# CalBench: Evaluating Coordination–Privacy Trade-offs in Multi-Agent LLMs

**Chelsea Zou*** **Yiheng Yao*** **Selena She** **Noah D. Goodman** **Robert D. Hawkins**
Stanford University
{cyzou, yaoyh, jshe, ngoodman, rdhawkins}@stanford.edu



## Abstract

Personal AI assistants are beginning to act as delegates with access to calendars, inboxes, and user preferences. Calendar scheduling makes the trust problem concrete: an assistant must coordinate with other assistants while deciding what to reveal about the person it represents. We introduce CalBench, a controlled benchmark for multi-agent calendar scheduling under private information. In each task, $N$ agents manage separate private calendars and schedule a stream of $M$ incoming meetings while minimizing disruption costs. Because no agent can inspect another agent's calendar, success requires language-mediated coordination rather than centralized planning. CalBench generates solvable scenarios with CP-SAT oracle solutions and decentralized non-LLM reference protocols, enabling evaluation of task success, excess cost, communication efficiency, burden fairness, and privacy leakage under matched information constraints. Across seven model families, we find that completion alone misses important failures: agents leave avoidable cost on the table, communication volume does not predict lower regret, and privacy-preserving silence can deprive teammates of cost information needed for fair burden allocation. CalBench provides a reproducible testbed for studying whether autonomous assistants can coordinate on behalf of users before deployment at scale. All traces and code can be found at https://d3mern3a2mjjur.cloudfront.net/leaderboard and https://anonymous.4open.science/r/calbench2026-235F/README.md.

## 1 Introduction

Calendar scheduling is a practical test of delegated agency. A scheduling assistant represents a person with private constraints, preferences, and commitments; when it negotiates with other assistants, it must act on that person's behalf without exposing the whole calendar or making unreasonable changes. This raises a new challenge: can assistants turn private user state into shared commitments without leaking too much or imposing avoidable costs on the people they represent?

Calendar scheduling is also *structurally* multi-agent. Each participant holds private information about availability, preferences, and commitments, and no single party has full visibility or authority to impose a solution. Yet the research community lacks rigorous benchmarks for *necessarily* multi-agent tasks with precise rewards. Many existing benchmarks do not cleanly isolate decentralized coordination: even when a task involves multiple actors, a centralized agent with full problem state can often solve it directly (Khatua et al., 2026). Calendar scheduling differs structurally. Each agent's calendar is private and cannot be disclosed without violating the privacy assumptions that make the task realistic; the multi-agent setup is the mechanism by which agents selectively share enough information to coordinate while withholding irrelevant private details.

We present **CalBench**, a benchmark for multi-agent calendar scheduling that generates scenarios with oracle solutions via CP-SAT (Perron and Didier, 2025) and non-LLM reference protocols under the same private-information constraints. Agents coordinate through configurable cheap-talk channels (Crawford and Sobel, 1982) and independently commit scheduling actions; the environment, phase structure, and evaluation metrics are summarized in Figure 1. The harness supports three communication conditions: private agent-to-agent DMs, meeting-participant groupchat, and all-agent groupchat. We evaluate seven models on a mixed-agent benchmark suite. Our contributions are: (1) a multi-agent calendar scheduling benchmark with precise, computable evaluation criteria; (2) a scenario-generation harness that produces con-

*Equal contribution.

trollable, solvable tasks with known oracle metrics and CP-SAT feasible-fraction difficulty buckets; and (3) analysis of seven models covering task success, coordination quality, communication efficiency, fairness, privacy leakage, and recurrent language-level failure modes.

## 2 Related Work

### 2.1 Agentic and multi-agent LLM benchmarks

Recent benchmarks evaluate LLM agents in interactive settings, including tool use and decision-making (Liu et al., 2025), tool-agent-user interaction (Yao et al., 2024; Barres et al., 2025), and natural-language planning with full task context (Zheng et al., 2024). Multi-agent benchmarks study social intelligence, collaboration, competition, and communication structure (Zhou et al., 2024; Zhu et al., 2025; Mahmud et al., 2025), with recent work showing that multi-agent systems can underperform single-agent baselines (Khatua et al., 2026; Davidson et al., 2025; Garcia et al., 2025). Partial-information work such as CRAFT further shows that communication skill need not translate into collective success (Nath et al., 2026). CalBench extends this line with a verifiable scheduling benchmark combining private information, oracle solutions, cost-sensitive objectives, fairness, and trace-level communication analysis.

### 2.2 Agents as user delegates

Recent work frames assistant agents as delegates in a principal-agent relationship: they must complete a task while exercising care, loyalty, and confidentiality toward the user they represent. SocialReasoning-Bench applies this framing to calendar coordination and marketplace negotiation, measuring whether an agent secures value for its user and follows a reasonable decision process (Payne et al., 2026). CalBench studies a different failure surface. Rather than a single assistant negotiating with a fixed counterparty, CalBench places several assistants in a shared scheduling problem where each agent has private state, no agent has global authority, and agreement must remain consistent across all affected calendars. This adds group-level metrics for feasibility, excess disruption, burden allocation, and privacy leakage through the language used to coordinate.

### 2.3 Negotiation and bargaining

Negotiation under private utilities is closely related to CalBench. Prior work studies neural negotiators with private rewards (Lewis et al., 2017), strategic dialogue in Diplomacy (Meta Fundamental AI Research Diplomacy Team (FAIR) et al., 2022), and LLM negotiation benchmarks over bargaining, trading, and multi-issue games (Bianchi et al., 2024; Abdelnabi et al., 2024). Meeting scheduling has also been framed as negotiation among autonomous agents (Renting et al., 2024). CalBench differs by treating scheduling as cooperative coordination over temporal constraints: agents must reach agreement, minimize disruption relative to an oracle optimum, and maintain consistent commitments.

### 2.4 Distributed constraint optimization and calendar scheduling

Calendar scheduling is a distributed constraint optimization problem, where agents coordinate over private calendars, participant constraints, and global utility (Maheswaran et al., 2004; Enembreck and Barthès, 2012; Modi and Veloso, 2004). Prior work includes DiMES for distributed multi-event scheduling (Maheswaran et al., 2004), MULBS for collaborative meeting scheduling (Enembreck and Barthès, 2012), and rescheduling models where later high-priority meetings may displace prior commitments (Modi and Veloso, 2004). CalBench adopts this DCOP framing but uses a streaming regime: meetings arrive sequentially against calendars already filled with earlier commitments. We therefore compare LLM agents against decentralized non-LLM reference protocols and use a full-information global oracle to define optimal cost (§3.1).

### 2.5 Privacy in distributed coordination

Privacy is central to distributed scheduling, yet negotiation can still leak private information (Greenstadt et al., 2006; Brito and Meseguer, 2008; Farhadi and Jennings, 2021). We use Valuations of Possible States (VPS) (Maheswaran et al., 2006) as our numeric-disclosure metric (§3.2). LLM-agent systems introduce further privacy risks under contextual integrity (Nissenbaum, 2004), including leaks through assistants, action traces, tools, prompt injection, inter-agent messages, shared memory, and tool arguments (Mireshghallah et al., 2024; Shao et al., 2025; Bagdasarian et al., 2024;

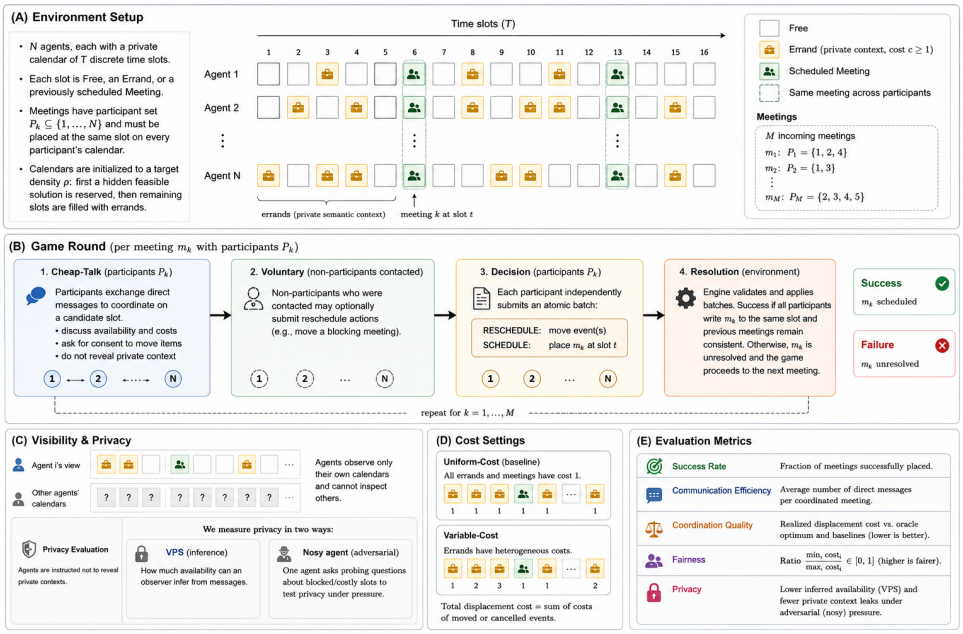


Figure 1: Overview of the CalBench environment. **(A) Environment setup:** Each of $N$ agents maintains a private calendar with $T$ discrete time slots containing free times, errands with private semantic contexts, and scheduled meetings. Meetings must be placed at the same slot across all participants' calendars. Calendars are initialized by reserving a hidden feasible solution and filling remaining slots to a global or per-agent target density. **(B) Game round:** Each meeting is processed through four phases: cheap-talk over private DMs, participant groupchat, or all-agent groupchat; voluntary rescheduling by contacted non-participants; independent decision submission via atomic action batches; and environment-level resolution and validation. **(C) Visibility and privacy:** Agents observe only their own calendars. Privacy is evaluated through inference-based leakage (VPS), with semantic-context leakage audited separately in the appendix. **(D) Cost settings:** We study both uniform-cost and variable-cost environments, where errands may carry heterogeneous displacement costs. **(E) Evaluation metrics:** Runs are scored by scheduling success, excess displacement cost relative to the oracle, communication volume, burden fairness across agents, and privacy leakage.

Debenedetti et al., 2024; Yagoubi et al., 2026). CalBench measures both *utility leakage* through availability and cost disclosures, and *semantic leakage* through unnecessary event details. This targets multi-agent contextual privacy (Juneja et al., 2025), not just output-only disclosure.

## 3 The CalBench Environment

Each CalBench game consists of $N$ agents, $T$ discrete time slots, and a stream of $M$ incoming meetings. Agent $i$ maintains a private calendar whose slots may be free, occupied by a movable errand, occupied by a previously scheduled meeting, or blocked by an immovable commitment. Occupied entries carry a displacement cost and a private semantic context visible only to the owning agent. Each incoming meeting $k$ has a participant set $P_k \subseteq \{1, \ldots, N\}$ and must be written to the same slot on every participant's calendar, so agents must coordinate from local views and received messages rather than from a shared global state.

**Calendar generation.** The task generator first samples a hidden feasible witness assignment for the incoming meetings. It then fills calendars with errands while preserving enough free absorbing slots for the witness schedule to remain feasible. Calendar *density* controls the fraction of each calendar initially occupied by errands. Some calendar commitments are immovable (blocked) so agents must distinguish flexible events from hard constraints and avoid schedules that rely on impossible displacement. CalBench supports shared and per-agent density settings, allowing tasks in which some agents represent humans with substantially denser calendars than others. In our experiments, starting configurations are balanced across models for fair evaluation.

**Costs.** Previously scheduled meetings have displacement cost 1 for each participant calendar on which the meeting must be moved. In the uniform-cost setting, every errand also has displacement cost 1. In the varied-cost setting, errands are as-

signed low, medium, or high disruption costs from a balanced set. Internally, these correspond to $\{1, 2, 3\}$, but agents are shown a logarithmic scale $\{1, 10, 100\}$.[1] The varied setting also creates cases where the best solution requires moving previously scheduled meetings instead of expensive errands, so agents should reach out to external meeting participants from prior meetings. Thus, CalBench tests whether agents balance preferential costs, recognize when external coordination is worthwhile, and leak private information while doing so.

**Private contexts and sensitivity.** Each errand and prior meeting is linked with a natural-language private context before it is shown to an LLM agent. Contexts are drawn from label banks with three sensitivity tiers: public, sensitive, and very sensitive. Agents see the private labels in their own calendar render, but they do not see the underlying tier labels. These contexts are not needed to solve the scheduling problem: agents may need to communicate availability, but not the private reason why a slot is occupied. This design lets the harness separate two forms of privacy loss. VPS measures inference about private calendar states (i.e., blocked slots and movement costs), while the context labels measure task-irrelevant semantic leakage in natural language.

**Communication protocols.** A game proceeds in rounds, one per incoming meeting. Each round begins with a cheap-talk phase in which agents coordinate under one or more communication channels. CalBench supports three message types: DM, a private direct message to one recipient; PARTICIPANT-GROUPCHAT, a shared channel visible only to the current meeting participants; and ALL-AGENT-GROUPCHAT, a shared channel visible to every agent in the task. Participants are active by default. Non-participants become active only after receiving a direct message or observing an all-agent groupchat message. This activation rule allows agents outside the current meeting to help repair conflicts, while preventing unrelated agents from acting unless the communication protocol has made them relevant.

