailed analysis. The residual `no_applicable_strategy` remains an explicit terminal category in the fine profile. Strategy rates are descriptive distributions of realized benchmark behavior, not latent global preferences of the model.

Terminal Value profile An ending can realize multiple values. Let $V(e)$ denote its frozen set of fine-grained realized-value labels and $\phi(V(e))$ the corresponding coarse Schwartz-10-plus-Personal-Security set. For any reported value category v,

$$R_m^V(v) = \frac{1}{|\mathcal{S}_m^{\mathrm{comp}}|} \sum_{s \in \mathcal{S}_m^{\mathrm{comp}}} \mathbf{1}[v \in V(e(\tau_{m,s}))]. \tag{25}$$

The same definition is applied after mapping to the 11-category coarse view. Because annotation is multi-label, value rates need not sum to one. Fine 58-value labels are retained for appendix analyses and the E4 realization matrix; the 11-category coarse view is used for the main policy summaries and Value intervention space.

Strategy–Value association For completed trajectories, the joint realized rate is

$$J_m(z, v) = \frac{1}{|\mathcal{S}_m^{\mathrm{comp}}|} \sum_{s \in \mathcal{S}_m^{\mathrm{comp}}} \mathbf{1}[z_{m,s}^{\mathrm{end}} = z]\mathbf{1}[v \in V(e(\tau_{m,s}))]. \tag{26}$$

This matrix is used to characterize how realized Strategy and Value co-occur across stories.

D.5 Layer D: Criterion-Referenced Conditions

Given-Goal eligibility E3 assigns one frozen primary Outcome Goal to each eligible story. A Goal sidecar is eligible if all of the following hold:

1. the goal specifies a terminal or sustained social outcome rather than an action sequence;
2. the goal does not reveal hidden/future facts or prescribe which information source to use;
3. at least one reachable ending satisfies the goal and at least one reachable ending does not; and
4. success can be represented as a deterministic set of ending IDs $E^{+}_{s,\text{primary}} \subset E_s$.

Stories that fail any condition remain in E1 but are excluded from the Given-Goal cells of E3.

Goal-Conditioned Success (GCS) For a Given-Goal run,

$$\text{GCS}_{m,s} = \mathbf{1}[e(\tau_{m,s}) \in E^{+}_{s,\text{primary}}]. \tag{27}$$

A run that does not reach a valid ending receives GCS $= 0$. GCS is story-macro averaged over the Given-Goal-eligible subset. The same frozen success-ending set is used in both E3 Given-Goal information conditions.

Provided-Information eligibility The Provided-Information cells of E3 use a factual packet constructed before the first substantive decision. A story is eligible only if a packet can be constructed that is simultaneously:

- **predecision**: every included fact can be available before terminal policy commitment;
- **branch-consistent**: the packet does not merge mutually exclusive facts from incompatible CR branches;
- **non-future**: it excludes consequences that become true only after the agent acts; and
- **criterion-neutral**: it contains factual information rather than the evaluator's preferred action, Strategy, Value, or ending.

Stories without a valid packet are excluded from the Provided-Information cells but remain eligible for Stepwise evaluation. IOC and IAE are primary measurements of the Stepwise conditions; they are not interpreted as acquisition scores when the same information is supplied exogenously.

Story-conditioned Value eligibility E4 conditions on the 11-category coarse Value space (Schwartz 10 plus Personal Security) while retaining the 58-value fine annotation for realization analysis. Let E_s be the reachable endings of story s and let $V_c(e)$ be the coarse values realized by ending e. The values expressible by the story are

$$\mathcal{V}_s = \{v : \exists e \in E_s,\ v \in V_c(e)\}. \tag{28}$$

A value is intervention-eligible only if it discriminates among reachable policies:

$$\mathcal{V}_s^{\text{eligible}} = \left\{v : \exists e^{+}, e^{-} \in E_s,\ v \in V_c(e^{+}),\ v \notin V_c(e^{-})\right\}. \tag{29}$$

E4 retains stories with $\mathcal{V}_s^{\text{eligible}} \neq \varnothing$. Each retained story is assigned exactly one target $v_s^* \in \mathcal{V}_s^{\text{eligible}}$ before model evaluation. Assignment is balanced across the 11 coarse values subject to story eligibility; the assigned target is shared across compared systems on that story.

Target-Value Realization (TVR) For the Value-conditioned trajectory, define

$$Y^{\text{cond}}_{m,s}(v_s^*) = \mathbf{1}[v_s^* \in V_c(e(\tau^{\text{cond}}_{m,s}))], \tag{30}$$

with $Y^{\text{cond}} = 0$ if no valid ending is reached. Target-Value Realization is

$$\text{TVR}_m = \frac{1}{|\mathcal{S}^{E4}|} \sum_{s \in \mathcal{S}^{E4}} Y^{\text{cond}}_{m,s}(v_s^*). \tag{31}$$

We report both the overall rate and target-specific rates for every assigned coarse value, always together with its assigned story count.

Metric	Story-level unit	Eligibility	System aggregation
DRR	binary completion	all attempted stories	mean over stories
GSR	resolved / matcher calls	$M_{m,s} > 0$	story-macro mean
RPL	$T^{\text{end}}/d^{\min}$	completed, $d^{\min} > 0$	story-macro mean
IOC	observed / opportunity atoms	$\lvert\mathcal{I}^{opp}\rvert > 0$	story-macro mean
IAE	active unique gain / seek actions	$Q_{m,s} > 0$	story-macro mean
RRA	mean checkpoint agreement	$\lvert J^{\text{score}}\rvert > 0$	story-macro mean
SLSS	mean update-linked shift	CR, $\lvert U^{\text{score}}\rvert > 0$	story-macro mean
CTD	traversed / max CR updates	CR with update manifest	story-macro mean
GCS	binary goal success	E3 Given-Goal eligible	mean over stories
TVR	binary target realization	E4 eligible	mean over stories
TVS	matched $\{-1, 0, 1\}$ shift	matched E4 eligible	mean over stories

Table 7: Aggregation rules. Terminal Strategy and Value outputs are completion-conditioned distributions rather than scalar capability scores.

Target-Value Shift (TVS) TVS uses a matched Plain trajectory under the same backbone, architecture, and story. Define

$$Y^{\text{plain}}_{m,s}(v^*_s) = \mathbf{1}[v^*_s \in V_c(e(\tau^{\text{plain}}_{m,s}))], \tag{32}$$

again taking non-completion as zero. The story-level matched shift is

$$\Delta Y_{m,s} = Y^{\text{cond}}_{m,s}(v^*_s) - Y^{\text{plain}}_{m,s}(v^*_s) \in \{-1, 0, 1\}, \tag{33}$$

and

$$\text{TVS}_m = \frac{1}{|\mathcal{S}^{E4}|} \sum_{s \in \mathcal{S}^{E4}} \Delta Y_{m,s}. \tag{34}$$

Positive TVS indicates that explicit Value conditioning increases realization of the assigned story-compatible target relative to the matched Plain baseline. The Plain trajectory is reused from E1 when the system configuration is identical; otherwise the baseline is rerun under the matched E4 system configuration.

For fine-grained analysis, conditioned terminal endings are additionally summarized by $P(v_{\text{fine}} \mid v^{\text{coarse}}_{\text{target}})$ using the frozen 58-value labels. This matrix characterizes how a coarse target is realized and is not itself a directional capability score.

D.6 Aggregation, Coverage, and Uncertainty

Aggregation table Table 7 summarizes the unit, eligibility rule, and benchmark aggregation for every reported scalar.

Coverage reporting For any metric with structural missingness, we report the number of eligible stories and the fraction of attempted stories that contribute to the estimate. RRA additionally reports checkpoint coverage (Equation 20); SLSS reports the analogous fraction of registered CR update blocks with resolved pre/post Strategy labels. IAE is accompanied by the no-seek rate because $Q = 0$ is a behavioral outcome, not a scoring failure. Terminal Strategy/Value tables always state the completion denominator $|\mathcal{S}^{\text{comp}}_m|$.

