GenCoord: Skill-Path Commitments under Private Information

Peng He*
Tsinghua University
hepeng@tsinghua-wx.org

Junning Zhu*
Beijing Normal-Hong Kong Baptist University
t330025113@mail.bnbu.edu.cn

Haohan Yuan
University of North Carolina at Charlotte
hyuan3@charlotte.edu

Jianpeng Liang
University of California San Diego
jil652@ucsd.edu

arXiv:2608.22055v2 [cs.AI] 25 Aug 2026

Abstract

Suppose one embodied agent knows what must be built, while its teammate alone knows which transformation its workcell can perform. Neither local view determines who should act, what should be handed off, or how the joint task should continue. We introduce *GenCoord*, which turns the task consequence of such private facts into an executable skill-path commitment. A local Qwen3.5-0.8B model emits a multi-step SELF plan and peer REQ; bounded feedback conditions route revision when the deciding capability is peer-local. The resolved commitment is parsed, checked, canonically materialized, compiled to Mineflayer skills, and verified by handoff and terminal state. Counterfactual interventions that hold the world, call schedule, and executor unchanged make requester revision and receiver execution follow the injected task consequence in both directions. Across three independently trained seeds, correct capability feedback closes the paired local-information gap from 50% to 100%. Multi-step commitments improve held-out-template success by 6.9 points while reducing model decisions by 32%. At matched closed-loop quality on 128 held-out semantic clusters, Short DSL reduces peer traffic by 92.8% and median time-to-commitment by 68.2% relative to controlled free-form communication. These results identify executable task consequences as the coordination unit connecting distributed local reasoning to verified joint action.

CCS Concepts

• **Computing methodologies** → **Multi-agent systems**; *Natural language generation.*

Keywords

multi-agent coordination, embodied agents, semantic communication, language-model agents, Minecraft

1 Introduction

Imagine two Minecraft agents fulfilling an order for a crafting table. Agent *A* sees the order; agent *B* alone knows whether its workcell can transform oak planks or can only receive the finished table. In the first case, *A* should hand off planks and *B* should craft. In the second, *A* must craft first and hand off the table. Agent *A* receives the same local input in both cases, yet the correct actor, handoff item, and continuation all change.

This small example captures a general problem in embodied teams. Goals, tools, workcells, inventories, and execution conditions are distributed across agents. The team can have a well-defined joint route even when no individual view determines it. Coordination must reveal the task consequence of a private fact: who acts, what crosses the handoff boundary, where it goes, and which suffix follows.

Centralized planning can reveal that consequence by assembling local contexts; free-form negotiation can reveal it through repeated state, intent, and plan exchange. GenCoord instead makes the consequence itself the coordination object, preserving the fields that directly determine execution.

The path-determining fact may reside at either endpoint of an edge. A sender-local consequence travels forward inside a peer request, after which the receiver generates its downstream skill path. A peer-local capability consequence travels back through bounded feedback, after which the requester revises the division of work. Both directions require one explicit object whose semantics survive communication, resolution, and execution.

GenCoord turns that object into an executable skill-path commitment. Its primary backend uses a local Qwen3.5-0.8B model [19] to compose goals, capabilities, objects, history, received messages, and shared task structure into variable-length role-local paths. The sender jointly emits a multi-step self-plan and peer request:

```
SELF resource.obtain(q=4,item=oak_planks)
   > resource.deliver(q=4,item=oak_planks,to=agent_b)
REQ agent_b craft.item(q=1,input=oak_planks,
   item=crafting_table)
   > resource.deliver(q=1,item=crafting_table,
      dst=order_chest)
```

Short DSL is the peer-facing executable interface for this canonical schema. Before resolution it carries a proposed SELF+REQ route; after resolution the same schema carries the role-local obligations that enter execution. In the main forward route, the receiver conditions on REQ and generates its downstream SELF path. In the paired capability diagnostic, a transparent Accept/Reject/Counter response conditions requester revision.

The resolved object follows a deterministic grounded path: parse, schema check, canonical materialization, skill compilation, Mineflayer execution [18], verified handoff, and terminal-state readback. Figure 1 summarizes the two directions and their shared execution closure.

Across three independently trained seeds, bounded capability feedback raises paired local success from 50% to 100%, and counterfactual interventions in both directions redirect the resolved route exactly as the injected task consequence specifies. Multi-step commitments improve held-out-template success by 6.9 points while reducing model decisions by 32%. Across 128 held-out clusters, Short DSL matches JSON and controlled free-form communication

*Equal contribution.