**Round structure.** After cheap-talk, the game enters a voluntary phase. In this phase, activated non-participants may submit rescheduling actions, for example to move their copy of a previously scheduled meeting that blocks the current participants. The decision phase then asks each current participant to independently submit an atomic batch of actions. A batch may contain RESCHEDULE actions followed by a SCHEDULE action for the incoming meeting. Every reschedule action must include a short justification explaining why the agent is moving its human user's existing commitment. The batch is validated against the agent's calendar and applied only if valid. Finally, the resolution phase checks that all meeting participants scheduled the incoming meeting to the same slot and that any prior meetings remain consistent across all of their participants' calendars. Failed meetings are marked unresolved and the game proceeds to the next round.

**Task difficulty.** Difficulty is based on CP-SAT feasibility score. For each task, we count how many distinct meeting-to-slot assignments satisfy the calendar constraints, and normalize by the total number of possible assignments. Lower scores correspond to harder tasks because fewer joint schedules are feasible. Because no two meetings may occupy the same slot, for $M$ meetings, $T$ total slots, and $F$ feasible assignments returned by CP-SAT, we define the feasibility difficulty score as

$$d = \frac{F}{\frac{T!}{(T-M)!}}$$

Tasks are then bucketed into easy, medium, and hard ranges based on this score.

### 3.1 Non-LLM Reference Protocols

CalBench implements non-LLM reference protocols under the same private-information constraints. These protocols are not meant to imitate open-ended LLM dialogue. They define interpretable operating points: full numeric disclosure, binary feasibility exchange, and a tunable mechanism-design frontier. IMAP sends full per-slot cost vectors to the initiator, who picks the joint minimum: this is our low-cost, high-disclosure reference point. SD-MAP (Modi and Veloso, 2004) sends only typed proposal status per slot and follows a scheduling-difficulty bumping rule: this is our low-disclosure feasibility-first reference point. DSM (Farhadi and Jennings, 2021) exposes a tunable privacy–cost curve through the interaction between offer-set size and a per-offer privacy-cost weight $\theta$: lower privacy pressure permits broader offer sets and leaks more

[1] We use a logarithmic presentation because small linear differences such as $\{1, 2, 3\}$ were not behaviorally salient in pilot runs: models often treated these costs as qualitatively similar. The displayed log scale makes the intended preference ordering clearer to agents, while we map costs back to the linear scale for leaderboard ranking.

score information, while higher privacy pressure narrows offers, leaks less, and often pays higher coordination-adjusted cost. Full specifications and inclusion rationales appear in Appendix D.

### 3.2 Privacy Leakage: VPS

We measure structural privacy using Valuations of Possible States (VPS) (Maheswaran et al., 2006). For each meeting round $r$, target agent $i$, and observer agent $j$, VPS measures how much the observer's belief about the target's slot occupancy moves away from an uninformed prior $p_0 = 0.5$:

$$\mathrm{VPS}^{r}_{j\to i} = \sum_{k\in S} \left|\mathrm{Bel}^{r,\mathrm{post}}_{j\to i}[k] - p_0\right|. \quad (1)$$

Because each slot contributes equally, VPS is reported in slot-equivalent units and is comparable across uniform- and varied-cost calendars. We estimate VPS differently for typed protocols and LLM agents. For non-LLM reference protocols, typed JSON messages have known meanings, so DSM score vectors, IMAP cost vectors, and SD-MAP proposal/reply messages are converted directly into slot-level evidence. For LLM agents, whose messages are unstructured, we use reflection-calibrated VPS: after each round, a measurement-only prompt asks each agent how its beliefs changed about every other agent's calendar, and we count only belief movements that point toward the target's true occupancy. This makes the LLM estimate conservative while keeping it in the same slot-equivalent units as the typed-protocol estimator. We validate this proxy with an external LLM-as-judge audit over delivered chat messages (Appendix I). The judge finds higher overall leakage but meaningful agreement with self-reflection, so we interpret reflection-calibrated VPS as a lower bound on chat-observable privacy leakage. VPS captures structural leakage about availability and costs; semantic-context leakage, such as revealing private event descriptions, is audited separately.

## 4 Results

### 4.1 Experimental Setup

We evaluate a 90-task mixed-agent suite with 5 sequential meetings, 16 calendar slots, $N{=}5$ agents, and 3 rotating participants per meeting. The suite contains 45 uniform-cost and 45 varied-cost tasks, with per-agent calendar density sampled from $\{0.6, 0.8, 1.0\}$ and blocked-errand counts from $\{2, 4, 6\}$. Table 1 evaluates four shared models over three calibration slices ($N = 135$ seats per setting) and Llama 4 Maverick, Qwen 3.6 Plus, and DeepSeek V4 Pro over their entrant slices ($N = 45$ seats per setting). Fixed calibration runs use the same canonical task suite; live-matchmaking and public leaderboard ratings use the OpenSkill procedure in Appendix E.

### 4.2 Metrics

We report five primary evaluation metrics for each model, shown in Table 1: task success, coordination cost, privacy cost, communication efficiency, and fairness (burden allocation). For all metrics, let $\mathcal{G}_i$ be the set of games in which agent or model $i$ appears.

***Task success*** measures whether agents complete the requested scheduling task. We define it as:

$$S_i = \frac{1}{|\mathcal{G}_i|} \sum_{g\in\mathcal{G}_i} \frac{M^{\mathrm{sched}}_{g,i}}{M_{g,i}}$$

where $M_{g,i}$ is the number of meetings involving agent $i$ in game $g$ and $M^{\mathrm{sched}}_{g,i}$ is the number of those meetings successfully scheduled. A meeting is successful only if all required participants write the meeting to the same slot and all previously scheduled meetings remain calendar-consistent. This is the basic feasibility metric: an agent team that cannot consistently place meetings has failed the core task, regardless of cost or privacy behavior.

***Coordination cost*** (excess cost) measures how much unnecessary displacement cost the agents incur relative to an oracle scheduler. Let $c^{\mathrm{real}}_{g,i}$ be the realized displacement cost incurred by agent $i$ in game $g$, and let $c^{\mathrm{oracle}}_{g,i}$ be the CP-SAT oracle cost assigned to the same agent for the same set of successfully scheduled meetings. We define scheduled excess cost as

$$C_i = \frac{1}{|\mathcal{G}_i|} \sum_{g\in\mathcal{G}_i} \max\left(0, c^{\mathrm{real}}_{g,i} - c^{\mathrm{oracle}}_{g,i}\right)$$

When a game is only partially successful, we compute the oracle cost only over the subset of meetings that were actually scheduled.

For Figure 2, however, scheduled-only excess cost can make failed coordination look artificially cheap: an unscheduled meeting creates no displacement cost ($c^{\mathrm{real}}_{g,i} = 0$), even though it is a failure of the scheduling task. We therefore plot a coordination-adjusted excess cost that penalizes unscheduled meetings. For each task, we solve a second CP-SAT problem that keeps the same

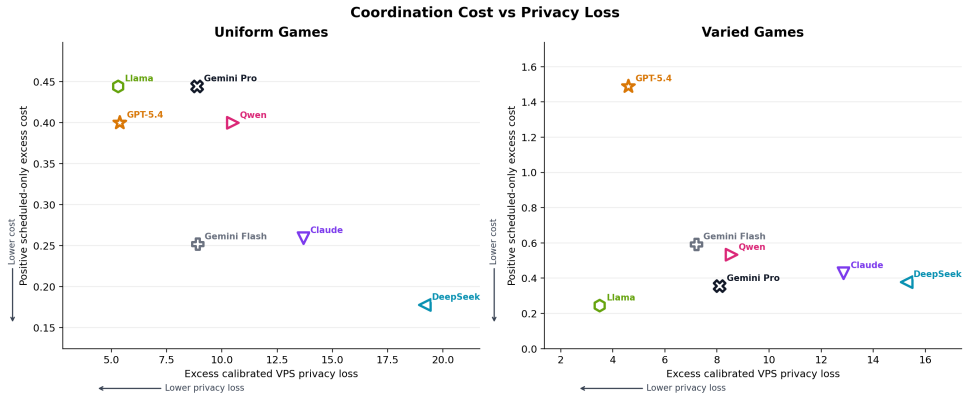


Figure 2: Privacy-cost plane for the canonical CalBench games, with uniform-cost games on the left and varied-cost games on the right. The closer to the origin, the better at reducing costs

| Setting | Model | $N$ | Meetings | Coord. | Excess | Messages/mtg | Fairness | Excess cal. VPS |
|---|---|---|---|---|---|---|---|---|
| Uniform | Claude Sonnet 4.6 | 135 | 2.54 | 84.69% | 0.26 | 3.46 | 0.585 | 13.68 |
| | Gemini 3.1 Pro | 135 | 2.58 | 85.93% | 0.44 | 3.07 | 0.616 | 8.87 |
| | Gemini 3 Flash | 135 | 2.59 | 86.17% | 0.25 | 3.36 | 0.474 | 8.90 |
| | GPT-5.4 Mini | 135 | 2.50 | 83.21% | 0.40 | 3.08 | 0.541 | 5.38 |
| | Llama 4 Maverick | 45 | 2.31 | 77.04% | 0.44 | 3.11 | 0.529 | **5.30** |
| | Qwen 3.6 Plus | 45 | **2.80** | **93.33%** | 0.40 | **2.96** | 0.489 | 10.49 |
| | DeepSeek V4 Pro | 45 | 2.69 | 89.63% | **0.18** | 3.21 | **0.396** | 19.18 |
| Varied | Claude Sonnet 4.6 | 135 | 2.53 | 84.20% | 0.43 | 3.50 | 0.913 | 12.85 |
| | Gemini 3.1 Pro | 135 | 2.45 | 81.73% | 0.36 | 3.40 | 0.748 | 8.10 |
| | Gemini 3 Flash | 135 | 2.36 | 78.52% | 0.59 | 3.56 | 0.721 | 7.21 |
| | GPT-5.4 Mini | 135 | 2.30 | 76.79% | 1.49 | 3.44 | 1.376 | 4.60 |
| | Llama 4 Maverick | 45 | 2.31 | 77.04% | **0.24** | **3.20** | **0.627** | **3.49** |
| | Qwen 3.6 Plus | 45 | 2.38 | 79.26% | 0.53 | 3.76 | 0.662 | 8.56 |
| | DeepSeek V4 Pro | 45 | **2.53** | **84.44%** | 0.38 | 3.24 | 0.964 | 15.27 |

Table 1: Results on the canonical mixed-agent CalBench task set, split by cost setting. $N$ is the number of evaluated agent-game seats. *Meetings* is the mean number of successfully scheduled meetings per agent-game (max 3). *Messages/mtg* counts direct, groupchat, and broadcast messages per successful participant-meeting. *Excess* is positive scheduled-only excess displacement cost, $\max(0, c^{\text{real}}_{g,i} - c^{\text{oracle}}_{g,i})$, using the 1/2/3 varied-cost scale. *Fairness* is the absolute deviation of each agent's signed excess burden from the within-game mean. *Excess cal. VPS* is calibrated slot-equivalent VPS. Lower is better for excess, messages, fairness, and VPS.

feasibility constraints as the oracle but maximizes total displacement cost over complete schedules. Let $w_{g,i}$ be agent $i$'s cost in this worst complete feasible schedule, and let $o_{g,i}$ be the same agent's cost in the minimum-cost complete oracle schedule. Let $m^{\text{miss}}_{g,i}$ be the count of unscheduled participant-meetings for agent $i$ in game $g$. The plotted cost uses an upper-bound regret proxy:

$$C^{\text{adj}}_i = \frac{1}{|\mathcal{G}_i|} \sum_{g \in \mathcal{G}_i} \frac{\Delta c_{g,i} + m^{\text{miss}}_{g,i} \Delta w_{g,i}}{M_{g,i}},$$

$$\Delta c_{g,i} = \max\left(0, c^{\text{real}}_{g,i} - c^{\text{oracle}}_{g,i}\right),$$

$$\Delta w_{g,i} = \max(0, w_{g,i} - o_{g,i}).$$

This keeps the unit as mean excess cost per meeting while assigning each missed meeting an equal share of the task-level worst-complete regret proxy rather than dropping it from the average. Because the penalty is the gap between the worst and best complete feasible schedules for the whole task, it should be read as a conservative coordination-failure penalty rather than the exact displacement cost of a particular missed meeting.

***Privacy cost*** measures excess calibrated VPS privacy loss as defined in Section 3.2. VPS counts slot-equivalent belief movement, giving all private slots equal weight so that model-to-model comparisons do not depend on the task's displacement-cost scale. Lower VPS cost indicates better privacy preservation.

$$V_i = \frac{1}{|\mathcal{G}_i|} \sum_{g \in \mathcal{G}_i} V_{g,i}$$

***Communication efficiency*** measures how many messages are exchanged to reach an outcome. For $N^{\text{msg}}_{g,i}$ communication messages sent by agent $i$, including direct messages, participant groupchats,

and all-agent broadcasts, we report messages per scheduled meeting:

$$E_i = \frac{1}{|\mathcal{G}_i|} \sum_{g \in \mathcal{G}_i} \frac{N_{g,i}^{\text{msg}}}{\max(M_{g,i}^{\text{sched}}, 1)}$$

Lower values indicate that agents reached outcomes with less communication overhead and fewer opportunities for privacy leakage.