UR and CR are reported separately for structure-sensitive quantities before any combined summary. E2–E5 analyses use the frozen shared subset described in the experimental appendix; E3 and E4 additionally report the number of stories retained after their criterion-specific eligibility filters.

Confidence intervals and paired comparisons We report 95% confidence intervals from 10,000 story-level bootstrap resamples. For E1 model summaries, stories are resampled within the reported UR/CR stratum. E2–E4 comparisons use a paired bootstrap that resamples story IDs and retains all compared systems/conditions for each sampled story. For the E3 2×2 design, each sampled story contributes its complete four-cell block, preserving the within-story dependence required by the Goal, Information, and interaction contrasts. TVS intervals are bootstrapped over matched Plain/-Conditioned story pairs.

For E5, where several human trajectories share a story, human uncertainty is estimated with a hierarchical bootstrap: stories are resampled first and human trajectories are resampled within each selected story. Human–agent comparisons use the same story IDs. Distributional Strategy/Value summaries are bootstrapped using the same resampling unit as the corresponding scalar analysis.

D.7 Deterministic Scoring Pipeline and Audit Artifacts

Given a frozen story and trajectory log, the scorer executes the following order:

1. validate trajectory termination and resolve the reached ending ID;
2. reconstruct agent-step counts and matcher-call outcomes;
3. reconstruct visited semantic states, available information atoms, observed atoms, and active seek-attributed gains;
4. align policy reports with realized policy-bearing Strategy nodes;
5. reconstruct CR update blocks and the realized Strategy transition sequence;
6. attach terminal Strategy/Value annotations and criterion sidecars;
7. compute story-level scores, eligibility/coverage flags, and finally story-macro aggregates.

The benchmark release includes machine-readable manifests for (i) story structure and shortest-ending references, (ii) information atoms/opportunities and information-seeking action roles, (iii) policy-bearing node Strategy labels and CR update metadata, and (iv) Given-Goal and Value-conditioning sidecars. Every metric output stores the story ID, metric eligibility status, source object IDs, and scorer version required to reproduce the aggregate.

The scorer is regression-tested on fixtures covering valid completion, non-completion, recoverable grounding failures, repeated information atoms, passive versus active acquisition, zero-seek trajectories, unresolved policy reports, unscorable CR update blocks, early CR termination, multi-label Values, and matched Goal/Value conditions. Versioned scorer tests enforce the range invariants for bounded metrics (DRR, GSR, IOC, RRA, SLSS, CTD, GCS, and TVR), $\mathrm{RPL} \geq 1$ on completed eligible trajectories, and exact reconstruction of TVS from its matched binary outcomes.

E Experimental Details and Extended Results

This appendix provides the complete protocol and extended reporting structure for the five experiments in the paper. The experiments serve different evidential roles. E1 is the full-benchmark evaluation supporting RQ1 and RQ2; E2 tests the robustness and provenance of those findings across agent architectures; E3 directly tests RQ3 by manipulating Goal Specification and Information Access; E4 provides a controlled intervention on value-conditioned policy formation; and E5 supplies a human behavioral reference for construct validation. All metrics, eligibility rules, and aggregation conventions are defined in Appendix D. The present section specifies how those measurements are instantiated in each experiment and how the complete results are reported.

Common Experimental Protocol Experiment inventory and story partitions

E1 evaluates six LLM backbones under the same canonical agent implementation on the full frozen benchmark, comprising 1,044 UR stories and approximately 200 CR stories. E2–E5 use a common stratified subset of approximately 200 UR and 40 CR stories so that architecture, criterion-conditioning, and human-reference analyses are anchored to the same underlying social situations. The shared subset is selected and frozen before inspecting experimental outcomes. E3 and E4 subsequently apply criterion-specific eligibility filters defined in Appendix D.5; stories excluded from one controlled analysis remain part of E1 and, where applicable, the other experiments.

Model and agent-system inventory

E1 holds the agent implementation fixed and varies the language-model backbone. The canonical implementation is the ReAct-style executable interaction harness used throughout the main benchmark. E2 instead crosses a smaller set of anchor backbones with four agent implementations—Direct, ReAct, OpenCode, and Codex—on the shared subset. OpenCode and Codex are evaluated

Experiment	Evidence role	Story set	Systems / conditions	Primary outputs
E1	RQ1 + RQ2 main evidence	1,044 UR + ∼200 CR	6 backbones, shared ReAct-style harness, Open Goal + Stepwise	DRR, GSR, RPL, IAP, RRA, CR dynamics, Strategy/Value profiles
E2	Architecture robustness / provenance	∼200 UR + 40 CR	2–3 anchor backbones × Direct / ReAct / OpenCode / Codex	paired Layer-A/B effects, Strategy/Value shifts
E3	RQ3 controlled decomposition	shared subset after Goal / Provided-Information eligibility	Open/Given Goal × Stepwise/Provided Information	four-cell Layer-A/B/C results; GCS in Given-Goal cells
E4	RQ2 value-policy intervention	shared subset after Value eligibility	one story-compatible coarse target value per story	TVR, TVS, target–Strategy and coarse–fine Value realization
E5	Human construct validation	∼200 UR + 40 CR	human participants in the matched executable interface	human Layer-A/B profiles and Strategy/Value reference distributions

Table 8: Experimental inventory. E2–E5 use the same frozen story subset before experiment-specific eligibility filtering.

end-to-end as agent systems; their internal planning or tool orchestration is not treated as a single-component ablation. Every compared system is given the same benchmark observation stream and the same externally available environment actions and tools permitted by the corresponding experimental condition.

E1 evaluates the built-in ReAct controller with the six model identifiers shown below. The identifiers are reported exactly as configured in the experiment suite.

Suite	Agent system	Model identifier
E1	`react`	`orcarouter/deepseek/deepseek-v4.1-flash`
E1	`react`	`orcarouter/google/gemini-3.8-flash`
E1	`react`	`orcarouter/openai/gpt-5.6-terra`
E1	`react`	`orcarouter/z-ai/glm-5.3`
E1	`react`	`orcarouter/anthropic/claude-sonnet-5`
E1	`react`	`orcarouter/qwen/qwen3.8-flash`

The current E2 implementation instantiates the OpenCode agent system with two anchor backbones.

Agent system	Model identifier
`opencode`	`orcarouter/deepseek/deepseek-v4.1-flash`
`opencode`	`orcarouter/openai/gpt-5.6-terra`

The built-in ReAct controller uses the benchmark's OpenAI-compatible chat-completions client. OpenCode runs are executed through the OpenCode CLI. The implementation records the model identifiers above, but does not record provider-side snapshot pins, model access dates, or an exact OpenCode CLI version in the experiment suite. Codex is not part of the current E2 suite implementation.

All systems use the same benchmark MCP story interface. The MCP tools and action grounding are specified in Appendix B.4; we do not repeat the tool schemas here. This section instead reports the prompt and agent-system configuration assembled by the runner.

For each story, the runner first constructs a shared benchmark prompt. This prompt is used as the task text for the built-in ReAct controller and is also passed as the command-line prompt argument to the OpenCode `story-benchmark` agent.

```
You are participating in a story benchmark.
The story engine is already running.

Use the benchmark MCP tools
to interact with the story.

First call benchmark_get_world_state.

Before each benchmark tool call,
briefly state why this action is appropriate.

Then play naturally
until the story reaches an ending.

Use the exact entity names shown before the colon
in world state lists when calling tools.

If any benchmark tool returns STORY ENDED
or an ending name, stop immediately.
```

When a cell enables a given goal, the runner appends the following fragment to the same shared prompt.

```
Benchmark condition:

Given goal:
<given_goal>
```

When a cell enables a given value, the runner appends the following fragment to the same shared prompt.

```
Benchmark condition:

Given value:
<given_value>

When choosing among plausible actions,
prefer actions consistent with the given value.
```

Goal and value text are added before agent dispatch. If enabled, the same condition text is received by the built-in ReAct controller and by the OpenCode `story-benchmark` agent. In the current E1 and E2 suite files, the configured cell disables both `use_goal` and `use_value`, so these conditional fragments are not appended for those runs.