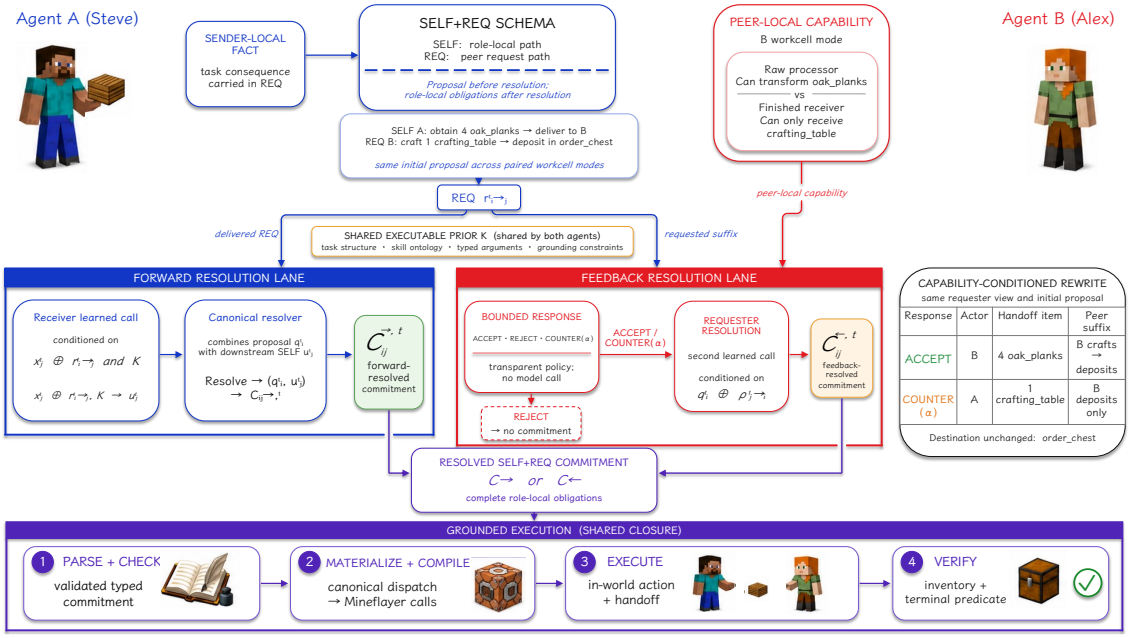

Figure 1: One SELF+REQ schema carries sender-local task consequences forward and peer-local capability consequences back before shared grounded execution.

in closed-loop quality while reducing peer traffic by 92.8% and median time-to-commitment by 68.2% relative to free-form. We make three contributions:

- **Private information.** We characterize paired local ambiguity and show how the executable consequence of a private fact crosses the agent boundary, forward in a request or back through bounded feedback.
- **Executable commitments.** GenCoord represents sender and peer obligations in one multi-step SELF+REQ object; Short DSL preserves that object from proposal and resolution through grounded execution.
- **Mechanism and systems evidence.** Counterfactual interventions establish a content-to-route causal link in both directions; horizon, communication-surface, and backend experiments show how the same grounded object can be extended, encoded, and formed efficiently.

2 Related Work

Commitments and task allocation. Joint-intention theory, SharedPlans, and STEAM model teamwork through shared goals, partial plan knowledge, and communication decisions [7, 12, 22]. Contract Net, multi-robot task allocation, and consensus-based auctions coordinate through bids, utilities, contracts, or assignments [5, 10, 20]. Azorus combines formal commitments, information protocols, and BDI programming [6]. We use commitment operationally: a resolved SELF+REQ object assigns sender and peer task obligations that are discharged by verified handoff and terminal completion.

Embodied and Minecraft collaboration. Voyager builds an executable code-skill library for open-ended single-agent learning in Minecraft [23]. RoCo uses LLM dialogue for multi-robot task and waypoint planning, with feedback from a motion planner [17]. MindCraft examines situated theory-of-mind dialogue [1]; MindAgent evaluates coordination and scheduling [11]; TeamCraft provides multimodal collaborative tasks [16]; and MINDcraft introduces the MineCollab benchmark for natural-language, action-by-action collaboration [25]. VillagerAgent and CausalMACE organize longer execution through graph-structured dependencies and causal planning [2, 8]. Gated Coordination decides when local events should escalate to shared coordination [14], while TickingCollab studies time-sensitive complementary collaboration [26]. GenCoord studies what crosses the coordination boundary: a multi-step task object that binds local evidence to peer execution and handoff consequence.