***Fairness*** measures whether excess displacement burden is evenly distributed across agents within a game. We first compute each agent's excess burden:

$$e_{g,i} = c_{g,i}^{\text{real}} - c_{g,i}^{\text{oracle}}$$

We then compute the mean excess burden within the game:

$$\bar{e}_g = \frac{1}{N_g} \sum_{j=1}^{N_g} e_{g,j}$$

Each agent's relative excess burden is

$$r_{g,i} = e_{g,i} - \bar{e}_g$$

We define the fairness cost for agent $i$ as the average absolute relative excess burden across games:

$$F_i = \frac{1}{|\mathcal{G}_i|} \sum_{g \in \mathcal{G}_i} |r_{g,i}|$$

Lower $F_i$ indicates that the agent's excess burden is usually close to the within-game average, while higher $F_i$ indicates that the agent often absorbs substantially more or less excess burden than its teammates.

**Canonical mixed-agent tasks.** Table 1 reports model results on the canonical mixed-agent task set. In uniform-cost games, Qwen 3.6 Plus schedules the most meetings (2.80 of 3; 93.33%), while DeepSeek V4 Pro has the lowest excess cost (0.18) and fairness cost (0.396). Uniform-cost excess costs are small overall, ranging from 0.18 to 0.44.

The varied-cost setting separates models more clearly. DeepSeek V4 Pro has the highest task success (84.44%), but Llama 4 Maverick has the lowest excess cost (0.24), fewest messages per successful participant-meeting (3.20), lowest fairness cost (0.627), and lowest calibrated VPS (3.49). GPT-5.4 Mini performs worst on varied-cost excess cost (1.49) and fairness (1.376), despite having low VPS (4.60). Thus, completion alone does not capture schedule quality: models can finish similar numbers of meetings while differing substantially in cost, fairness, and privacy leakage. No semantic-context leakage was observed. For comparison, baseline protocol results are reported in Table 2.

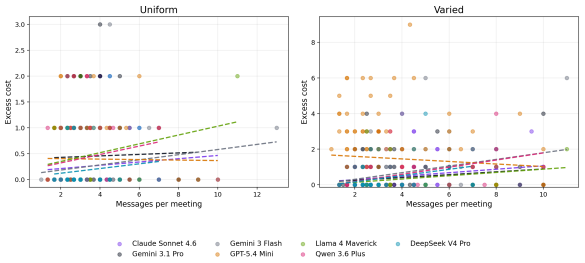


Figure 3: Communication efficiency for the canonical model cohort. Each point is one model seat in one game. The $x$-axis reports messages per successful participant-meeting, and the $y$-axis reports positive scheduled-only excess displacement cost using the 1/2/3 varied-cost scale. Dashed lines show per-model OLS fits.

### 4.3 Message Volume Is Not Coordination Quality

Figure 3 compares message volume with excess cost. If more communication reliably improved coordination, higher message counts should predict lower excess cost. The aggregate results do not show that pattern.

In uniform-cost games, Qwen 3.6 Plus sends the fewest messages (2.96) but has higher excess cost (0.40) than DeepSeek V4 Pro, which sends more messages (3.21) and has the lowest excess cost (0.18). Claude Sonnet 4.6 sends the most messages (3.46) without achieving the lowest excess cost. In varied-cost games, Llama 4 Maverick is both the least verbose and the lowest-cost model (3.20 messages; 0.24 excess), but this relationship does not hold generally: GPT-5.4 Mini sends 3.44 messages and incurs the highest excess cost (1.49), while DeepSeek V4 Pro sends fewer messages (3.24) and incurs much lower excess cost (0.38). Raw message count is therefore a weak proxy for coordination quality; what matters is whether messages convey useful availability and cost information.

**Fairness in varied-cost games.** Figure 4 shows that fairness costs are tightly clustered in uniform-cost games, ranging from 0.396 to 0.616. Varied-cost games create larger burden-allocation differences. Llama 4 Maverick has the lowest fairness cost (0.627), followed by Qwen 3.6 Plus (0.662) and Gemini 3 Flash (0.721). GPT-5.4 Mini has the highest fairness cost (1.376).

This suggests a model-specific privacy–fairness

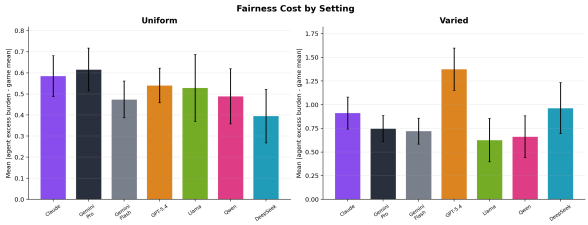


Figure 4: Fairness cost by model and cost setting. Lower values mean an agent's signed excess burden is closer to the within-game average. Error bars show 95% confidence intervals across task instances.

Table 2: Deterministic baseline results on the canonical mixed-agent CalBench task set. Each metric is computed over N = 45 tasks. VPS is computed using the baseline VPS formula and is not directly comparable to the reflection-calibrated VPS reported for language-model agents.

| Method | Coord. | Excess | Msgs. | Fair. | VPS |
|---|---|---|---|---|---|
| **Uniform Cost** | | | | | |
| IMAP | 100.0 | 0.26 | 2.00 | 0.530 | 12.40 |
| DSM-private | 45.8 | 0.98 | 4.54 | 0.809 | 0.00 |
| DSM-welfare | 100.0 | 0.27 | 2.69 | 0.530 | 25.05 |
| SD-MAP | 62.2 | 1.68 | 7.48 | 0.741 | 0.12 |
| **Varied Cost** | | | | | |
| IMAP | 100.0 | 0.32 | 2.00 | 0.665 | 12.40 |
| DSM-private | 48.9 | 2.08 | 4.43 | 1.865 | 0.00 |
| DSM-welfare | 99.1 | 0.46 | 2.78 | 0.816 | 23.55 |
| SD-MAP | 63.1 | 3.64 | 7.30 | 1.376 | 0.08 |

tension. GPT-5.4 Mini has low VPS in varied-cost games (4.60) but the worst fairness cost, whereas Llama 4 Maverick has both the lowest VPS (3.49) and the best fairness. The issue is therefore not privacy preservation itself, but the kind of information withheld. A message audit shows that GPT-5.4 Mini proposes its own slots at a similar rate to other models (0.875 vs. 0.866) and reports availability at least as often (0.853 vs. 0.769), but mentions its own costs or constraints far less often (6.3% vs. 29.2%; $p < 5 \times 10^{-6}$). This suggests that GPT-5.4 Mini's privacy comes partly from omitting cost-relevant context, leaving teammates with less evidence for fair burden allocation.

## 5 Discussion and Limitations

The benchmark separates three behaviors that are easy to conflate. First, booking the meeting is weaker than coordinating well: models can complete all assigned meetings while incurring extra displacement cost relative to the scheduled-only oracle. Second, more talk does not reliably reduce that cost; raw message count is not a quality measure. Third, privacy can conflict with fairness when agents hide cost-relevant constraints that teammates need for allocation decisions.

**Limitations and Future Work.** The reported results cover one topology ($N = 5$, 3 rotating participants); larger groups and longer meeting streams remain to be tested, though the harness supports them. Reflection-calibrated VPS measures reported belief movement rather than latent model state. CalBench assumes truthful reporting, leaving strategic misrepresentation to future work. The oracle and VPS machinery also apply to other streaming coordination problems, such as resource allocation or supply-chain handoffs.

## 6 Conclusion

CalBench tests whether calendar assistants can coordinate from private state without treating privacy, cost, and fairness as afterthoughts. Its mixed-agent evaluation shows that feasibility and cost optimization are separable: uniform games are near-saturated for most models, but varied-cost games expose large differences in excess burden. Message volume is a poor proxy for coordination quality, and low VPS can coincide with unfair burden allocation when agents omit cost-relevant context. CalBench is released with its scenario generator, reference protocols, VPS pipeline, and trace analysis toolkit.

## References

Sahar Abdelnabi, Amr Gomaa, Sarath Sivaprasad, Lea Schönherr, and Mario Fritz. 2024. Cooperation, competition, and maliciousness: Llm-stakeholders interactive negotiation. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA. Curran Associates Inc.

Eugene Bagdasarian, Ren Yi, Sahra Ghalebikesabi, Peter Kairouz, Marco Gruteser, Sewoong Oh, Borja Balle, and Daniel Ramage. 2024. Airgapagent: Protecting privacy-conscious conversational agents. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, pages 3868–3882, New York, NY, USA. Association for Computing Machinery.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. 2025. $\tau^2$-bench: Evaluating conversational agents in a dual-control environment. *Preprint*, arXiv:2506.07982.

Federico Bianchi, Patrick John Chia, Mert Yuksekgonul, Jacopo Tagliabue, Dan Jurafsky, and James Zou. 2024. How well can llms negotiate? negotiationarena platform and analysis. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Ismel Brito and Pedro Meseguer. 2008. Privacy in distributed meeting scheduling. In *Frontiers in Artificial Intelligence and Applications*, volume 184, pages 118–127.

Vincent P. Crawford and Joel Sobel. 1982. Strategic information transmission. *Econometrica*, 50(6):1431–1451.

Tim R. Davidson, Adam Fourney, Saleema Amershi, Robert West, Eric Horvitz, and Ece Kamar. 2025. The collaboration gap. *Preprint*, arXiv:2511.02687.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. 2024. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems*, volume 37, pages 82895–82920. Curran Associates, Inc.

Fabrício Enembreck and Jean-Paul André Barthès. 2012. Distributed constraint optimization with MULBS: A case study on collaborative meeting scheduling. *Journal of Network and Computer Applications*, 35(1):164–175.

Farzaneh Farhadi and Nicholas R. Jennings. 2021. A faithful mechanism for incremental multi-agent agreement problems with self-interested and privacy-preserving agents. *SN Comput. Sci.*, 2(4).

Jose L. Garcia, Karolina Hajkova, Maria Marchenko, and Carlos Miguel Pati no. 2025. Reproducibility study of cooperation, competition, and maliciousness: Llm-stakeholders interactive negotiation. *Preprint*, arXiv:2502.16242.

Rachel Greenstadt, Jonathan P. Pearce, and Milind Tambe. 2006. Analysis of privacy loss in distributed constraint optimization. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, AAAI'06, pages 647–653. AAAI Press.

Ralf Herbrich, Tom Minka, and Thore Graepel. 2006. Trueskill™: a bayesian skill rating system. *Advances in neural information processing systems*, 19.

Vivek Joshy. 2024. Openskill: A faster asymmetric multi-team, multiplayer rating system. *Journal of Open Source Software*, 9(93):5901.

Gurusha Juneja, Alon Albalak, Wenyue Hua, and William Yang Wang. 2025. Magpie: A dataset for multi-agent contextual privacy evaluation. *Preprint*, arXiv:2506.20737.

Arpandeep Khatua, Hao Zhu, Peter Tran, Arya Prabhudesai, Frederic Sadrieh, Johann K. Lieberwirth, Xinkai Yu, Yicheng Fu, Michael J. Ryan, Jiaxin Pei, and Diyi Yang. 2026. Cooperbench: Why coding agents cannot be your teammates yet. *Preprint*, arXiv:2601.13295.

Mike Lewis, Denis Yarats, Yann Dauphin, Devi Parikh, and Dhruv Batra. 2017. Deal or no deal? end-to-end learning of negotiation dialogues. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2443–2453, Copenhagen, Denmark. Association for Computational Linguistics.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, and 3 others. 2025. Agentbench: Evaluating llms as agents. *Preprint*, arXiv:2308.03688.

Rajiv T. Maheswaran, Jonathan P. Pearce, Emma Bowring, Pradeep Varakantham, and Milind Tambe. 2006. Privacy loss in distributed constraint reasoning: A quantitative framework for analysis and its applications. *Autonomous Agents and Multi-Agent Systems*, 13:27–60.

Rajiv T. Maheswaran, Milind Tambe, Emma Bowring, Jonathan P. Pearce, and Pradeep Varakantham. 2004. Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '04, pages 310–317, Washington, DC, USA. IEEE Computer Society.

Saaduddin Mahmud, Eugene Bagdasarian, and Shlomo Zilberstein. 2025. CoLLAB: A framework for designing scalable benchmarks for agentic LLMs. In *Workshop on Scaling Environments for Agents*.

Meta Fundamental AI Research Diplomacy Team (FAIR), Anton Bakhtin, Noam Brown, Emily Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew Goff, Jonathan Gray, Hengyuan Hu, Athul Paul Jacob, Mojtaba Komeili, Karthik Konath, Minae Kwon, Adam Lerer, Mike Lewis, Alexander H. Miller, Sasha Mitts, Adithya Renduchintala, and 8 others. 2022. Human-level play in the game of Diplomacy by combining language models with strategic reasoning. *Science*, 378(6624):1067–1074.

Niloofar Mireshghallah, Hyunwoo Kim, Xuhui Zhou, Yulia Tsvetkov, Maarten Sap, Reza Shokri, and Yejin Choi. 2024. Can llms keep a secret? testing privacy implications of language models via contextual integrity theory. *Preprint*, arXiv:2310.17884.

Pragnesh Jay Modi and Manuela Veloso. 2004. Multiagent meeting scheduling with rescheduling. In *Proceedings of the Fifth Workshop on Distributed Constraint Reasoning (DCR)*.

Abhijnan Nath, Hannah VanderHoeven, and Nikhil Krishnaswamy. 2026. Craft: Grounded multi-agent coordination under partial information. *Preprint*, arXiv:2603.25268.

Helen Nissenbaum. 2004. Privacy as contextual integrity. *Washington Law Review*, 79(1):119–158.