The built-in ReAct controller uses the shared benchmark prompt as its user task and adds the following controller system prompt.

```
You are a simple ReAct controller.

Execute benchmark tool actions
according to the user task.
```

The ReAct controller uses native chat-completions tool calls over the benchmark MCP tools. It does not use a separate JSON action schema. Its visible task text is the shared benchmark prompt, optionally with the goal/value fragments above.

For OpenCode runs, the shared benchmark prompt is passed to the `story-benchmark` agent through the following command form.

```
opencode run
  --agent story-benchmark
  --auto
  --format json
  --thinking
  --variant high
  --model <model>
  <shared_benchmark_prompt>
```

The OpenCode `story-benchmark` agent has its own agent prompt.

```
You are testing an interactive story as a player.

Use the benchmark MCP tools
to observe the current story state
and send player inputs or player actions.

Play naturally
until the story reaches an ending.
```

The same OpenCode agent is configured with restricted tool permissions. It may use benchmark MCP tools to play the story, while ordinary workspace and file-inspection tools are denied. This prevents the agent from reading story configuration files, experiment configuration files, or other local artifacts that could reveal hidden benchmark state.

```
Allowed:
  benchmark_get_world_state
  benchmark_send_player_input
  benchmark_send_player_action
  todowrite

Denied:
  bash
  read
  edit
  glob
  grep
  list
  webfetch
  websearch
  task
  lsp
  skill
  external_directory
```

Canonical condition

Unless an experiment explicitly changes a factor, runs use the canonical OPEN GOAL + STEPWISE INFORMATION + PLAIN VALUE condition. The system receives the current textual observation and interaction history but no task-level Outcome Goal and no exogenous Value target. Information is acquired only through the executable interactions made available by the story runtime. The system emits a free-form action that is grounded by the benchmark matcher / resolver. UR runs additionally emit one compact `current_policy` statement immediately before terminal policy commitment. CR runs emit the same field at reached policy-bearing checkpoints, as required for RRA.

The policy report is intentionally compact: it records the policy the system currently intends to enact but does not request chain-of-thought, rationale, or step-by-step deliberation. Report-to-realization scoring is performed in the shared five-family Strategy space defined in Appendix D.3.

Run unit, decoding, budgets, and failures

The experimental unit is one system–story–condition trajectory. Each such cell is run once in E1–E4; no metric relies on repeated stochastic sampling of the same cell. Human E5 trajectories are the only intentionally replicated story-level observations, with a target of three independent participants per story. All automated systems use the same maximum environment interaction horizon within a condition. Provider-level transport retries do not create a new benchmark trajectory; they only recover an interrupted model call. Environment-level invalid or unresolved actions remain part of the trajectory and are reflected in the executable-interaction measurements.

A run terminates when the environment reaches a valid ending, the interaction budget is exhausted, or an unrecoverable execution failure occurs. The scorer distinguishes a non-completing benchmark trajectory from a provider/infrastructure failure according to a frozen run-validity rule. Trajectory inclusion is determined entirely by this run-validity rule.

Each trajectory is bounded by a wall-clock budget of 300 s, applied equally to the built-in ReAct controller and the OpenCode agent. This budget covers the complete interaction, including model request and response latency and all tool calls. No separate model-request timeout, maximum generation length, or client retry count is configured, so these remain client-side defaults and the 300 s budget is the effective outer bound. On the environment side, an individual world-state read is limited to 10 s, a player-turn request to 60 s, an action-result wait to 15 s, and a speak reply to 10 s.

After the interaction ends, the runner waits up to 20 s for the authoritative ending, and a fallback engine-result fetch uses 10 s. The ReAct controller generates with a temperature of 0.2; OpenCode uses its own CLI decoding configuration, which is not part of the benchmark source. A resumed run keeps successes and timeouts as measured results, replays attempts that produced no ending and stories with no row, and a separate flag widens the replay to every non-success attempt; when a story has several rows, the latest one determines its result. The inspected source contains no separate scorer-level inclusion rule.

Logging and score extraction

Every trajectory retains the raw model output, grounded semantic action ID, matcher status, visited semantic state, available / resolved / newly acquired information IDs, information-seeking action attribution, policy report, policy-bearing node IDs, CR update IDs, terminal ending ID, and runtime metadata needed by the deterministic scorer in Appendix D.7. The same frozen trajectory logs are used for all scalar measurements and Strategy / Value analyses.

For Strategy and Value profiles, the denominator is the number of completed trajectories contributing a terminal ending. RRA and SLSS are accompanied by their structural scoring coverage. IAE is accompanied by the no-seek rate. UR and CR results are shown separately for structure-sensitive measurements before any combined summary.

Statistical reporting

All scalar results are reported as story-macro averages using the eligibility rules in Table 7. E1 uncertainty uses story-level bootstrap resampling within the reported UR/CR stratum. E2–E4 use paired story-level bootstrap resampling so that all compared systems or conditions for a sampled story remain together. The E3 2×2 design resamples complete four-cell story blocks. E5 uses the hierarchical bootstrap defined in Appendix D.6, resampling stories first and human trajectories within story. All reported confidence intervals are 95% intervals.

For metrics that are undefined on a structurally ineligible story, analyses use the common eligible story set required by the comparison and report its size. A value of zero is never substituted for structural ineligibility. Detailed Strategy / Value tables report raw support counts in addition to normalized frequencies. The paper emphasizes paired effect sizes and uncertainty; label- level multiplicity adjustment is applied only to families of inferential tests that are interpreted jointly.

E1: Full Open-Goal Benchmark Design and estimands

E1 is the primary standardized comparison of current agents under Open-Goal Interactive evaluation. Six LLM backbones are evaluated under the same ReAct-style harness on all 1,044 UR stories and the final CR set. The design therefore isolates backbone variation while keeping the executable agent interface fixed.

For Layer A, E1 reports DRR, GSR, RPL, and the two-dimensional Information Acquisition Profile $\mathrm{IAP} = (\mathrm{IOC}, \mathrm{IAE})$. For Layer B, it reports RRA for UR and CR, together with SLSS, CTD, and the Strategy Transition Matrix on CR. Layer C reports terminal Strategy and Value profiles and the Strategy–Value association defined in Appendix D.4. Scalar results are presented with 95% confidence intervals and the applicable coverage counts.

Complete Layer-A results

The full E1 table reports each model's DRR, GSR, RPL, IOC, and IAE for the overall benchmark and for UR/CR separately where applicable. RPL is presented as an execution / interaction-depth profile rather than as a monotonic quality score. IOC and IAE are shown jointly: IOC captures how much of the information available along the realized path is acquired, whereas IAE captures new information gained per active seek action.

Policy-report validity and CR policy dynamics

RRA evaluates whether the system's compact reported policy is realized by the subsequent policy-bearing Strategy. CR additionally exposes within-trajectory policy dynamics through SLSS and CTD. The Strategy Transition Matrix retains the direction of those changes and is therefore reported alongside the two scalar CR summaries.