Communication content and representation. Differentiable communication and bottleneck objectives learn task-specific channels end to end [9, 21, 24]. Alternative formats can improve LLM reasoning and communication [3]; Optima jointly optimizes communication quality and cost [4]; and OPTiMACS learns task-aware message representations [13]. AgenticCache removes repeated planning from the critical path through reusable transitions [15]. These lines study when agents communicate and how messages are encoded; GenCoord isolates what the peer must receive: an inspectable task consequence whose execution can be intervened on and verified end to end.

3 Coordination under Private Information

3.1 Local Contexts

Let $\mathcal{A} = \{a_1, \ldots, a_n\}$ be a team. At coordination round t, agent a_i has local context

$$x_i^t = (o_i^t, h_i^t, m_i^t),$$

where o_i^t is its observation, h_i^t its execution history, and m_i^t received task messages. Agents share an executable prior $\mathcal{K}$ containing a skill ontology, task structure, typed arguments, and grounding constraints. Instance goals, capabilities, inventories, and bindings remain local until their task consequences cross a coordination boundary.

3.2 Paired Ambiguity

Let z denote a task instance and $\Pi^\star(z;\mathcal{K}) \subseteq \Pi(\mathcal{K})$ the feasible goal-reaching paths allowed by the shared prior. A coordinator must form some $\hat{\pi} \in \Pi^\star(z;\mathcal{K})$ from information distributed across local contexts. The path-determining fact induces an execution-relevant binding $\beta(z)$ over the fields that select actor, handoff object, destination, and continuation. Under the fixed prior $\mathcal{K}$ and canonical schema used here, a cross-boundary object is coordination-sufficient when it identifies a resolved route in $\Pi^\star(z;\mathcal{K})$. GenCoord communicates $\beta(z)$ together with the role-local paths that discharge it.

Proposition 1 (Paired local ambiguity). *Consider equiprobable instances z_1, z_2 and an agent a_i with identical model-visible contexts $x_i(z_1) = x_i(z_2)$. Let $F_k := \Pi^\star(z_k;\mathcal{K})$. If $F_1 \cap F_2 = \emptyset$, then any a_i-local policy whose output distribution is identical under the two views has pairwise expected success at most $1/2$.*

Proof sketch. Let μ be the common output distribution. Since $F_1 \cap F_2 = \emptyset$, $\mu(F_1) + \mu(F_2) \le 1$, and the equiprobable success is at most $\frac{1}{2}\mu(F_1) + \frac{1}{2}\mu(F_2) \le \frac{1}{2}$. Deterministic policies are point-mass special cases.

The proposition characterizes whichever endpoint lacks the path-determining fact. In the main forward suites, the receiver cannot recover the active binding from its local view by construction. In the Goal × Capability diagnostic, paired requester views are identical while the peer capability changes the feasible actor, handoff object, and suffix.

3.3 Commitment Object and Resolution

A grounded task is a tuple $\tau = (a, k, \mathbf{v}, \ell, P)$ containing an actor a, skill k, typed arguments $\mathbf{v}$, grounding location ℓ, and predecessor set P. Let $\mathcal{T}_\mathcal{K}^*$ denote finite sequences supported by $\mathcal{K}$. A sender proposes

$$q_i^t = (s_i^t, r_{i\to j}^t), \qquad s_i^t, r_{i\to j}^t \in \mathcal{T}_\mathcal{K}^*,$$

where s_i^t is its proposed self path and $r_{i\to j}^t$ the requested peer path. The same canonical schema supports two resolution directions.

Forward resolution. When the sender holds the deciding fact, the receiver observes the delivered request and generates a downstream self path u_j^t. The joint object is

$$C_{ij}^{\rightarrow,t} = \mathrm{Resolve}_{\rightarrow}(q_i^t, u_j^t).$$

$\mathrm{Resolve}_{\rightarrow}$ returns a commitment only when the receiver identity and normalized SELF suffix realize the request, handoff fields align, and predecessor links admit an acyclic merge.

Feedback resolution. When the peer holds the deciding capability, it returns $\rho_{j\to i}^t \in \{\textsc{Accept}, \textsc{Reject}, \textsc{Counter}(\alpha)\}$, with α selected from bounded executable alternatives. The requester then produces a revised proposal $q_{i|\rho}^t = (s_{i|\rho}^t, r_{i\to j|\rho}^t)$, and

$$C_{ij}^{\leftarrow,t} = \mathrm{Resolve}_{\leftarrow}\big(q_i^t, \rho_{j\to i}^t, q_{i|\rho}^t\big).$$

Accept requires the revision to preserve the requested branch, Counter requires it to realize the selected alternative, and Reject yields no commitment. Missing or conflicting obligations also yield no resolved object. Resolution aligns role, route, and handoff across the initial and revised proposals; the checker validates typed fields and the materializer instantiates the dispatch plan.