Tyler Payne, Will Epperson, Safoora Yousefi, Zachary Huang, Gagan Bansal, Wenyue Hua, Maya Murad, Asli Celikyilmaz, and Saleema Amershi. 2026. Socialreasoning-bench: Measuring whether ai agents act in users' best interests. Microsoft Research Blog.

Laurent Perron and Frédéric Didier. 2025. Cp-sat. Google OR-Tools, version 9.12.

Bram M. Renting, Holger Hoos, and Catholijn M Jonker. 2024. Multi-agent meeting scheduling: A negotiation perspective. In *The Sixteenth Workshop on Adaptive and Learning Agents*.

Yijia Shao, Tianshi Li, Weiyan Shi, Yanchen Liu, and Diyi Yang. 2025. Privacylens: Evaluating privacy norm awareness of language models in action. *Preprint*, arXiv:2409.00138.

Faouzi El Yagoubi, Godwin Badu-Marfo, and Ranwa Al Mallah. 2026. Agentleak: A full-stack benchmark for privacy leakage in multi-agent llm systems. *Preprint*, arXiv:2602.11510.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. $\tau$-bench: A benchmark for tool-agent-user interaction in real-world domains. *Preprint*, arXiv:2406.12045.

Huaixiu Steven Zheng, Swaroop Mishra, Hugh Zhang, Xinyun Chen, Minmin Chen, Azade Nova, Le Hou, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, and Denny Zhou. 2024. Natural plan: Benchmarking llms on natural language planning. *Preprint*, arXiv:2406.04520.

Xuhui Zhou, Hao Zhu, Leena Mathur, Ruohong Zhang, Haofei Yu, Zhengyang Qi, Louis-Philippe Morency, Yonatan Bisk, Daniel Fried, Graham Neubig, and Maarten Sap. 2024. SOTOPIA: Interactive evaluation for social intelligence in language agents. In *The Twelfth International Conference on Learning Representations*.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. 2025. Multiagentbench: Evaluating the collaboration and competition of llm agents. *Preprint*, arXiv:2503.01935.

# A Prompt Construction and Game Scaffolding

This appendix provides a self-contained description of how the experiment harness constructs agent prompts, sequences messages across game phases, and validates agent responses. Together with the system-prompt text in §A and the trace format in §A.6, this section contains sufficient detail to replicate the communication protocol from scratch.

## A.1 Scenario and Calendar Generation

Every game is seeded from a *scenario*, a deterministic data structure produced by `generate_scenario(seed, ...)`. Fixing the seed reproduces the exact calendar layouts, cost functions, participant lists, and witness solution used in the original run.

**Parameters.** Table 3 lists the key generation parameters.

**Calendar construction.** The generator uses a backward-from-witness design to guarantee feasibility:

1. A *witness slot* $w_c$ is drawn uniformly at random for each meeting $c$ from slots not already used by any of its participants. This hidden solution achieves a known cost.

2. For each agent $i$, errand items are placed in $\lfloor S \cdot \rho_i \rfloor$ slots drawn randomly, where $\rho_i$ is either the shared `density` value or the agent-specific entry in a density vector. Witness slots are deliberately seeded with errands (when `force_witness_errand` is true) to create displacement pressure.

3. A small set of *absorbing slots* — one per errand occupying a witness slot — is kept free so displaced errands always have a valid landing pad. This preserves the invariant that the witness solution is feasible under any sequence of moves needed to clear the witness slots.

4. Each errand is assigned a displacement cost drawn from Uniform$[1, \texttt{errand_cost_level}]$.

5. The optimal cost, greedy cost, and CP-SAT feasible-assignment fraction are computed and stored in the scenario dict. The feasible-assignment fraction counts distinct injective meeting-slot assignments that satisfy the CP-SAT constraints and divides by $S!/(S-C)!$.

**Calendar data model.** Each agent's calendar is a fixed-length list of *slots*. A slot is either `null` (free), an errand object `{errand_id, cost}`, or a meeting object `{meeting_id, cost}`. Blocked errands carry an additional `blocked: true` flag and cannot be moved. The rendered string shown to LLM agents is produced by `Calendar.render()`, which emits one line per slot in the format:

Table 3: Scenario generation parameters.

| Parameter | Type | Description |
|---|---|---|
| `seed` | int | RNG seed; fixes all randomness |
| `num_agents` | int | Number of agents $N$ |
| `num_slots` | int | Calendar length $S$ (default 16) |
| `density` | float or list | Shared or per-agent fraction of slots pre-filled with errands, $\in [0, 1]$ |
| `num_meetings` | int | Number of rounds $C$ |
| `pref_level` | int | Max errand cost draw from Uniform$[1,$ `pref_level`$]$ |
| `errand_cost_level` | int | Override for errand cost ceiling (defaults to `pref_level`) |
| `meeting_cost_level` | int | Max prior-meeting cost |
| `participant_lists` | list | Optional fixed participant sets per round |

**Calendar Render**

```
Slot 0: [FREE]
Slot 1: Errand #3 (cost=2)
Slot 2: Blocked Errand #7 (cost=5)
Slot 3: Meeting M2 (cost=1) participants=[0,
2]
```

**Private label hydration.** Before the rendered calendar or meeting description is delivered to an agent, the harness injects *private semantic labels* — naturalistic event descriptions such as "Intake session at the Willow Tree Eating Disorder Clinic" — drawn from a pre-generated label bank. Labels are assigned to slots once per scenario and remain stable across all rounds. The hydration layer enforces information asymmetry:

- An agent sees its own errand label on every turn.
- An agent sees a meeting's private label only if it is a participant in that meeting.
- Non-participant agents see the slot type (`errand`, `meeting`) and cost but not the semantic label.

This asymmetry is enforced at the scaffolding level and does not depend on the LLM's own compliance with privacy instructions.

**Label bank construction.** Labels are generated offline using an LLM (Gemini 3 Flash) prompted with one of three tier-specific instructions: SENSITIVE (medical, legal, financial, identity), NEUTRAL (ambiguously private everyday logistics), and PUBLIC (routine low-stakes errands). The resulting label bank is stored as a JSON file and loaded deterministically at scenario generation time.

### A.2 System Prompt Construction

Each agent receives a single **system prompt** at registration time, delivered as the `system` role message before any user turns. The prompt is assembled once per agent from a fixed template parameterised by the following game-configuration values:

- `agent_id` — the agent's own integer identifier.
- `all_agent_ids` — the complete list of agent identifiers in the game (e.g. `0, 1, 2, 3, 4` for a 5-agent game).
- `num_agents` — total number of agents.
- `num_slots` — number of calendar slots.
- `decision_retries` — maximum retries allowed if a DECISION batch fails validation.

The template is divided into nine fixed sections (shown verbatim in §A.2):

1. RULES — hard constraints: no double-booking, same-slot consistency for all participants, same-slot constraint for rescheduled meetings.
2. IMPORTANT CONSTRAINTS — information constraints: own-only visibility, prohibition on sharing errand/meeting details with non-participants, prohibition on sharing cost functions.
3. NEGOTIATION STRATEGY — six negotiation guidelines instructing the agent to prefer free slots, push back on costly proposals, and use only qualitative language about difficulty.
4. CALENDAR SLOT TYPES — definitions of the four slot types (`free`, `blocked`, `errand`, `meeting`).
5. TOOLS — the JSON tool-call schema for `dm`, participant groupchat, all-agent groupchat, `schedule`, and `reschedule`, with phase-specific validity rules.
6. PHASES — descriptions of the four phases each agent participates in (CHEAP_TALK, VOLUNTARY, DECISION, RESOLUTION).

7. RESPONSE FORMAT — the required `{"thinking": ..., "actions": [...]}` envelope with a worked example.

8. IDENTITY — the agent's own ID and the full agent list.

9. ENVIRONMENT PARAMETERS — slot count, retry budget, multi-round structure.

**Adversarial variant injection.** Agents configured with the NOSY-HIGHPRESSURE or REDTEAM variants receive one additional section appended verbatim after the base prompt (see §A.2 and §A.2). No other section changes; the injection is self-contained.

**DSPy variant injection.** Agents configured with a DSPy prompt variant (`type: dspy` in the experiment YAML) receive a separately versioned Markdown block loaded from a `prompt_variants/` directory and appended after the base prompt. The variant filename and directory are stored in the experiment config and trace metadata, making the augmentation fully reproducible. The base prompt is identical to the standard agent prompt; the DSPy block is the only difference.

### A.3 Per-Round User Message Sequence

After registration, the harness drives each round by appending **user-role messages** to each agent's conversation. A single game keeps one per-agent conversation thread alive for the full multi-round game; messages accumulate in the thread rather than being reset between rounds. Table 4 lists the message types in delivery order. The same round loop is used for all communication conditions. The private-DM condition exposes only addressed messages, the meeting-participant groupchat condition exposes a shared channel to the current meeting participants, and the all-agent groupchat condition exposes a shared channel to every agent. Meeting participants are active by default; non-participants are activated only by a direct DM or by an all-agent groupchat message.

**Round-start message.** Produced by `build_round_start_message(meeting, calendar_render, round_num, incurred_penalty, turn_index, max_turns_per_round)`. Delivered to every participant at the start of CHEAP_TALK (turn 0). Contains, in order:

1. A `=== ROUND N START ===` header.

2. The meeting to schedule: ID, participant list, duration in slots, and the private meeting label (visible only to participants).

3. The agent's own calendar as a rendered slot list.

4. The agent's cumulative penalty incurred in previous rounds.

5. The active phase (`CHEAP_TALK`) and, when `max_turns_per_round` is set, a turn-budget line of the form "CHEAP_TALK turn budget: turn 1 of N. $k$ turn(s) remain after this one."

6. A reminder that DECISION follows CHEAP_TALK and that agents should negotiate for a low-displacement slot.

**Turn message.** Produced by `build_turn_message(messages, turn_index, max_turns_per_round)`. Delivered for every subsequent CHEAP_TALK turn to agents in the current speaker order. When the agent has new messages, the body lists each incoming private or groupchat message with channel metadata, for example:

**Incoming Message**

```
[1] From Agent 2 (meeting 3): <content>
```

When the inbox is empty, the body reads "No new messages in your inbox." On the final allowed turn (`turn_index + 1 == max_turns_per_round`), an additional sentence instructs the agent not to ask open-ended questions and to return `[]` if coordination is complete.

**Voluntary-reschedule message.** Produced by `build_voluntary_reschedule_message(meeting, calendar_render)`. Delivered after CHEAP_TALK closes to non-participant agents who received at least one DM during the round. The message explains that the agent may execute `reschedule` calls to honour commitments made during negotiation but may not use `schedule`.

**Decision message.** Produced by `build_decision_message(meeting, calendar_render)`. Delivered to each participant to collect the final scheduling batch. The message repeats the meeting metadata and a *frozen* calendar snapshot taken at the moment DECISION

Table 4: User-message sequence per round. Each row is one message appended to an agent's conversation thread.

| Phase / condition | Message builder | Delivered to |
|---|---|---|
| CHEAP_TALK, turn 0 | `build_round_start_message` | all participants |
| CHEAP_TALK, turn $t \geq 1$, agent received messages | `build_turn_message` | agents with non-empty inbox |
| CHEAP_TALK, turn $t \geq 1$, no new messages | `build_turn_message` (empty inbox path) | agents with empty inbox but still in speaker order |
| VOLUNTARY (non-participant contacted) | `build_voluntary_reschedule_message` | non-participants activated during CHEAP_TALK |
| DECISION | `build_decision_message` | all participants |
| DECISION retry (batch rejected) | `build_retry_message` | the participant whose batch failed |

begins (post-VOLUNTARY). It instructs the agent to submit at least one `schedule` call and any `reschedule` calls needed to free the target slot, reminds the agent that the batch is resolved atomically, and instructs the agent to use the slot agreed during CHEAP_TALK.

**Retry message.** Produced by `build_retry_message(attempt, max_attempts, conflict)`. Delivered when a DECISION batch fails validation. The body states the attempt counter, the exact conflict description (e.g. "Expected exactly 1 schedule action, got 0"), and instructs the agent to resubmit the complete batch from scratch without referencing the previous attempt.

### A.4 Game-Round Phase Orchestration

The harness runs each round synchronously within a single async loop. Algorithm 1 gives the complete pseudocode.

**Message routing.** A private DM is a tool call `{"type": "dm", "to": <int>, "content": "<string>"}` emitted during CHEAP_TALK. The meeting-participant groupchat and all-agent groupchat tools use the same content payload but differ in visibility: participant groupchat messages are appended to the inboxes of the current meeting participants, while all-agent groupchat messages are appended to every agent's inbox. Recipients drain their inboxes at the start of their next turn; the drained snapshot is what populates the TURN message. Non-participant agents who receive a private DM or all-agent groupchat message are queued to speak in the same CHEAP_TALK turn and may themselves send messages, which triggers further routing. Participant-only groupchat does not activate non-participants. Each turn index $t$ corresponds to one sweep through all active speakers.

**Turn budget.** When `max_turns_per_round` is set, the loop terminates after that many sweeps even if DMs are still being sent. The final turn message includes an explicit instruction to close coordination, preventing unbounded negotiation.

**Participant vs. non-participant paths.** At each CHEAP_TALK turn, participants always receive a message (round-start on turn 0, turn message thereafter) regardless of whether they have inbox messages. Non-participants only receive a message if they were queued because a direct DM or all-agent groupchat message reached them; otherwise they are silent.