Realized Strategy and Value profiles

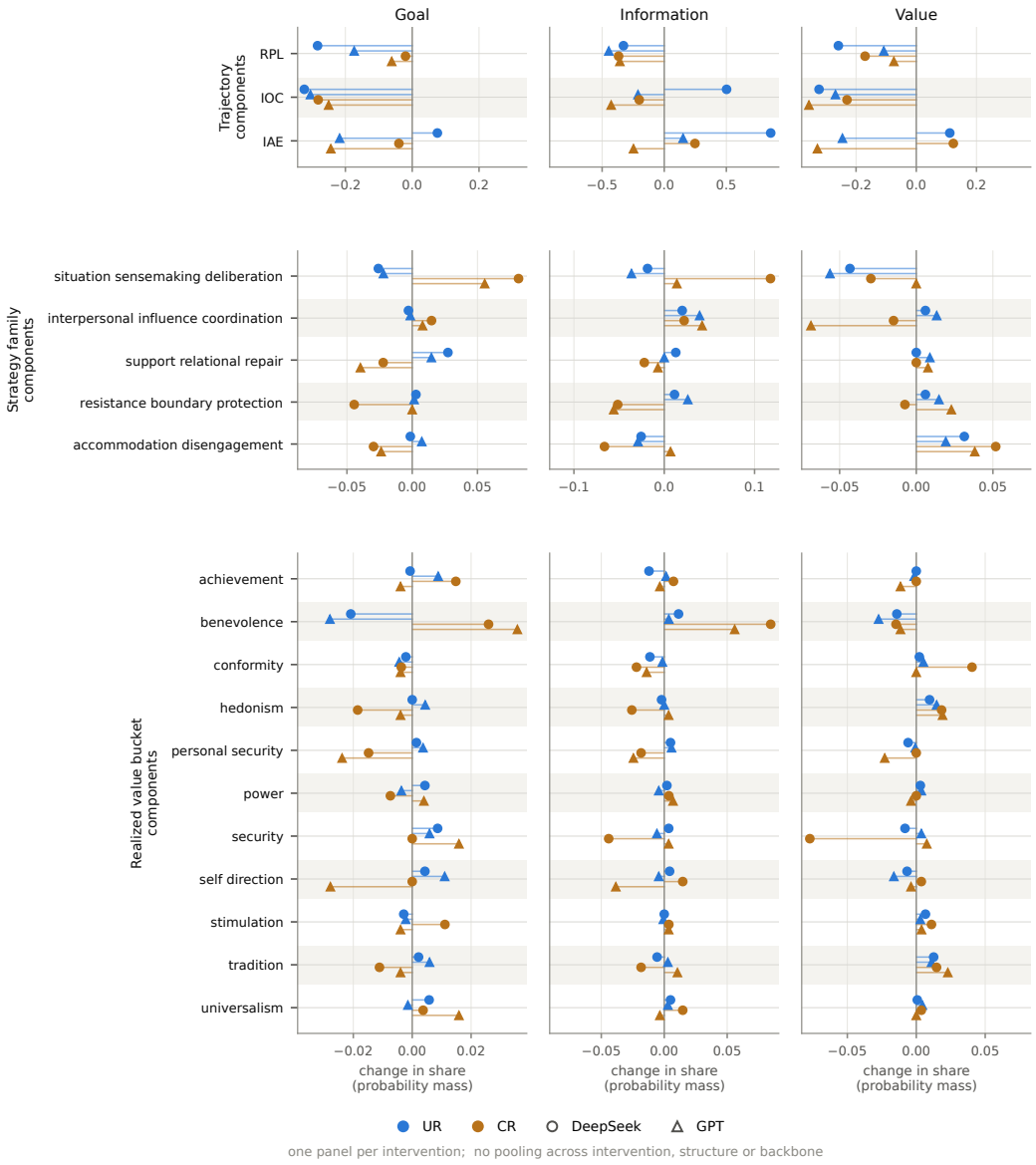

Figure 15: **Component decomposition of the trajectory footprint.** Each panel reports the matched change in one trajectory component — relative path length (RPL), information opportunity coverage (IOC), and information acquisition efficiency (IAE) — under the Goal, Information, and Value interventions, shown separately for UR and CR and for the two anchor systems. Points are point estimates with 95% confidence intervals; one panel per intervention, with no pooling across intervention, structure, or backbone. This decomposes the aggregate displacement in Fig. 7.

The main policy characterization uses the five Strategy families and the 11-category coarse Value space. The appendix additionally reports all 25 fine Strategy labels plus the residual category and all 58 fine Value labels. These are completion-conditioned distributions, not capability scores.

Structural heterogeneity

Extended E1 analysis reports the primary measurements by major story family and by UR/CR structure. These breakdowns are used to localize aggregate differences, not to define additional benchmark scores.

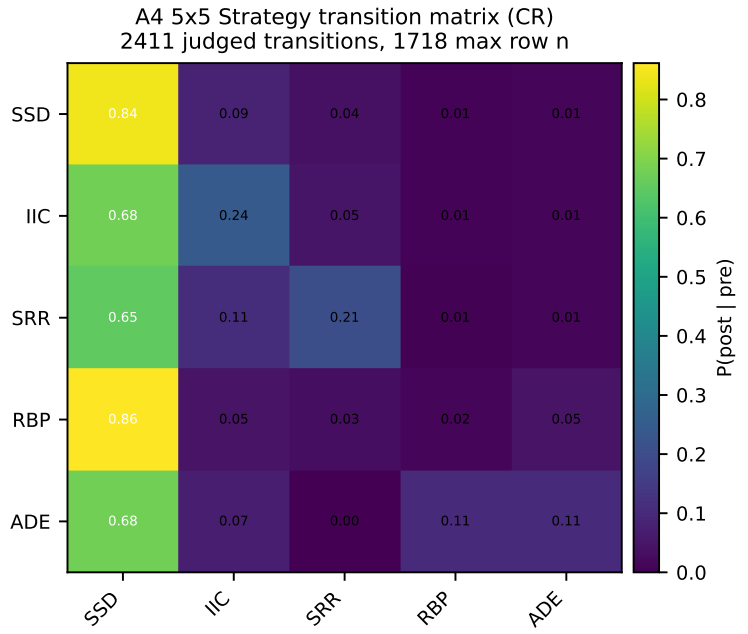

Figure 16: **CR Strategy transition matrix.** Row-normalised transition probabilities over the five coarse Strategy families in CR, estimated from 2,411 judged transitions. Rows are the pre-update family and columns the post-update family; each row is conditioned on the pre-update family, and the maximum row count is 1,718. Diagonal cells measure persistence across a registered state update; the Self-Directed Search row is the most persistent, at 0.84.

E2: Agent-Architecture Robustness and Provenance **Design**

E2 evaluates whether the E1 agency profile is robust to agent implementation and separates backbone effects from harness effects. Two to three anchor backbones are crossed with Direct, ReAct, OpenCode, and Codex on the shared approximately 200-UR/40-CR subset. Open Goal, Stepwise Information, and Plain Value are held fixed. The story set and runtime affordances are identical within each backbone–architecture comparison.

ReAct is the controlled reference because it is the canonical E1 harness. For anchor backbones whose ReAct configuration is identical to E1, the E2 ReAct trajectories on the shared subset are reused directly from E1. For a scalar metric M, the architecture effect for architecture a and backbone m is reported on the metric's common eligible story set:

$$\Delta^{\text{arch}}_{m,a}(M) = \frac{1}{|\mathcal{S}^{\text{pair}}_{m,a,M}|} \sum_{s \in \mathcal{S}^{\text{pair}}_{m,a,M}} [M_{m,a,s} - M_{m,\text{ReAct},s}]\,. \tag{35}$$

Coverage differences are shown separately rather than absorbed into the paired effect.

Full architecture grid

Architecture-induced policy shifts

E2 additionally reports Strategy and Value profiles for each backbone–architecture combination. The objective is to determine whether agent architecture changes the realized social-policy distribution even when the underlying backbone is held fixed.