4 GenCoord

4.1 Local Proposal Generation

A local autoregressive model G_θ receives x_i^t and $\mathcal{K}$, generates a string $\hat{y}_i^t$, and parses it into the initial proposal:

$$\hat{y}_i^t = G_\theta(x_i^t, \mathcal{K}), \qquad q_i^t = \mathrm{Parse}(\hat{y}_i^t) \in \mathcal{Q} \cup \{\bot\}.$$

The input explicitly identifies the decision agent, local observation, history, incoming messages, peer, and shared task prior. The model maps local semantics to the task consequence $\beta(z)$, binds active objects and destinations, composes multi-step paths, allocates transformations, and chooses the inter-agent cut represented by SELF+REQ.

4.2 Generative Task-Path Composition

The shared prior supplies legal skills and coarse dependencies; the learned backend instantiates the current path. For a bilateral episode, write a feasible joint path as a sender prefix, an inter-agent handoff, and a receiver suffix. GenCoord selects the participating actor for each transformation, grounds objects and destinations, and places the cut between role-local paths. This cut is task dependent: Destination changes where the suffix terminates; Recipe changes the transformation and handoff object; Allocation changes the actor and residual work; Active Branch changes the continuation released after handoff.

The resulting object carries more than an atomic assignment. It records the local steps required before the boundary, the peer obligation activated at the boundary, and the dependencies that connect both. A sender-local fact selects and transmits the suffix directly. A peer-local capability consequence can move the cut, changing which prefix the requester must complete before handoff. Both cases therefore use the same semantic operation—composition of complementary role-local paths around an explicit coordination edge.

4.3 Resolution across the Agent Boundary

Forward resolution. This is the main protocol used by the request, horizon, representation, and factor experiments. The sender generates $q_i^t = (s_i^t, r_{i\to j}^t)$. After receiving $r_{i\to j}^t$, the receiver makes the second learned call:

$$\hat{y}_j^t = G_\theta(x_j^t \oplus r_{i\to j}^t, \mathcal{K}), \qquad u_j^t = \mathrm{Parse}(\hat{y}_j^t),$$

Table 1: Capability-conditioned commitment rewrite. The matched rows share the requester view and initial proposal; the counterfactual condition injects the paired response.

Peer mode / response	Actor	Handoff item	Peer suffix
Raw processor / ACCEPT	B	oak planks	craft table → deposit
Finished receiver / COUNTER	A	crafting table	deposit crafting table

where u_j^t contains the receiver's downstream SELF path. The canonical resolver joins sender and receiver obligations into $C_{ij}^{\rightarrow,t}$ under the role, path, handoff, and dependency conditions above.

Feedback resolution. The Goal × Capability diagnostic uses the reverse information direction. A transparent capability policy maps the requested suffix and peer-local capability to ACCEPT, REJECT, or COUNTER(α) without a model call. For accepted or countered proposals, the requester performs a second learned call:

$$\hat{y}_{i|\rho}^t = G_\theta(x_i^t \oplus q_i^t \oplus \rho_{j\to i}^t, \mathcal{K}),$$
$$q_{i|\rho}^t = \mathrm{Parse}(\hat{y}_{i|\rho}^t).$$

The resolver then forms $C_{ij}^{\leftarrow,t}$ from the initial proposal, response, and revised proposal under the conditions in Section 3.3. The bounded response communicates the capability consequence for the current request and can move the inter-agent cut by changing the transformation actor, handoff object, and continuation. Because the response is explicit, counterfactual feedback can be injected while the requester view, initial proposal, executable world, and two-call schedule remain fixed. Table 1 shows the affected fields.

4.4 Short DSL

The peer-facing grammar is

```
SELF <task> [ > <task> ...]
REQ <agent> <task> [ > <task> ...]
```

where each task is a hierarchical skill path with typed arguments. Short DSL has four properties.

Role completeness. A sender output couples its proposed self path with the requested peer contribution; a receiver output records the downstream obligation it accepts.

Semantic sufficiency. Actor, skill path, object, quantity, destination, binding, and dependency order are explicit because they determine executable behavior.

Compositional horizon. Variable-length paths express acquisition, transformation, handoff, and deposit in one commitment, allowing the model to compose multi-step paths before acting.