### A.5 Batch Validation Rules

Every agent response must be a JSON object with exactly two keys:

- `"thinking"` — a free-text chain-of-thought string. Logged in the trace but never forwarded to other agents.
- `"actions"` — a list of tool-call objects, or `[]` to pass without action.

The harness extracts `"actions"` and applies the following validation rules before committing any changes to the calendar:

1. **Slot bounds.** All `slot`, `from_slot`, and `to_slot` values must be integers in $[0, S)$.
2. **Item identity.** Each `reschedule` action's `item_id` must match the `errand_id` or `meeting_id` of the item actually present at `from_slot`.
3. **No blocked moves.** Items with `blocked: true` may not be moved.
4. **No destination conflicts.** No two actions in the batch may target the same `to_slot`.

5. **Freeness after batch.** Each `to_slot` must be free on the calendar *or* be freed by another `reschedule` in the same batch.

6. **Exactly one schedule.** In the DECISION phase, the batch must contain exactly one `schedule` action. (The VOLUNTARY phase uses *require_schedule*= FALSE.)

7. **Schedule slot free.** The `schedule` slot must be free after all `reschedule` operations are applied.

Validation is transactional: nothing is applied until the entire batch passes. On failure the harness emits a BATCH_REJECTED event with the exact conflict string and delivers a RETRY message to the agent. Up to `decision_retries` retries are attempted; if all fail, the last attempted batch is discarded and the round proceeds without that agent's scheduling action, triggering a consistency violation.

### A.6 Trace Format

Each game run produces a single JSON file containing:

- `game_id` — UUID generated at run time.
- `config` — the full experiment configuration dict, including all agent specs, scenario parameters, and experiment metadata.
- `events` — a chronologically ordered list of typed event objects. Key event types are listed in Table 5.
- `final_state` — aggregate metrics: total displacement cost, rounds succeeded/failed, DM counts, consistency violations.
- `metrics` — derived metrics computed at run end (e.g. normalised cost vs. optimal, cost vs. greedy).
- `started_at`, `ended_at` — ISO-8601 timestamps.

The `system_prompt` field of each `agent_registered` event contains the exact text delivered to the model, including any adversarial or DSPy variant appended section. Replaying a trace therefore requires only the model API; all prompts are self-contained in the file.

## B Model configurations

Table 6 lists the seven models evaluated in the benchmark-lite experiments, along with their API provider and model identifier as used in the experiment configuration files.

All cooperative benchmark-lite runs reported in the main table use the same prompt scaffold and game configuration (5 agents, 3 participants per meeting, 5 meetings per task, 16 time slots, 15 turns per round, no DM cap), with homogeneous same-model teams and private agent-to-agent DMs. The current harness also supports heterogeneous model teams by assigning a model provider and identifier per agent, as well as meeting-participant groupchat and all-agent groupchat communication conditions. The adversarial condition uses agent 0 with a frozen DSPy-compiled probing prompt (variant `c006`) against the same-model agents on the 5-agent, 3-participant subset (6 turns per round, DM cap 100).

## C API Costs

Claude traces did not record provider token counts, so Claude costs are estimated from reconstructed prompt and response lengths. Two Gemini 3 Flash calls with missing token usage are estimated the same way; all other rows use recorded token counts.

## D Non-LLM Reference Protocols

This appendix specifies the four non-LLM reference protocols introduced in §3.1: their per-round message flows, slot-selection rules, parameters, and privacy-relevant invariants. All four share the CalBench harness (per-round cheap-talk, voluntary, decision, and resolution phases; same DM and broadcast channels; same SCHEDULE/RESCHEDULE action validation against local state). Differences below are entirely in the policy that maps observations to messages and actions.

### D.1 Incremental MAP (IMAP)

**Protocol.** For each meeting, the lowest-id participant acts as initiator. The initiator sends a single COST_REQUEST DM to every other participant, listing the slot indices of the calendar. Each responder computes its local insertion cost for every slot — 0 if the slot is free, the errand displacement cost if a movable errand can be displaced to a known free slot, and infeasible otherwise — and returns the full vector in a single COSTS DM. The initiator sums

Table 5: Key event types in the game trace.

| Event type | Contents |
|---|---|
| `game_start` | Scenario seed, num agents/slots, optimal/greedy cost, nosy agent IDs |
| `agent_registered` | Agent ID, system prompt text, initial calendar render, is-nosy flag |
| `round_start` | Round number, meeting dict, speaker order |
| `turn_start` | Round, turn, phase, agent ID, inbox snapshot, prompt sent |
| `turn_end` | Tool calls, thinking text, token usage, latency |
| `dm_sent` | From/to agent IDs, meeting ID, content, char count |
| `decide_start` | Phase (VOLUNTARY or DECISION), prompt sent, calendar render |
| `decide_end` | Tool calls, thinking, usage, latency |
| `batch_rejected` | Attempt number, conflict description, rejected actions |
| `batch_applied` | Applied actions, calendar render after |

| Model name | Provider | API / access | Model identifier |
|---|---|---|---|
| Claude Sonnet 4.6 | Anthropic | Google Vertex AI (Anthropic) | `claude-sonnet-4-6` |
| DeepSeek V4 Pro | DeepSeek | OpenRouter | `deepseek/deepseek-v4-pro` |
| Gemini 3 Flash | Google | Google Vertex AI | `gemini-3-flash-preview` |
| Gemini 3.1 Pro | Google | Google Vertex AI | `gemini-3.1-pro-preview` |
| GPT-5.4 Mini | OpenAI | OpenAI API | `gpt-5.4-mini` |
| Llama 4 Maverick | Meta | Google Vertex AI (OpenAI-compat.) | `llama-4-maverick-17b-128e-instruct-maas` |
| Qwen3.6 Plus | Alibaba | OpenRouter | `qwen/qwen3.6-plus` |

Table 6: Model identifiers and API providers for all seven models in the benchmark-lite evaluation. All models are accessed via the LiteLLM routing layer; the API format column reflects the provider-specific adapter used by the CalBench harness.

Table 7: Estimated API costs for the canonical mixed-agent trace set. The four shared models appear in all 270 traces; Llama, Qwen, and DeepSeek appear in one 90-trace slice each. Costs assume non-cached, non-batch API pricing and bill output as completion plus reasoning tokens where reasoning-token usage is reported.

| Model | Traces | Uniform | Varied | Total |
|---|---|---|---|---|
| Gemini 3.1 Pro | 270 | $42.61 | $41.81 | $84.42 |
| Claude Sonnet 4.6 | 270 | ∼ $97.45 | ∼ $94.44 | ∼ $191.89 |
| Gemini 3 Flash | 270 | $13.98 | $13.81 | $27.80 |
| GPT-5.4 Mini | 270 | $17.67 | $17.40 | $35.07 |
| Llama 4 Maverick | 90 | $2.73 | $2.64 | $5.37 |
| Qwen3.6 Plus | 90 | $7.07 | $6.95 | $14.02 |
| DeepSeek V4 Pro | 90 | $4.09 | $3.98 | $8.07 |

per-slot costs across itself and all responders, picks the slot with minimum total feasible cost (breaking ties by slot index), and broadcasts a DECISION DM. All participants then schedule the agreed slot in the decision phase.

**Cost-optimality and limits.** Because every participant reveals its full per-slot cost vector to the initiator, IMAP achieves the minimum-cost feasible insertion for the *current* meeting whenever one exists in the local-errand subproblem. It treats previously scheduled multi-agent meetings as hard commitments and does not bump them: renegotiating a confirmed multi-agent meeting requires restarting a coordinated rescheduling episode (Modi and Veloso, 2004), which IMAP does not initiate. IMAP is therefore exact only with respect to the local errand-displacement subproblem of the current meeting.

**Numeric disclosure footprint.** Two DMs per responder per meeting (one COST_REQUEST, one COSTS reply). Each COSTS message carries one integer per slot index. There is no conditional or selective disclosure: the entire local cost structure is sent on every meeting.

### D.2 Scheduling-Difficulty MAP (SD-MAP)

**Protocol.** SD-MAP is a low-disclosure reference protocol adapted from Modi and Veloso (2004)'s scheduling-difficulty rescheduling heuristic. Meetings are processed in the incoming order defined by the task fixture; the protocol does not globally reorder meetings by difficulty. For each meeting, the lowest-id participant acts as initiator. The initiator iterates over candidate slots in local calendar order and sends a PROPOSE DM containing only the meeting id and proposed slot to every responder. Each responder evaluates the slot with a binary feasibility check: PENDING if the slot is free or holds a movable errand; PENDING if the slot holds a confirmed prior meeting that is bumpable under the scheduling-difficulty rule (in which case a tentative bump is recorded locally); IMPOSSI-

| Protocol | Rationale for inclusion |
|---|---|
| IMAP | High-disclosure, low-regret reference point. Participants reveal full local cost vectors, so the initiator can choose the minimum-cost feasible slot for the current meeting. |
| SD-MAP | Low-disclosure feasibility reference point adapted from multi-agent meeting scheduling. Agents exchange binary proposal status rather than costs, exposing what is lost when the protocol avoids utility disclosure. |
| DSM-WELFARE | Mechanism-design reference point with broad offers and welfare-leaning parameters, representing the high-coordination side of the DSM privacy–cost frontier. |
| DSM-PRIVATE | Matched DSM variant with narrow offers and a high privacy penalty, isolating the effect of pushing the same mechanism toward low disclosure. |

Table 8: Why each non-LLM reference protocol is included. The protocols are not intended to model natural-language dialogue; they provide interpretable operating points for comparing LLM agents under the same private-information scheduling task.

BLE otherwise. There are no cost vectors or score vectors — feasibility is strictly binary. Replies arrive in a REPLY DM. If all replies are PENDING, the initiator broadcasts a CONFIRM DM and the slot is agreed. Otherwise the initiator tries the next candidate slot. If no slot succeeds, the initiator sends a FAIL DM and the meeting is unresolved.

**Bump repair via cheap-talk.** After a new meeting is confirmed, any tentative bump triggers a cheap-talk repair episode coordinated through the cheap-talk phase of the next round. The bumping participant sends a RESCHEDULE_REQUEST DM to the bumped meeting's initiator. That initiator proposes a repair slot with a PROPOSE_RESCHEDULE DM to all affected participants, who reply with RESCHEDULE_REPLY (PENDING or IMPOSSIBLE). Once all affected participants accept, the initiator sends a CONFIRM_RESCHEDULE DM. Current participants then apply the agreed moves in their DECISION phase actions; non-participants apply them in the VOLUNTARY phase. If no repair slot can be found, the initiator sends FAIL_RESCHEDULE and the bumped meeting falls back to its pre-bump slot.

**The bumping rule.** SD-MAP has no slot score vector. Feasibility is binary: PENDING or IMPOSSIBLE. Scheduling difficulty $\Delta$ is a meeting-priority quantity used only for bump decisions, not slot ordering:

$$\Delta(M) \;=\; \sum_{a \in \text{participants}(M) \setminus \{\text{self}\}} \sigma(a),$$

where $\sigma : \mathcal{A} \to \mathbb{R}_{\geq 0}$ is a per-agent difficulty weight. A confirmed prior meeting $M_2$ is tentatively bumped in favor of the new meeting $M_1$ only when

$$\textsc{Bump}(M_2 \mid M_1) \;\equiv\; \Delta(M_2) < \Delta(M_1),$$

i.e., rescheduling the existing meeting is strictly easier than rescheduling the new one.

**SD model.** CalBench supplies $\sigma$ via a configurable `sd_model` entry on the game config; absent any specification all agents default to unit difficulty, which reduces the rule to "bump iff the existing meeting has strictly fewer other participants than the new meeting." We use this default for all reported SD-MAP results.

**Numeric disclosure footprint.** One PROPOSE DM per responder per attempted slot, and one binary REPLY in return. Successful confirmation adds one CONFIRM DM per responder. Bump repair adds one RESCHEDULE_REQUEST, one PROPOSE_RESCHEDULE per affected participant, one RESCHEDULE_REPLY per participant, and one CONFIRM_RESCHEDULE or FAIL_RESCHEDULE. No costs, cost vectors, or slot indices beyond the one being proposed are ever transmitted.

### D.3 Distributed Score-based Multi-round (DSM)

**Origin.** The two DSM reference protocols (DSM-WELFARE and DSM-PRIVATE) are implemented using the PAPERDSM client, which follows the Distributed Score-based Multi-round mechanism of Farhadi and Jennings (2021) including satisfaction-level scoring, score-cost bookkeeping, and the flexibility-adjusted reward to the selected responder. We extend the protocol with multiple proposal sub-rounds per meeting: when the initial offer set yields no jointly feasible slot, the initiator proposes the next untried batch, repeating until a feasible slot is found or the slot pool is exhausted.

**Protocol.** Every meeting has a designated initiator (the lowest-id participant). In each sub-round, the initiator selects an offer set of $L$ candidate slots

from those it can locally schedule into, ordered by its own local feasibility score, and sends them as a single PROPOSALS DM to every responder. Each responder maps each proposed slot to a discrete satisfaction level $s \in \{0, 1, \ldots, D-1\}$ with $D = 12$, where 0 encodes infeasibility, $D-1$ encodes a free slot with no displacement, and intermediate levels decrease monotonically with displacement cost. Responders return the full score vector in a SCORES DM. The initiator combines local and reported scores into a per-slot rank and either announces the best fully-feasible slot via a DECISION DM, or proposes another untried batch. The protocol also supports cross-meeting displacement: when a candidate slot is blocked by a prior multi-agent meeting, the initiator can attach a proposed displacement plan to its offer, which the bumped meeting's participants score as additional responders (Modi and Veloso, 2004). All participants then commit the agreed slot and any agreed reschedules in the decision phase.