E3: Goal Specification × Information Access **Eligible story set and four-cell protocol**

E3 directly tests RQ3 on the shared subset after applying the Given-Goal and Provided-Information eligibility rules in Appendix D.5. The primary four-cell analysis uses the intersection of stories for which both a valid primary Outcome Goal and a valid Provided-Information packet are frozen. Each

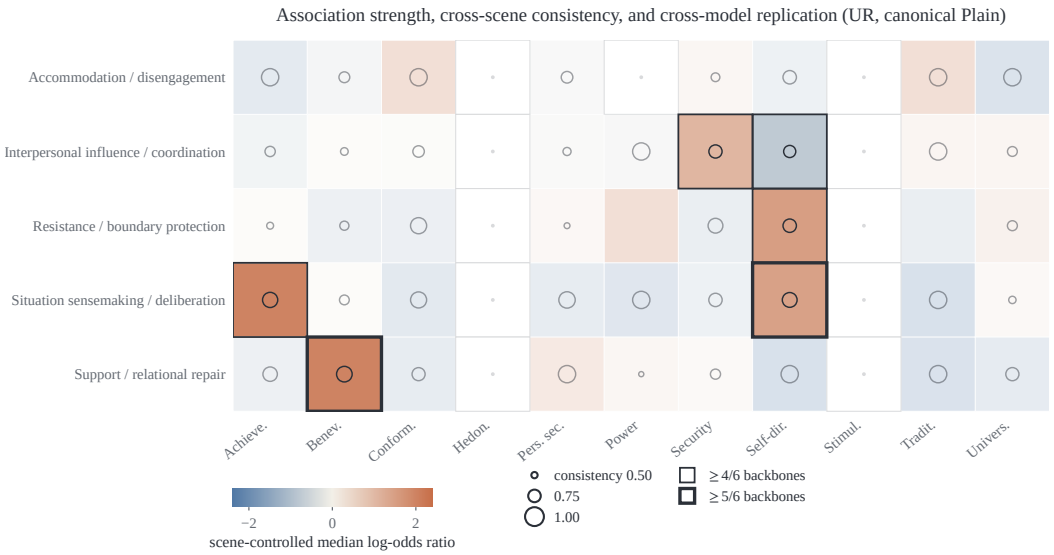

Figure 17: **Stable Strategy–Value modes in the canonical condition.** Association between the five coarse Strategy families and the eleven coarse Value domains in UR under the canonical Plain condition, on the scene-controlled median log odds-ratio scale. Marker size encodes the proportion of scenes in which the sign agrees, and boxed cells mark associations replicating across at least four of the six backbones. The flagship Support–Benevolence mode replicates in 6/6 backbones.

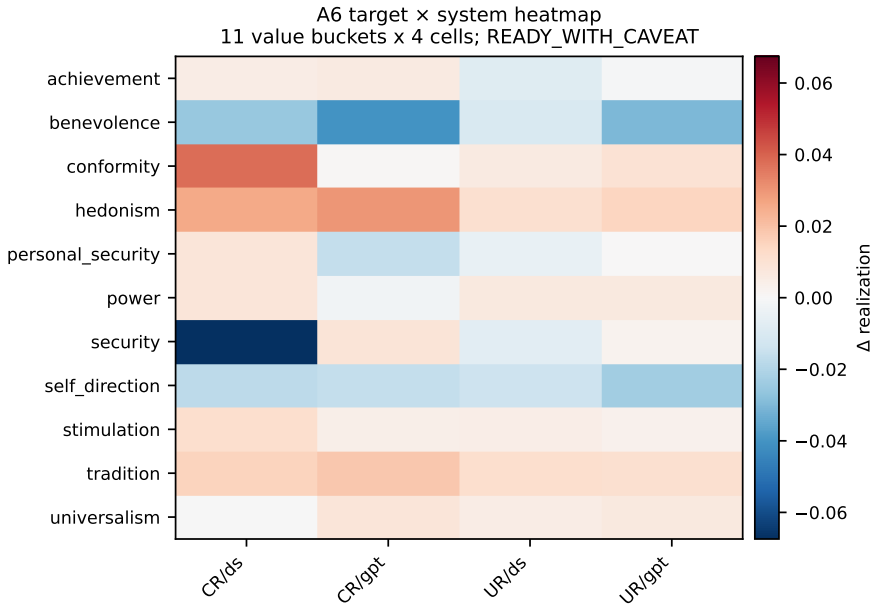

Figure 18: **Target–system Value realization.** Change in realized Value share for each of the eleven coarse Value domains under each conditioning cell, shown per anchor system. Steerability is concentrated in specific target–system cells rather than spread broadly across the Value space.

anchor system is evaluated once per story in each cell:

	Stepwise Information	Provided Information
Open Goal	C_{00}	C_{01}
Given Goal	C_{10}	C_{11}.

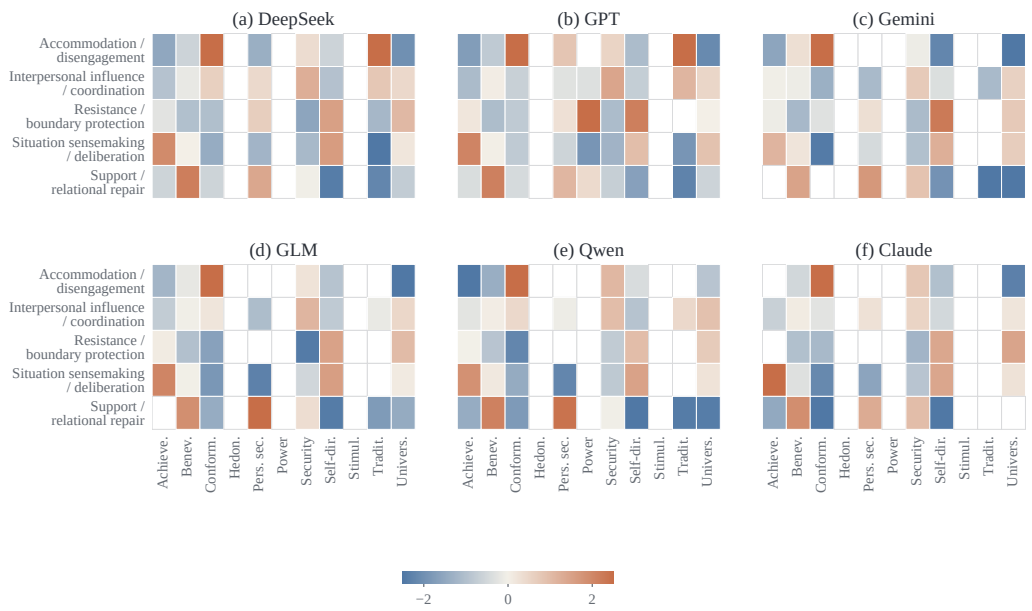

Figure 19: **Strategy–Value structure within each backbone.** The canonical Plain-arm association between the five coarse Strategy families and the eleven coarse Value domains, shown separately for each of the six backbones (log odds ratio). Agreement across panels is what licenses treating the association as cross-model structure rather than a single-model artifact.

All factors other than Goal Specification and Information Access are held fixed. The C_{00} cell is the canonical E1 condition and is reused from E1 whenever the anchor system configuration is identical. Given-Goal runs receive the same frozen story-level Outcome Goal in both information conditions. Provided-Information runs receive the same factual packet for Open- and Given-Goal conditions.

Factorial estimands

For a scalar metric M defined in all four cells, let M_{gi} denote the story-macro estimate in cell C_{gi}, with $g \in \{0, 1\}$ indexing Open/Given Goal and $i \in \{0, 1\}$ indexing Stepwise/Provided Information. We report

$$E_G(M) = \frac{(M_{10} - M_{00}) + (M_{11} - M_{01})}{2}, \tag{36}$$

$$E_I(M) = \frac{(M_{01} - M_{00}) + (M_{11} - M_{10})}{2}, \tag{37}$$

$$E_{G\times I}(M) = (M_{11} - M_{10}) - (M_{01} - M_{00}). \tag{38}$$

These contrasts are applied to DRR, GSR, RPL, RRA, and any other scalar whose eligibility is well-defined in all four cells. IOC and IAE characterize active acquisition and are therefore interpreted only in Stepwise conditions. Their Goal effect is evaluated by the same-story $C_{10} - C_{00}$ contrast rather than forcing acquisition scores onto Provided-Information runs.