**Satisfaction levels and scoring cost.** Each non-zero score $s$ carries a scoring cost $C(s) = D - s - 1$ that the responder pays to the initiator's point budget at decision time. The initiator pays back a flexibility-adjusted reward to the responder whose score was selected; responders whose offered slot is not chosen receive no reward but still pay $C(s)$ on each non-zero score they reported, mirroring the paper's faithfulness construction.

**Offer-size utility.** The initiator chooses its offer size $L$ at each sub-round to maximize expected utility:

$$U(L) \;=\; p_{\text{succ}}(L)\,\bar{v}(L) \;+\; w_{\text{soc}}\,p_{\text{succ}}(L) \\ -\; \theta\,c_{\text{priv}}\,L \;-\; \beta\big(1 - p_{\text{succ}}(L)\big)$$

over $L \in [L_{\min}, L_{\max}]$, where:

- $p_{\text{succ}}(L) = 1 - (1-\hat{p})^L$ is the estimated probability that at least one of the top-$L$ candidate slots is jointly feasible. $\hat{p}$ is a density-based estimate of single-responder feasibility, obtained from the initiator's own calendar density and the responder count.
- $\bar{v}(L)$ is the mean local feasibility score of the top-$L$ candidates under the initiator's local cost ordering, normalized to $[0, 1]$.
- $\theta \geq 0$ scales a per-offer privacy cost (each disclosed slot is treated as an additional unit of leakage).
- $\beta \geq 0$ scales the cost of an extra negotiation round when the current offer fails to yield a fully feasible slot.
- $w_{\text{soc}} \geq 0$ is a social-welfare bonus on success.

The $\beta/\theta$ ratio is the dominant operating-point control: large $\beta$ relative to $\theta$ favors broad offer sets (DSM-WELFARE), while large $\theta$ relative to $\beta$ favors narrow offer sets (DSM-PRIVATE).

**Aggregate scoring statistics.** For a score vector $(s_1, \ldots, s_L)$ returned by a responder, we compute three summary statistics used by the reward and observability machinery: *availability* $A = |\{i : s_i > 0\}|$, the number of feasible offers; *flexibility*, a base-$(A+1)$ polynomial encoding of the score multiset that monotonically rewards both more feasible offers and higher satisfaction within them; and *scoring cost* $\sum_i C(s_i)$, the total points the responder pays. The initiator's selection reward to the responder whose score was chosen is then

$$R \;=\; (D - 1 - s^\star) \;+\; \max\big(0, \min(A, L) - 1\big),$$

where $s^\star$ is the selected score; the second term is the flexibility bonus that ensures responders with multiple feasible offers are not penalized for cooperativeness. We clip $R = 0$ when $s^\star = 0$ (infeasible) or $s^\star = D-1$ (free slot, no concession to compensate). The initiator's point budget is updated as $\Delta = \sum C(s_i) - R$ across all responders for that decision, preserving the faithful-mechanism property that points flow from those who concede less to those who concede more.

**Numeric disclosure footprint.** One PROPOSALS DM per responder per sub-round (carrying $L$ slot indices) and one SCORES reply (carrying $L$ values in $\{0, \ldots, D-1\}$). The number of sub-rounds per meeting is bounded by the search policy: DSM-WELFARE runs exhaustively until a fully-feasible slot is found or the slot pool is exhausted; DSM-PRIVATE stops early under its tighter $\theta$. Cross-meeting displacement attempts add one SCORES DM per displaced-meeting participant per offer set in which the displacement appears.

**Preset values.** Table 9 summarizes the parameter settings for the two DSM presets. DSM-WELFARE uses CalBench's defaults; DSM-PRIVATE overrides them to anchor the high-privacy end of the frontier.

| Parameter | DSM-WELFARE | DSM-PRIVATE |
|---|---|---|
| Client type | `paper_dsm` | `private_dsm` |
| $L_{\min}$ | 1 | 1 |
| $L_{\max}$ (`dsm_num_proposals`) | 12 | 2 |
| $\beta$ (failure penalty) | 1.0 | 0.25 |
| $\theta$ (per-offer privacy cost) | 0.0 | 10.0 |
| $w_{\text{soc}}$ (social-welfare weight) | 1.0 | 0.25 |
| Cascade depth | 2 | 1 |
| Displacement targets considered | 4 | 2 |
| Exhaustive search of feasible offers | yes | no |

Table 9: DSM preset parameters for the benchmark-lite reference-protocol runs. DSM-WELFARE uses the `paper_dsm` client with social-welfare-leaning settings from Farhadi and Jennings (2021). DSM-PRIVATE uses the `private_dsm` client, which internally overrides $L_{\max}$, $\beta$, $\theta$, $w_{\text{soc}}$, cascade depth, and displacement targets to minimize per-meeting disclosure.

### D.4 Common privacy invariants

All four non-LLM reference protocols share two structural privacy properties relative to LLM agents. First, the message vocabulary is restricted to typed JSON: cost vectors (IMAP), proposal/reply/confirm/reschedule status (SD-MAP), or quantized scores over candidate slots (DSM). None transmit any natural-language content. Second, by construction none can leak the natural-language semantic context attached to calendar entries (low/medium/high sensitivity, §3), since semantic context is never present in the message vocabulary. The protocols therefore set a quantitative floor of zero on context leakage, against which we measure the leakage rate of LLM agents communicating in unrestricted English over the same task.

## E OpenSkill Leaderboard Ranking

The hosted leaderboard uses OpenSkill (Joshy, 2024), a multiplayer rating system related to TrueSkill (Herbrich et al., 2006). Each calendar trace is treated as a free-for-all rating event over the agent seats in that game. Seats are mapped back to stable player identities by model name, so a model can receive rating updates from many heterogeneous teams. For each model $i$, we maintain a separate rating distribution for each leaderboard metric, $(\mu_{i,k}, \sigma_{i,k})$, where $\mu$ is estimated skill and $\sigma$ is uncertainty.

We rate models on three dimensions:

$$k \in \{\text{coordination}, \text{cost}, \text{privacy}\}.$$

Coordination is higher-is-better; excess cost and excess VPS are lower-is-better. Each dimension is updated separately, so the leaderboard does not collapse a game to a single win/loss outcome: a model can gain coordination rating while losing privacy rating in the same trace.

**Score-margin updates.** The leaderboard uses OpenSkill's score-margin variant rather than a rank-only update. Instead of discarding metric magnitudes after ranking, we pass the raw per-seat scores to OpenSkill. Coordination uses the scheduled-meeting ratio directly, while excess cost and excess VPS are negated so that larger OpenSkill scores are always better. OpenSkill then applies a logarithmic margin factor to larger score gaps, so a decisive cost or privacy difference can produce a larger rating update than a narrow one.

The margin parameters are $0.2$ for coordination, $10.0$ for excess cost, and $5.0$ for excess VPS. These margins are calibrated to the observed per-game score ranges rather than treated as tie tolerances. Varied-cost errand values remain on their raw $\{1, 10, 100\}$ scale; we do not remap them to $\{1, 2, 3\}$ for the rating update. Model privacy scores use reflection-calibrated excess VPS; protocol-semantic VPS is used for non-LLM reference-protocol reports.

**Displayed values.** The raw parenthetical values shown in the leaderboard are not OpenSkill rating scores. They report interpretable means such as coordination ratio, raw game-level excess cost, and excess VPS. The separate $\mu \pm \sigma$ columns expose the underlying metric-specific OpenSkill ratings, while MMR provides a single number for matchmaking and headline ordering.

Non-LLM reference protocols are shown separately from model ratings and ranked by raw metrics only: coordination descending, then excess cost ascending, then excess VPS ascending. They are not mixed into the model OpenSkill pool because they serve as protocol reference points rather than hosted matchmaking competitors.

## F VPS metric: implementation details

This appendix specifies the VPS estimators introduced in §3.2: protocol-semantic VPS for typed DSM/IMAP/SD-MAP reference-protocol messages and reflection-calibrated VPS for LLM agents. Both are measurement procedures rather than online mechanisms; they do not alter the agents' scheduling behavior.

### F.1 Belief representation and update rule

For each $(round, target, observer)$ triple $(r, i, j)$, the metric maintains a belief vector $\mathrm{Bel}^{r}_{j\to i} \in [0,1]^{|S|}$ initialized to the uniform prior $\mathrm{Bel}^{r,\mathrm{prior}}_{j\to i}[k] = p_0 = 0.5$. Each visible DM $m$ in trace order is replayed and may produce one or more belief-update events, each a tuple $(k, e, \alpha, \texttt{source})$ specifying the slot $k$ the message implicates, the evidence value $e \in [0,1]$ about feasibility (1.0 = "slot is feasible", 0.0 = "slot is infeasible"), the strength $\alpha \in [0,1]$ of the inference, and the source tag for downstream decomposition. The update is the strength-weighted nudge

$$\mathrm{Bel}'[k] \;=\; (1-\alpha)\,\mathrm{Bel}[k] \;+\; \alpha\, e,$$

applied independently to each slot $k$. With $\alpha = 1$ the posterior at $k$ is replaced by $e$; with $\alpha = 0$ the posterior is unchanged; intermediate values produce a partial move. Updates are accumulated over the round, and end-of-round leakage is computed from Eq. 1. We track each update's source so that VPS can be decomposed by message channel in the results.

### F.2 Typed-message update rules for reference protocols

All non-LLM reference protocols emit typed JSON DMs with known semantics. Table 10 gives the full protocol-semantic rule set. The visibility model is point-to-point: each DM contributes only to the belief pair defined by its sender and recipient. Most typed updates are exact; SD-MAP proposals receive a softer strength because proposing a slot is strong evidence of local availability but not logically identical to a hard feasibility report.

DSM SCORES replies require care because the reply payload carries plan ids rather than slot indices; the parser maintains a $(round, sender, recipient) \to \texttt{plan\_id} \mapsto \texttt{slot}$ map populated by the corresponding PROPOSALS message and uses it to attribute scores to the right slots. IMAP COSTS replies are similarly bounded by the slot set in the originating COST_REQUEST, so a responder that returns extra slots is not credited with extra leakage. Errand-occupied slots that admit a movable-displacement insertion are treated as feasible ($e = 1$) by IMAP COSTS, since the responder reports a finite cost rather than `None`; this matches IMAP's protocol semantics, where slot $k$ being feasible-via-displacement is exactly the information the message reveals.

### F.3 Reflection-calibrated VPS (LLM agents)

LLM-agent cheap-talk is unstructured, so the main LLM privacy metric does not attempt to recover beliefs from regex rules over messages. Instead, CalBench records measurement-only reflection calls. After each round in the reported mixed-team runs, each agent is asked, for every other target agent and every slot, how its belief changed about whether the target is occupied. The prompt requires a JSON array of signed integers in $\{-3, -2, -1, 0, 1, 2, 3\}$, where positive values mean "more willing to say occupied" and negative values mean "less willing to say occupied." These calls use `record_history=false` and are not inserted into later game context.

The calibration pipeline first extracts the emitted deltas and, when available, token log-probabilities for the numeric slot decisions. The VPS conversion is slot-equivalent: a maximal $\pm 3$ movement counts as one slot of raw belief movement. We also compute calibrated loss by checking the direction of movement against the target's actual calendar state: movement toward the truth counts as privacy leakage, while wrong-direction movement counts as zero calibrated leakage. Per-game target summaries subtract a fixed five-slot communication floor and report `excess_calibrated_vps_loss_total`, which is the privacy value used for model-based OpenSkill analyses.

### F.4 Slot-equivalent units and the prior

**Slot-equivalent units.** VPS treats every slot equally regardless of displacement cost. This matches the original VPS formulation (Maheswaran et al., 2006) and makes comparisons across uniform- and varied-cost calendars directly interpretable: a one-slot belief movement has the same privacy cost whether the occupied item is cheap or expensive to move.

| Source protocol | Message | Slot(s) updated | Evidence $e$ |
|---|---|---|---|
| DSM | PROPOSALS | each proposed slot | 1.0 |
| DSM | SCORES | each scored slot | 0 if score $\leq 0$, else $s/(D-1)$ |
| DSM | DECISION | chosen slot | 1.0 |
| IMAP | COST_REQUEST | (no update) | — |
| IMAP | COSTS | each requested slot | 0 if cost is None, else 1.0 |
| IMAP | DECISION | chosen slot | 1.0 |
| SD | PROPOSE | proposed slot | 0.85 |
| SD | REPLY | proposed slot | 0 if IMPOSSIBLE, else 1.0 |
| SD | PROPOSE_RESCHEDULE | target slot | 0.85 |
| SD | RESCHEDULE_REPLY | target slot | 0 if IMPOSSIBLE, else 1.0 |

Table 10: Typed-message belief updates. DSM and IMAP rules use $\alpha = 1$. SD PROPOSE and PROPOSE_RESCHEDULE use $\alpha = 0.70$, because proposing a slot is strong but not logically identical to a hard feasibility report; SD reply rules use $\alpha = 1$. COST_REQUEST carries no evidence about the sender's calendar (the initiator is asking, not telling), so it produces no update; the corresponding COSTS reply is matched against the request to bound the slot set being scored. DSM SCORES replies are matched against the originating PROPOSALS message to recover slot indices from plan ids.

**Prior choice.** We use $p_0 = 0.5$ throughout, reflecting maximal observer uncertainty about an unfamiliar calendar at round start. Sensitivity to this choice is bounded: under any prior $p_0 \in [0.3, 0.7]$, the relative ordering of clients on VPS is unchanged in our data; results in the main text use $p_0 = 0.5$.

### F.5 Output schema and reporting

The protocol-semantic pipeline emits four CSV tables per analysis run.

**`belief_evidence.csv`** Per-update rows. Columns: `trace_path`, `game_id`, `event_index`, `round`, `target_agent`, `observer_agent`, `slot`, `source`, `evidence`, `strength`, `belief_before`, `belief_after`. This table supports source-level decomposition of exact typed-protocol leakage.

**`pair_round_vps.csv`** Per-$(round, target, observer)$ rows. Columns: `trace_path`, `game_id`, `round`, `target_agent`, `observer_agent`, `target_is_participant`, `observer_is_participant`, `num_agents`, `num_slots`, `observations`, `prior_distance_to_ideal`, `posterior_distance_to_ideal`, `vps_loss`, `vps_loss_per_slot`, and `prior`.

**`game_summary.csv`** Per-$(trace, game)$ rows. Aggregates pair-round rows: `vps_loss_total`, `vps_loss_mean`, `participant_pair_vps_loss_total`, `participant_pair_vps_loss_mean`, and `observation_count`.

**`game_target_summary.csv`** Per-$(trace, game, target)$ rows. This table is used for per-agent rating inputs and reports `vps_loss_total`, `excess_vps_loss_total`, participant-pair totals, and the fixed floor subtracted from each game-target total.

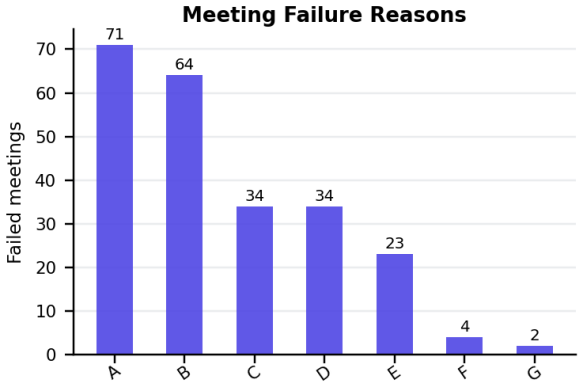


Figure 5: Meeting-level scheduling failure modes in the mixed-model CalBench traces. Each failed meeting is assigned one reporting reason. Letter labels denote: **A** = new meeting schedule mismatch; **B** = attempted blocked item reschedule; **C** = missing schedule action; **D** = meeting reschedule mismatch; **E** = new meeting schedule slot occupied; **F** = reschedule slot occupied; **G** = multiple reasons.

**Headline metrics.** In the main results we report mean per-game VPS loss as the primary privacy-leakage axis for direct comparability across uniform- and varied-cost calendars. We retain participant-pair-only summaries to distinguish within-meeting disclosure from accidental exposure involving non-participants.

## G Additional Results

### G.1 Completion-conditioned regret

Table 11 reports excess cost among agent-game seats that completed all of their assigned meet-

| Model | Full seats | Mean excess | Nonzero regret |
|---|---|---|---|
| Claude Sonnet 4.6 | 52 | 0.76 | 11.5% |
| DeepSeek V4 Pro | 62 | 0.27 | 19.4% |
| Gemini 3 Flash | 52 | 0.31 | 26.9% |
| Gemini 3.1 Pro | 50 | 0.85 | 28.0% |
| GPT-5.4 Mini | 38 | 6.00 | 42.1% |
| Llama 4 Maverick | 43 | 1.62 | 41.9% |
| Qwen3.6 Plus | 61 | 0.27 | 23.0% |

Table 11: Completion-conditioned regret. Rows include only agent-game seats with 100% completion. Mean excess is scheduled excess cost per successful meeting within those full-completion seats. Nonzero regret is the share of full-completion seats with any oracle excess cost.

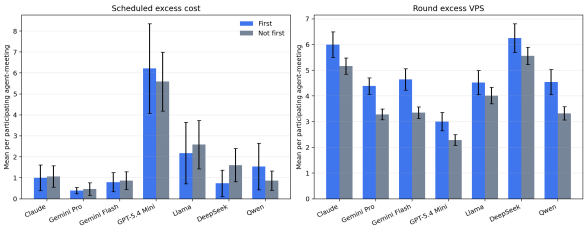


Figure 6: First-speaker effects by model. Bars report mean scheduled excess cost and mean round-level excess VPS per participating agent-meeting, stratified by whether the agent was the first speaker for that meeting. Error bars are 95% confidence intervals over participating agent-meeting rows.

| Model | Δ VPS | VPS $q$ | Δ cost | Cost $q$ |
|---|---|---|---|---|
| Claude | 0.83 | **0.006** | -0.03 | 0.952 |
| Gemini Pro | 1.10 | $2.84 \times 10^{-8}$ | -0.10 | 0.832 |
| Gemini Flash | 1.30 | $5.13 \times 10^{-8}$ | 0.07 | 0.901 |
| GPT-5.4 Mini | 0.72 | $2.32 \times 10^{-4}$ | 1.31 | 0.727 |
| Llama | 0.50 | 0.058 | -0.46 | 0.832 |
| DeepSeek | 0.69 | **0.033** | -0.99 | 0.546 |
| Qwen | 1.22 | $1.44 \times 10^{-5}$ | 0.67 | 0.727 |

Table 12: Paired first-speaker tests by model. Each test pairs an agent-game's mean first-speaker meetings with its mean non-first-speaker meetings. Values are first minus subsequent speaker; $q$-values are Benjamini–Hochberg corrected within metric.

ings. This isolates whether agents schedule well after conditioning on feasibility. Across the canonical mixed-agent cohort, 358 of 630 evaluated agent-game seats completed every assigned meeting. Among those full-completion seats, 26.3% still incurred nonzero excess cost relative to the oracle, and the mean excess cost was 1.20 per scheduled meeting.

### G.2 First-speaker effects

The sequential meeting protocol creates an exposure asymmetry: the first speaker often anchors the negotiation by proposing feasible slots or ruling out bad ones before seeing how much other participants are willing to reveal. We therefore stratify each participating agent-meeting by whether that agent was first in the meeting's speaker order. For this diagnostic, the unit is a participating agent-meeting: VPS is the round-level reflection-calibrated excess VPS for that target agent, and excess cost is the scheduled agent-meeting cost above the oracle schedule.

The aggregate effect is concentrated in privacy rather than welfare. Across all models, first speakers incur 4.71 excess VPS compared with 3.78 for subsequent speakers, a paired increase of 0.93 VPS per participating agent-meeting ($p = 9.26 \times 10^{-24}$, Benjamini–Hochberg $q = 2.22 \times 10^{-22}$). The same comparison is significant in both uniform-cost games ($\Delta = 1.03$, $q = 9.29 \times 10^{-14}$) and varied-cost games ($\Delta = 0.82$, $q = 1.33 \times 10^{-9}$). In contrast, excess cost is statistically indistinguishable by speaker role: descriptive means are 1.84 for first speakers and 1.86 for subsequent speakers, and the paired aggregate test is not significant ($\Delta = 0.10$, $p = 0.69$, $q = 0.83$).

### G.3 Communication volume by meeting and first-speaker role

### G.4 Prior-meeting rescheduling

Figure 9 reports how often agents move previously scheduled multi-agent meetings while coordinating later meetings. For each mixed-cohort game, we count the number of distinct prior meetings rescheduled by the selected model seat, then average this count across games for each model and setting. This metric captures rare but important cross-meeting repair behavior: rather than resolving conflicts only by moving local errands or selecting a different free slot, agents sometimes reopen an earlier multi-agent commitment to reduce disruption.

### G.5 Validation of own-slot disclosure regexes

The fairness analysis in §4 uses a deterministic message audit to measure whether agents propose their own slots and whether they disclose cost- or constraint-relevant information about those slots. This audit is separate from the reflection-calibrated VPS metric: it is a targeted diagnostic used to interpret GPT-5.4 Mini's fairness behavior. The extractor first detects slot mentions, then requires

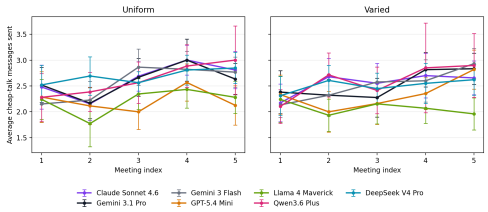


Figure 7: Average cheap-talk messages sent by selected model seats at each meeting index in the canonical mixed-agent cohort. The metric is included because increasing message counts across meetings indicate whether agents accumulate unresolved coordination burden over the sequence rather than treating each meeting independently. Mixed5 traces contribute all five model seats; Qwen and DeepSeek arena-entry traces contribute entrant agent 0.

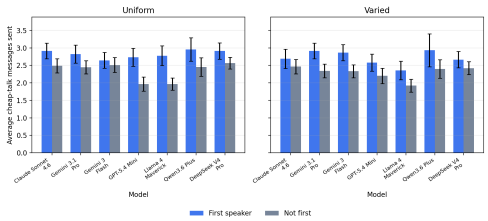


Figure 8: Average cheap-talk messages sent by selected model seats, stratified by whether the seat was the first speaker for that meeting. We collapse second and third speaker positions because first-speaker exposure is balanced by task construction, whereas later speaker positions are confounded with fixed seat identity in the canonical task suite.

first-person language (e.g., "I", "my calendar", "for me") before assigning one of three labels: own-slot proposal, own-slot availability, or own cost/constraint disclosure. The cost/constraint label includes qualitative cost language such as "difficult", "cheap", "blocked", "can move", and "move my errand"; exact numeric cost mentions are tracked separately and are rare.

To estimate extraction quality, we manually annotated a balanced validation sample drawn from the varied-cost mixed-model traces. For each label, we sampled 15 regex-positive messages and 15 regex-negative messages, then annotated whether the message actually contained the target phenomenon under the definitions above. Table 13 reports precision and recall on this audit sample with Wilson 95% confidence intervals. The regexes are intentionally conservative: precision is high for own-slot proposal and availability, but recall is moderate because the rules miss elliptical statements such as "both 12 and 13 are feasible for me"

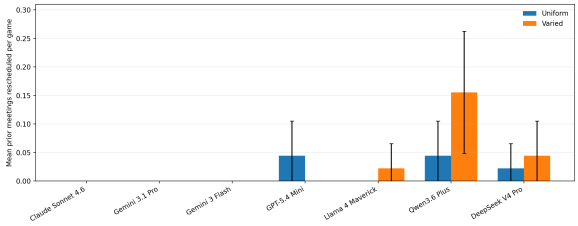


Figure 9: Mean number of prior meetings rescheduled per game by selected model seats in the canonical mixed-agent cohort. Higher values indicate more frequent cross-meeting repair, where an agent moves an already scheduled multi-agent meeting to accommodate a later one.

| Regex label | Precision | Recall |
|---|---|---|
| Own-slot proposal | 1.00 [0.80, 1.00] | 0.65 [0.45, 0.81] |
| Own-slot availability | 1.00 [0.80, 1.00] | 0.56 [0.37, 0.72] |
| Own cost/constraint | 0.80 [0.55, 0.93] | 0.63 [0.41, 0.81] |

Table 13: Manual validation of the regex labels used in the own-slot disclosure audit. Bracketed intervals are Wilson 95% confidence intervals computed on a balanced audit sample of 15 regex-positive and 15 regex-negative messages per label.

or "slot 4 it is; I'll make the necessary adjustment." The cost/constraint detector has lower precision because some messages mention another agent's cost or constraint in a sentence that also contains first-person language. These errors make the main claim conservative with respect to GPT-5.4 Mini: GPT-5.4 Mini is not lower on own-slot proposals, and its much lower cost/constraint disclosure remains large despite recall limits.

## H Homogeneous-team Behavioral Trace Audit

The behavioral taxonomy reported here comes from an earlier homogeneous-agent audit suite, including the probe condition used to surface semantic-context leakage. These traces are useful for identifying recurring language and coordination failures, but they are not used for the headline VPS comparisons in the main text because they do not include reflection-calibrated VPS scores. Table 14 summarizes the role of this suite.

Table 15 reports all twelve emergent patterns, ranked by occurrence count. Detection is non-exclusive: a single message may match multiple patterns.

*Errand mention* (M, rank 3): messages referencing "errand" as a calendar item type, revealing structure beyond abstract availability. Concen-

| Property | Value |
|---|---|
| Team composition | Homogeneous model teams |
| Games | 504 |
| Messages | 36,858 DMs |
| Models | 7 model families |
| Available signals | Message text, actions, trace-state detectors |
| Unavailable signals | Reflection-calibrated VPS |
| Use in paper | Behavioral taxonomy and mechanism audit |

Table 14: Provenance for the homogeneous-team trace audit. Counts from this suite are reported separately from the mixed-agent benchmark results.

trated in Gemini 3 Flash (2,058), Llama (1,548), and Qwen (862); nearly absent in Claude (34) and GPT (0).

*Cost gossip* (S, rank 4): an agent relays another agent's displacement costs or errand details to a third party—a compounded privacy violation. Gemini 3 Flash (678) and Gemini 3.1 Pro (516) are the primary sources.