GCS is defined only under Given Goal. The information-access effect on goal-conditioned success is

$$\Delta_{\text{Info}}\text{GCS} = \text{GCS}_{11} - \text{GCS}_{10}. \tag{39}$$

For CR, SLSS and CTD are reported by cell with the same eligibility and coverage conventions as E1.

Complete four-cell results

Policy-distribution changes across cells

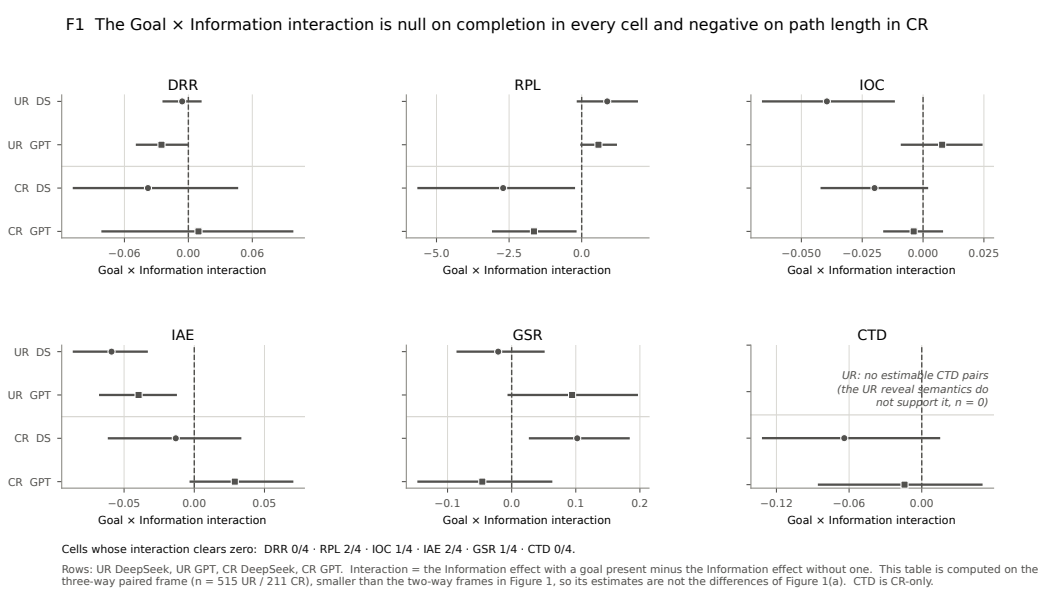

Figure 20: **Goal × Information interaction across cells.** The interaction between Goal specification and Information access for terminal realization, computed separately within every structure and cell. The interaction is null in each cell, which is why the joint Goal+Information arm is reported as a compositional diagnostic rather than as an additional source of agency.

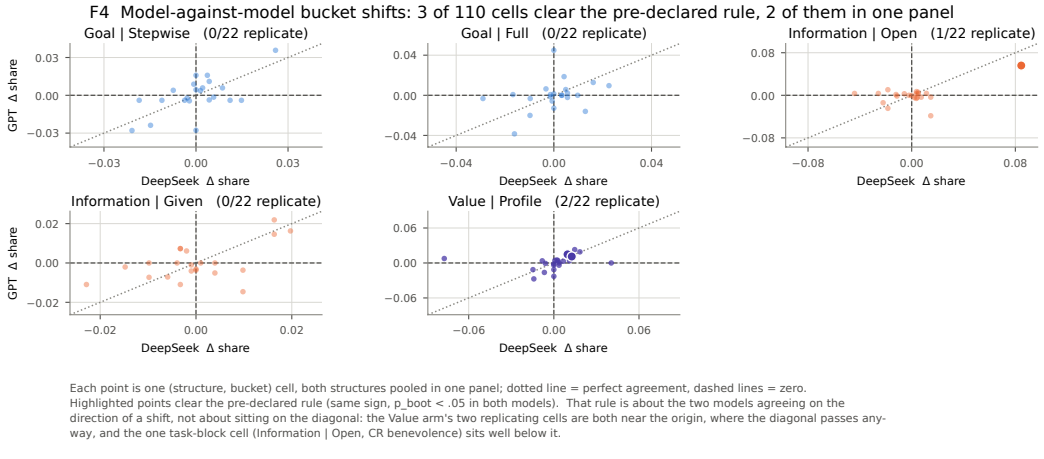

Figure 21: **Model-against-model replication of terminal shifts.** Each point is one structure-by-bucket cell, with structures pooled within a panel. The dotted line is perfect agreement and the dashed lines mark zero. A shift must have the same sign in both models; three of 110 cells clear that rule, two of them in one panel.

Strategy and Value profiles are compared across the four cells on matched stories. These analyses test how supplying the Goal and/or the information basis changes the policy that the same system realizes, rather than treating policy-profile movement as intrinsically better or worse.

E4: Story-Conditioned Value Steering Story-specific target assignment

E4 begins from the shared experimental subset and retains stories with non-empty $\mathcal{V}_s^{\text{eligible}}$ as defined in Equation 29. Conditioning uses the 11-category coarse Value space (Schwartz 10 plus Personal Security). Each eligible story receives exactly one target v_s^* that is realizable by at least one reachable ending and absent from at least one alternative ending. The same assigned target is used for all

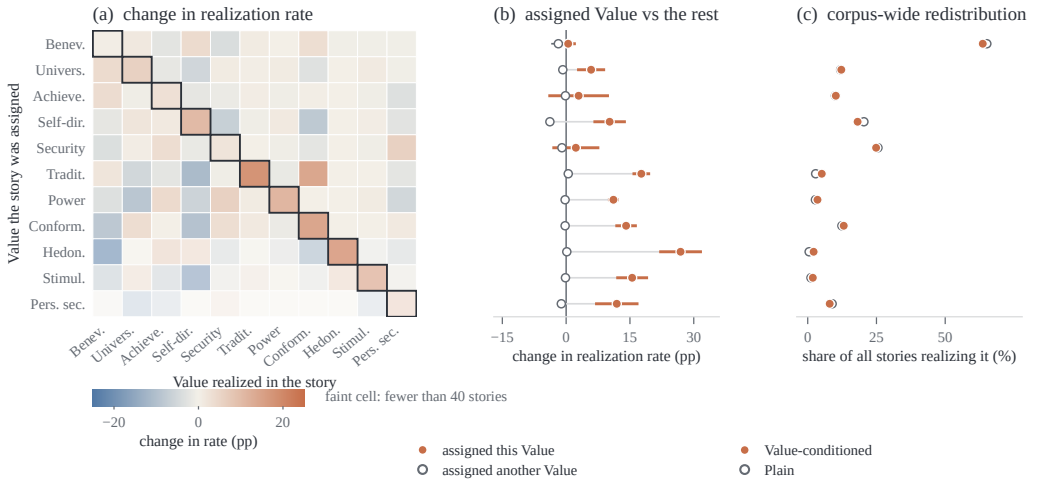

Figure 22: **Target-specific Value realization.** (a) Change in realization rate for each assigned Value (rows) against each realized Value (columns); the diagonal is target-local following, and faint cells indicate fewer than 40 contributing stories. (b) For each assigned Value, the change when it is the target compared with the change for Values assigned elsewhere (placebo). (c) Corpus-wide share of stories realizing each Value under the Value-conditioned and Plain arms.

compared systems on that story. Target assignment is balanced across coarse values subject to the story-specific eligibility graph and is frozen before conditioned trajectories are generated.

The 58 fine Value labels are not used as intervention targets. They are used after execution to characterize how a coarse target is concretely realized.

Plain and conditioned runs

For each system–story pair, E4 compares one Plain trajectory with one Value-conditioned trajectory. When the system configuration is identical to E1, the Plain trajectory is reused from E1; otherwise a matched Plain run is generated under the E4 configuration. The Value-conditioned prompt adds only the assigned target value specification; the story, Stepwise information process, action interface, interaction budget, and all other runtime settings remain unchanged.