*Cost-number disclosure* (M, rank 5): messages containing explicit numerical cost values (e.g., "cost of 100"), a direct system-prompt violation. Gemini 3.1 Pro accounts for 89% of all instances (712 of 797).

*Unilateral meeting displacement* (S, rank 8): an agent moves a previously scheduled meeting without coordinating with its other participants, causing a consistency violation. The dominant coordination failure mode; Qwen 3.6 Plus is most affected (33 of 74 instances).

*Errand-ID leakage* (M, rank 7): messages containing internal identifiers such as "Errand #10." Almost entirely a Llama issue (56 of 133 instances in varied-cost traces).

*Semantic label leakage* (M, rank 10): messages revealing personal event descriptions (e.g., "medical appointment," "library drop-off"). The rarest (35 total) but most sensitive privacy violation.

*Cascading failure* (S, rank 11): a single consistency violation propagates across multiple rounds. In one Qwen trace, a Meeting M2 displacement caused three consecutive round failures.

*Output truncation* (S, rank 12): an agent's JSON response is cut off before the actions array, producing a null scheduling decision. Almost exclusively Llama (10 of 12 instances), caused by 65-token response truncation.

# I Post-hoc LLM-as-judge privacy audit

This appendix records the external LLM-as-judge audit we ran as a robustness check on the main VPS measurements. The audit is not the primary privacy metric: protocol-semantic VPS remains the primary measurement for typed non-LLM reference protocols, and reflection-calibrated VPS remains the primary measurement for LLM agents. Instead, the audit asks whether an independent language model, given only the messages delivered to a recipient, detects a similar pattern of calendar-state leakage.

## I.1 Corpus and artifacts

We audited 180 initial-calibration games: the 90-game DSM-private baseline run and the 90-game mixed-agent model run used for the main cohort analysis. We did not include the later live-matchmaking games. The audited trace roots were:

- `calbench-mixed/baseline-dsm-private-calbench-mixed-001`
- `calbench-mixed/mixed5-calbench90-001`

We also preserved the run outputs in S3 under `s3://calbench-traces/calbench-mixed/_analysis/posthoc-judge-vps-180-20260525/`. The most important generated tables are:

- `judge_beliefs.csv`: row-level judge belief deltas.
- `judge_cases.json`: the delivered-message cases shown to the judge.
- `judge_beliefs_calibrated.csv`: judge deltas with truth-direction calibration.
- `metric_comparison.csv`: aggregate comparison against protocol-semantic and reflection-calibrated VPS.
- `judge_vs_reflection_precision.csv`: per-game-target comparison of judge and reflection precision.
- `judge_more_precise_than_reflection.csv`: filtered cases where the judge's nonzero deltas were more often truth-directed than self-reflection.
- `judge_reflection_cell_agreement.csv`: slot-belief cell agreement between judge and self-reflection.

| Rank | Pattern | Count | Level | Top models | Detection method |
|---|---|---|---|---|---|
| 1 | Availability gossip | 6,971 | Social | DeepSeek (1,782), Claude (1,643), Flash (1,067) | Third-agent ID + availability keyword within 60 chars |
| 2 | Confirmation tax | 6,537 | Msg | Flash (1,428), Gem. Pro (1,317), GPT (939) | $<$ 80 chars + agreement phrase match |
| 3 | Errand mention | 5,981 | Msg | Flash (2,058), Llama (1,548), Qwen (862) | Substring match: "errand" |
| 4 | Cost gossip | 1,757 | Social | Flash (678), Gem. Pro (516), Llama (245) | Third-agent ID + cost/errand keyword within 60 chars |
| 5 | Cost-number disclosure | 797 | Msg | Gem. Pro (712), Flash (72), Llama (12) | Regex: `cost[= ]+\d+` and variants |
| 6 | Qualitative flattening | 311 | Msg | GPT (102), Llama (71), DeepSeek (44) | Cost $\geq$ 100 errand displaced in varied-cost trace |
| 7 | Errand-ID leakage | 133 | Msg | Llama (56), others scattered | Regex: `[Ee]rrand\s*#\d+` |
| 8 | Unilateral displacement | 74 | Social | Qwen (33), Llama (18), GPT (15) | Non-empty consistency-violation field |
| 9 | Ambiguous consensus | 37 | Social | GPT (21), Llama (9), DeepSeek (7) | $\geq$ 2 distinct non-null slots in failed resolution |
| 10 | Semantic label leakage | 35 | Msg | DeepSeek (7), Qwen (2), Flash (1) | 27 curated semantic-context keywords |
| 11 | Cascading failure | 21 | Social | Qwen, GPT, Llama | Same meeting ID violated in $>$ 1 resolution per trace |
| 12 | Output truncation | 12 | Social | Llama (10), DeepSeek (2) | Agent slot is `null` in failed resolution |

Table 15: Full taxonomy of emergent behavioral patterns across 504 homogeneous-team games (36,858 DMs, 7 models), ranked by occurrence. *Level*: Message (individual message content) or Social (multi-agent dynamics). Detection is non-exclusive.

### I.2 Judge task

The judge script was `analysis/scripts/round_message_judge_vps.py`. We used the Vertex AI model `publishers/google/models/gemini-3.1-pro-preview`. Each judge API call corresponded to one recipient's delivered transcript for one game round, not to one slot. The prompt included the messages that were visible to that recipient in that round, including point-to-point DMs, participant group-chat messages, and all-agent broadcast messages. The judge then emitted belief updates for the other agents' 16 calendar slots.

The output scale matched the self-reflection VPS scale:

$$\Delta \in \{-3, -2, -1, 0, 1, 2, 3\},$$

where positive values mean the recipient should become more willing to believe that the target agent is occupied in that slot, negative values mean less willing to believe occupied, and zero means no belief movement. Raw judge VPS uses the slot-equivalent conversion $|\Delta|/3$. Calibrated judge VPS counts only movements in the truth-directed direction, using the target agent's actual calendar state.

Overall, the audit created 2,702 judge cases and 9,721 nonzero judged belief rows. The total raw judge VPS was 8,999.33 slot-equivalents, or 3.33 slot-equivalents per judged case on average. The mean absolute nonzero judged belief delta was 2.78 on the seven-point scale.

### I.3 Aggregate comparison

Table 16 summarizes the aggregate audit results. For DSM-private, the comparison point is the protocol-semantic VPS parser for typed DSM messages. For mixed5, the comparison point is the existing reflection-calibrated VPS measurement collected during the game runs.

The mixed5 judge found substantially more calibrated leakage than self-reflection: 4,490.33 calibrated slot-equivalents versus 1,752.17 from reflection-calibrated VPS. After subtracting the five-slot per-game-target communication floor, the mixed5 judge excess calibrated VPS was 2,552.33, compared with 1,491.33 for self-reflection. This suggests that the external judge is more sensitive to message-level implications than the acting models' own reported belief updates.

DSM-private shows the opposite calibration pattern: the judge's raw VPS is higher than the protocol-semantic parser, but its calibrated VPS is much lower. This is expected to be less directly interpretable because DSM messages expose typed feasibility and score information, while the judge is asked to express belief changes about occupied slots in natural language terms. Protocol-semantic parsing is therefore the appropriate primary measure for DSM.

### I.4 Truth-direction precision

For nonzero judge deltas, 4,639 of 7,870 mixed5 belief rows moved in the truth-directed direction, giving a nonzero precision of 0.589. The corresponding self-reflection precision was 0.528 over

| Dataset | Beliefs | Raw judge VPS | Calibrated judge VPS | Primary metric VPS |
|---|---|---|---|---|
| DSM-private | 1,851 | 1,344.67 | 327.67 | 863.64 |
| Mixed5 | 7,870 | 7,654.67 | 4,490.33 | 1,752.17 |

Table 16: External judge VPS compared with the primary privacy measurement for each audited dataset. The DSM-private primary metric is protocol-semantic uniform VPS. The mixed5 primary metric is reflection-calibrated VPS.

10,445 nonzero self-reflection slot deltas. At the game-target level, 447 targets could be joined between the two measurement procedures. The external judge was more precise than self-reflection for 256 game-targets, self-reflection was more precise for 182 game-targets, and the two were tied for 9 game-targets. The mean precision gap, judge minus self-reflection, was 0.016 overall and 0.174 among targets where the judge was more precise.

These results are useful for interpreting the role of self-reflection. The external judge is often a sharper detector of explicit message implications, but self-reflection still captures the model's own reported belief movement. The two measurements are therefore complementary rather than redundant.

### I.5 Cell-level agreement with self-reflection

We also computed agreement between the external judge and self-reflection at the slot-belief cell level, using the key $(\text{game\_id}, \text{observer\_agent}, \text{target\_agent}, \text{slot})$. Repeated rows for the same cell were summed before comparison, with mean deltas also retained in the output table. Table 17 reports the main agreement statistics.

| Agreement statistic | Value |
|---|---|
| Union cells compared | 24,727 |
| Judge nonzero cells | 6,828 |
| Self-reflection nonzero cells | 7,914 |
| Both nonzero | 4,472 |
| Nonzero-support Jaccard | 0.435 |
| Judge coverage of reflection nonzero | 0.565 |
| Reflection coverage of judge nonzero | 0.655 |
| Directional agreement when both nonzero | 0.643 |
| Cohen's $\kappa$, signs over full union | 0.353 |
| Pearson signed-delta correlation, full union | 0.290 |
| Pearson signed-delta correlation, both nonzero | 0.426 |

Table 17: Slot-belief cell agreement between post-hoc judge deltas and self-reflection deltas for the mixed5 90-game run.

Agreement is weak-to-moderate rather than interchangeable. The two methods share a meaningful signal: when both mark a slot-belief cell as nonzero, they agree on direction 64.3% of the time, and the signed-delta correlation rises to 0.426. However, the support overlap is incomplete, with 2,356 judge-only nonzero cells and 3,442 reflection-only nonzero cells. This supports the interpretation that self-reflection is not merely a noisy external annotation proxy. It measures an internally reported belief update, while the external judge provides an independent audit of what an outside reader could infer from the same delivered messages.

### I.6 Takeaways

The audit supports three conclusions. First, external LLM judging can be run on both typed protocol agents and LLM agents because it only requires delivered messages and trace-state calibration, although typed protocol agents are still best measured by their protocol semantics. Second, for LLM cheap-talk, the judge detects more calibrated leakage than self-reflection, especially in explicit availability and free-slot disclosures. Third, the moderate but incomplete agreement between judge and self-reflection justifies using reflection-calibrated VPS as the main model-based privacy metric while reporting external judging as a robustness audit rather than a replacement.

**Algorithm 1** One game round in the calendar scheduling harness.

**Require:** meeting $m$, speaker order $P$ (participants), all agents $A$, `max_turns`

Emit ROUND_START event
**// CHEAP_TALK PHASE**
$t \leftarrow 0$;  *has_activity* $\leftarrow$ TRUE
**while** *has_activity* **and** $t <$ `max_turns` **do**
  *has_activity* $\leftarrow$ FALSE
  **for all** $a \in P$ in speaker order **do**
    msg $\leftarrow$ ROUNDSTART($m$, cal$_a$, $t$) **if** $t$=0 **else** TURN(inbox$_a$, $t$)
    Deliver msg; emit TURN_START
    result $\leftarrow a$.`turn`($t$); emit TURN_END
    **for all** message $d \in$ result.tool_calls **do**
      Route $d$ to visible recipients; emit message event
      *has_activity* $\leftarrow$ TRUE
      **if** $d$ activates non-participants **then** queue those agents
      **end if**
    **end for**
  **end for**
  **for all** non-participant $a$ queued this turn **do**
    Deliver TURN(inbox$_a$, $t$); emit TURN_START
    result $\leftarrow a$.`turn`($t$); emit TURN_END
    Process any DMs in result (same as above)
  **end for**
  $t \leftarrow t + 1$
**end while**
**// VOLUNTARY PHASE**
**for all** non-participant $a$ activated during CHEAP_TALK **do**
  Deliver VOLUNTARY($m$, cal$_a$); emit DECIDE_START
  result $\leftarrow a$.`voluntary_decide`($m$); emit DECIDE_END
  actions $\leftarrow$ [calls with type `reschedule`]
  **for** attempt $= 0$ to `decision_retries` **do**
    (ok, err) $\leftarrow$ VALIDATEBATCH(cal$_a$, actions, *require_schedule*= FALSE)
    **if** ok **then** apply actions to cal$_a$; emit BATCH_APPLIED; **break**
    **else** emit BATCH_REJECTED; result $\leftarrow a$.`retry_decide`($\cdot$)
    **end if**
  **end for**
**end for**
**// DECISION PHASE**
**for all** $a \in P$ **do**
  Deliver DECISION($m$, cal$_a$); emit DECIDE_START
  result $\leftarrow a$.`decide`($m$); emit DECIDE_END
  actions $\leftarrow$ result.tool_calls
  **for** attempt $= 0$ to `decision_retries` **do**
    (ok, err) $\leftarrow$ VALIDATEBATCH(cal$_a$, actions, *require_schedule*= TRUE)
    **if** ok **then** stage (cal$_a$, actions, cost); **break**
    **else** emit BATCH_REJECTED; Deliver RETRY(attempt, err); result $\leftarrow a$.`decide`($m$)
    **end if**
  **end for**
**end for**
Commit all staged decisions atomically
**// RESOLUTION PHASE**
*slots* $\leftarrow$ {scheduled slot chosen by each $a \in P$}
**if** $|$*slots*$| = 1$ **then** round succeeds; update game state
**else** record consistency violation; do not update
**end if**