TVR and TVS results

TVR and TVS are defined in Equations 31 and 34. Results are reported overall and separately for every coarse target with adequate support, together with the number of assigned stories.

How target values are behaviorally realized

E4 uses Layer C to characterize the mechanism of value steering at the policy level. A Target-Value $\times$ realized-Strategy table shows which Strategy families and fine strategies are associated with each assigned value under the conditioned runs. A second matrix reports $P(v_{\text{fine}} \mid v_{\text{target}}^{\text{coarse}})$, using the frozen 58-value terminal annotations.

Interaction and policy-process side effects

The conditioned and Plain trajectories are also compared on DRR, GSR, RPL, IOC, IAE, and RRA. CR stories additionally permit paired SLSS and CTD comparisons. These measurements show whether value steering changes interaction execution, acquisition behavior, or reported-to-realized policy correspondence while altering the terminal policy distribution.

E5: Human Behavioral Reference Participants and assignment

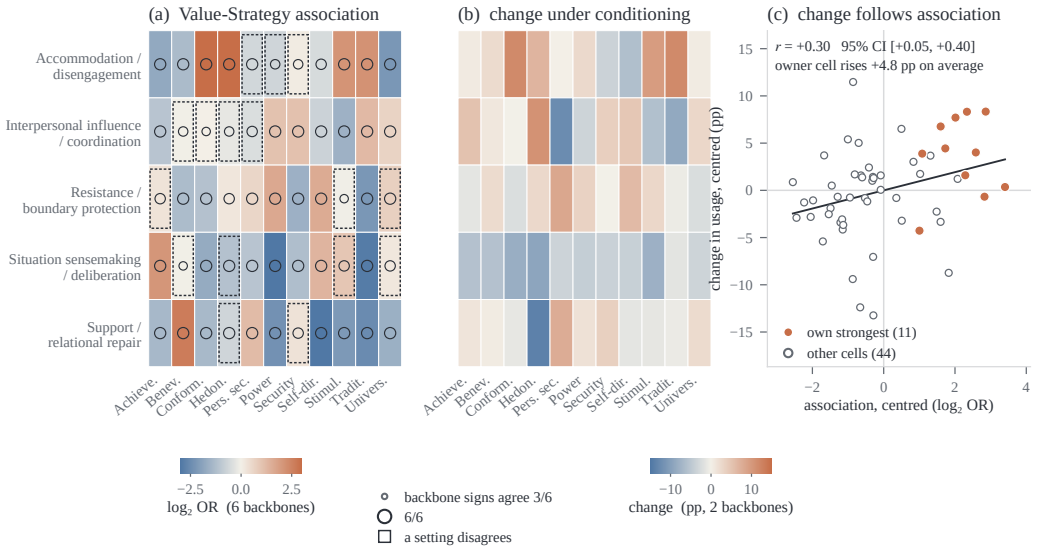

Figure 23: **Value-specific Strategy steering follows the pre-existing association.** (a) Canonical association between Strategy families and Value domains, with marker size recording the number of agreeing backbones and boxed cells marking settings that disagree. (b) Change in Strategy usage under Value conditioning, with global Strategy and Value shifts removed. (c) The centred conditioning change against the centred canonical association: $r = +0.300$ (95% CI $[+0.05, +0.40]$). The corresponding CR interval includes zero.

E5 uses the same approximately 200-UR/40-CR shared subset as E2–E4. Human participants interact with the textual executable environment under the canonical Open Goal + Stepwise Information + Plain Value condition. They do not see Strategy labels, Value labels, story graphs, hidden state, or evaluator annotations. Actions are entered in free form and pass through the same matcher / resolver interface used for agent runs. Participants also provide the same compact `current_policy` field at the registered UR/CR checkpoints.

The target design collects three independent human trajectories per story. Story assignment is balanced across participants, and no participant receives the same story more than once. Participant-level inclusion rules are applied before comparing humans with agents; story-level measurements retain the hierarchical dependence induced by multiple participants on the same story.

Human measurement profile

Human trajectories are scored by the same deterministic pipeline wherever the measurement is applicable. We report DRR, GSR, RPL, IOC, IAE, and RRA; CR trajectories additionally contribute SLSS and CTD. Terminal Strategy and Value annotations yield human reference distributions in the same coarse and fine spaces used for agents.

Human–agent behavioral comparison

Human data provide a behavioral reference distribution, not a normative target. Comparisons therefore report where agent measurements and realized policy distributions lie relative to observed human variation on the same stories. Scalar comparisons preserve story matching. Strategy and Value comparisons use the same story-level hierarchical resampling scheme as the human scalar analysis.

Cross-Experiment Reporting and Reproducibility Main-paper and appendix result map

The main paper reports the minimum set of artifacts needed to answer the three research questions. The appendix contains the complete system grids, eligibility counts, fine-grained policy distributions, and subgroup analyses.

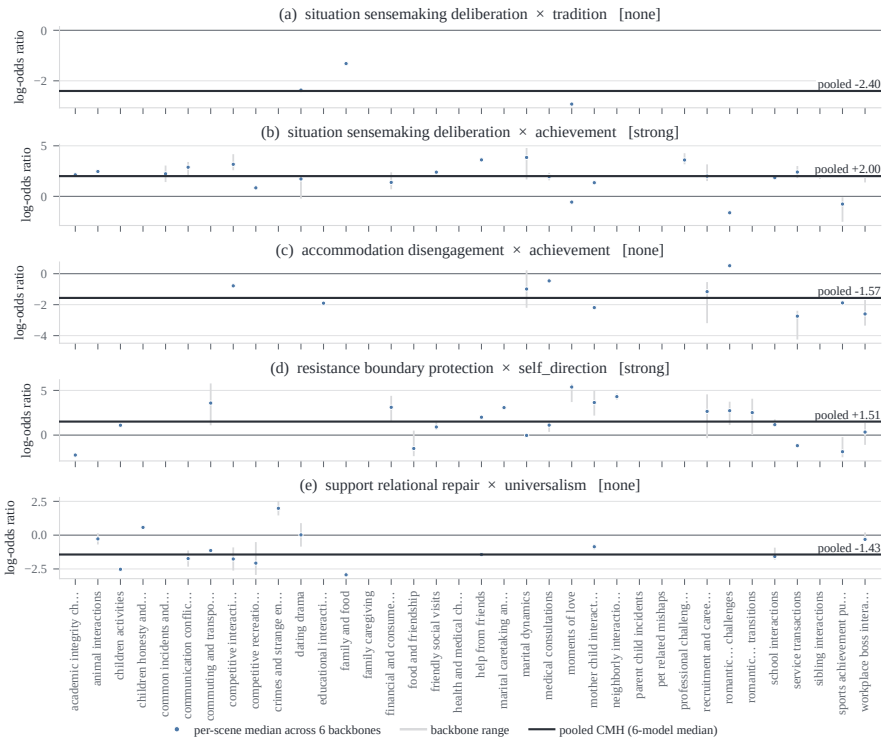

Figure 24: **Strategy–Value stability under leave-one-scene-out.** Five representative Strategy–Value pairs, each recomputed with one scene subtype withheld. Points are per-scene medians across the six backbones, the thin line is the range across scene bases, and the heavy line is the pooled conditional mutual information estimate. No single scene family carries the reported structure.

Paper claim / role	Experiment	Main-paper artifact	Appendix expansion
RQ1: Open-Goal Social Agency	E1	full-benchmark summary + IAP / RPL	all metrics, UR/CR splits, coverage, heterogeneity
RQ2: Policy Formation & Dynamics	E1	RRA, SLSS/CTD, policy landscape	transition matrices, fine Strategy/Value, association tables
Architecture robustness / provenance	E2	paired architecture summary	complete backbone × architecture grid
RQ3: Beyond Goal-Conditioned Evaluation	E3	2 × 2 contrasts + GCS	all four cells, eligibility, policy shifts, CR results
RQ2 value intervention	E4	TVR/TVS + target–Strategy analysis	per-target counts, fine realization, side effects
Construct validation	E5	human reference summary	full human distributions and story-matched comparisons

Table 9: Mapping from paper claims to main and appendix experimental evidence.

Operational cost and failure accounting

For reproducibility, the final release reports the number of attempted and valid trajectories per experiment, model-call counts, token use where available, wall-clock runtime, and API cost for paid providers. Infrastructure failures and provider retries are counted separately from benchmark non-completion and action-grounding failures.

Released experiment artifacts

The reproducibility package should contain the frozen E1 corpus IDs, E2–E5 shared-subset IDs, E3 eligibility and sidecar files, E4 target assignments, model and harness configurations, complete raw

trajectory logs, deterministic score outputs, bootstrap seeds, and the scripts used to generate every table and figure in the paper.

F RELEASE, REPRODUCIBILITY DETAILS, AND RESPONSIBLE USE

This section indexes the released artifacts and their version identifiers, states the configuration needed to reproduce the reported experiments, and records the responsible-use and limitation considerations that support the Reproducibility and Ethics Statements in the main paper. The reproducibility evidence itself is distributed across Appendices B–E; the material below serves as the release, version, and configuration index for that evidence and collects the detailed responsible-use considerations that are too long for the main text.

Released Artifacts and Versioning **Artifact Inventory** We release the environment implementation, the frozen story specifications and their executable configurations, the benchmark manifests and eligibility filters, the annotation taxonomy with its frozen prompts, the metric and scorer implementations, the experiment drivers, and the analysis scripts that produce the reported tables and figures. We release the frozen benchmark as a *probe* set: the story specifications, configurations, and annotations are public, while the held-out evaluation content used for leaderboard-style comparisons is not. Human trajectories are not released, because consent was obtained for analysis rather than redistribution; aggregate human summaries sufficient to reproduce the reported comparisons are released instead.

Version Identifiers Each released component carries its own version identifier, so that a future update to one component can be distinguished from the version used in this paper. The version identifiers are recorded in the repository's release manifest and are frozen at the artifact tag corresponding to this submission. We separate the environment, the benchmark, the annotation taxonomy and prompts, the scorers, and the analysis code, because a benchmark result is only interpretable relative to the specific versions of the generator and the annotator that produced it.

Benchmark Maintenance Policy PRAXISCENE is released as a frozen benchmark rather than a continuously updated one. The frozen release is the object of record for the results in this paper, and we do not modify it in place. Any subsequent extension is released as a new version, with the previously frozen version retained, so that reported numbers stay comparable within a version. We do not claim comparability across versions; where a change would alter an annotation, a scorer, or an eligibility rule, the version identifier changes.

Reproduction Configuration **Model and Runtime Configuration Index** Reproducing E1–E4 requires the model identifiers and access dates, the frozen prompts, the decoding parameters, the interaction budgets, and the package and runtime versions. These are recorded with the released artifacts rather than reproduced here, and the model and runtime settings needed for the reported experiments are specified alongside the experiment drivers. The benchmark itself is model-agnostic: the story specifications and configurations do not depend on the system being evaluated, so the same frozen benchmark can be reused with a different harness.

Data, Subset, and Scorer Availability The frozen benchmark, its manifests, the annotation taxonomy, and the scorer implementations are public. Tables and figures that report trajectory-level aggregates can be reproduced from released trajectories once a user generates them; the analysis scripts consume the trajectory record format documented in Appendix B.2 and do not depend on undisclosed intermediate state. Rerunning a system from scratch requires only the frozen benchmark and the system's own model access.

Known Reproducibility Constraints Exact bitwise reproduction is not available for closed-weight models, whose served versions may change without notice, and is not guaranteed for any sampling-based system, because decoding involves provider-side nondeterminism. Two further constraints follow from the design. First, some measurements are completion-conditioned and are therefore undefined for trajectories that reach no ending, so reproducing them depends on reproducing the ending rate as well. Second, the human reference is a fixed sample, and its confidence intervals reflect that sample rather than the population. Our reported intervals therefore separate procedural reproducibility, which the artifacts support, from provider-level stochasticity, which they cannot eliminate.

Responsible Use and Human-Data Considerations **Normative Interpretation Boundary** Strategy and Value labels are descriptive categories, not a moral ranking. A realized profile records what an action observably does in a situation, and it does not license reading one Value or one Strategy family as a better outcome than another. In particular, the human reference is a behavioral comparison, not a gold policy: agreement with the human majority is not evidence of correctness, and divergence from it is not evidence of failure. The benchmark therefore supports claims about which behavioral components an agent can realize, not claims about the social desirability of its resolutions.

Social and Cultural Coverage Risks The situations are generated from a fixed scene and value taxonomy authored in the language and social conventions of a single cultural context, and they emphasize a bounded set of interpersonal settings, relationship types, and conflicts. Coverage of social worlds outside that context is incomplete, and the generated content may reproduce conventions or associations present in the authoring models, including stereotyped role distributions. A profile obtained on this benchmark is therefore evidence about behavior within its coverage, not about social competence in general.

Human Data, Consent, and Privacy Participation was voluntary and compensated, and participants were informed before taking part that their interaction trajectories would be recorded and analyzed. The study was reviewed within our research department, which determined that formal ethics-board approval was not required. We collect no identifying information and do not link any recorded trajectory to a participant identity; all reported human results are aggregate, and trajectories are not released. Because the situations are synthetic rather than drawn from participants' own lives, the privacy risk of the recorded content is limited to what a participant chose to type into a fictional scenario.

Deployment and Misuse Boundary PRAXISCENE is an evaluation benchmark. It is not a certification that a system is safe, appropriate, or competent for unrestricted social deployment, and a high score does not transfer to open-ended real interaction. The most plausible misuse is over-reading the scores as a general measure of social acceptability, or optimizing a system against the frozen benchmark in ways that improve its measured profile without improving the behavior the measurement aims to characterize. Because the benchmark is public and its ending and annotation spaces are fixed, benchmark-specific overfitting is possible, which is one reason we release it as a frozen version rather than a moving target.

Detailed Limitations **Environment and Benchmark Limitations** Stories are bounded programs: each is a fixed graph with a limited number of interactive scenes, choices, and endings, so an agent's behavior is observed within a small, authored state space rather than in an open environment. The content is generated, and although candidates are filtered by structural validation and admission tests, generated situations may contain implausible details, uneven cultural grounding, or weak social motivation. Coverage of relationships, institutions, and norms is incomplete, and every annotation depends on the frozen Strategy and Value taxonomies, so behavior that falls outside those categories is not represented.

Measurement Limitations The benchmark measures identifiable components of social agency rather than a single latent construct, and no scalar score over them is reported: the observables disagree, and that disagreement is the finding rather than an artifact to be averaged away. Several measurements are defined only on trajectories that reach an ending, so they are conditioned on a successful completion and are undefined otherwise, and information-acquisition quantities are undefined when Information is supplied. Strategy–Value association is observational: it describes how realized policies co-occur, not what causes them. Finally, Strategy and Value labels are produced by a language-model judge, and although value labels are separately validated against human annotation, the annotator and the evaluated systems share a model family in part, so agreement between them cannot be treated as fully independent evidence.

Experimental Limitations The canonical protocol runs one trajectory per system–story pair, so estimates reflect a single sample of each system's behavior rather than a distribution over repeated runs; repeated-run variance is therefore not measured for the full corpus. The controlled analyses in E3 and E4 operate on targeted subsets rather than the full benchmark, and their estimates apply to the eligible sets defined in Appendix D.5. The architecture comparison covers two anchor systems, because only those were run under more than one harness. The human reference is a modest sample

and is matched by story rather than by participant, and the human and agent interfaces are closely matched but not identical. Where an effect is estimated on a small number of story clusters, the reported intervals are correspondingly wide, and we treat those estimates as provisional rather than as stable properties of the systems.