Deterministic grounding. For every legal object under the fixed prior and canonical schema, parsing yields a canonical object and the codec preserves task semantics:

$$\mathrm{Decode}_{\mathcal{K}}(\mathrm{Encode}_{\mathcal{K}}(q)) \equiv_{\mathrm{sem}} q, \qquad q \in \mathcal{Q}_{\mathrm{valid}}(\mathcal{K}).$$

Here $\equiv_{\mathrm{sem}}$ denotes equality of normalized ordered tasks, roles, typed arguments, bindings, destinations, and dependencies. The schema carries a proposal before resolution and role-local obligations afterward, with dynamic arguments attached to the consuming skill.

Operational commitment semantics. A resolved object assigns role-specific obligations with observable discharge: the sender prefix closes at verified handoff, and the peer suffix closes at terminal completion. ACCEPT preserves the division of work; COUNTER changes its task fields.

4.5 Commitment Backends

Direct Short DSL predicts the complete role-local path at both learned stages. Factor-Code + Rule instead predicts one stage-specific binding code at the sender and one at the receiver; a deterministic composer expands those codes through the shared task structure into the same canonical Short DSL object. Both backends use the same model-visible context, two-stage forward schedule, peer-facing interface, and grounded execution stack. When a code identifies one branch of the task structure, composition recovers its actor, path, object, destination, and dependency fields. Exact target-field names appear in the supplement.

4.6 Grounded Execution

The resolved commitment follows

$$\text{generate} \to \text{parse} \to \text{schema check} \to \text{materialize}$$
$$\to \text{compile} \to \text{execute} \to \text{verify}.$$

The checker validates the resolved commitment against a typed execution schema: agent identity, skill availability, arguments, quantities, bindings, destinations, and predecessor constraints. A passing object is a *validated commitment*. The selected binding and shared task structure then canonically materialize an equivalent dispatch plan, which the compiler lowers to Mineflayer skills while preserving dependency order. In the running example, validation preserves the selected actor and either the oak-planks or crafting-table handoff before materialization. Handoff succeeds only when the recipient inventory reflects the intended item and quantity; the episode succeeds only when the terminal world predicate holds. Figure 2 shows this runtime closure.

5 Experimental Setup

5.1 Environment and Tasks

We evaluate on the official Minecraft Java 1.21.4 server with Mineflayer 4.37.1. Each episode contains two agents, local observations, a shared skill ontology and task prior, and a mandatory in-world handoff. The environment is static and communication reliable, isolating how distributed task facts enter a joint plan. Four families vary distinct commitment fields: Destination changes the target location; Recipe changes the transformation and handoff item; Allocation changes the actor and remaining work; Active Branch changes the downstream continuation. Each family contains two templates. Figure 3 summarizes the coverage; complete schemas and bindings appear in the supplement.

5.2 Data and Training

Targets are generated programmatically from task templates, sampled local facts, and canonical task specifications. Public Minecraft resources inform the ontology and scenario design. We use no language-model teacher or manually annotated reciprocal requests. Forward records contain two learned stages: a sender-local context

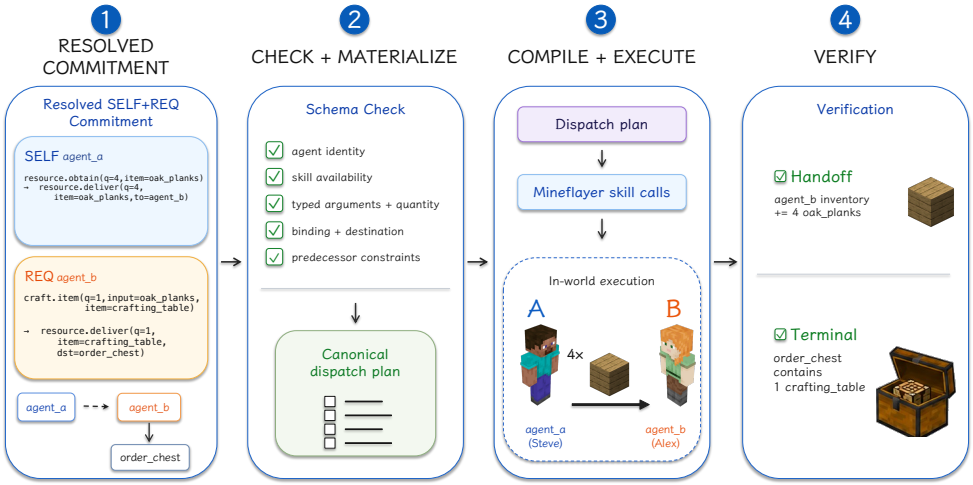

Figure 2: Grounded closure from a schema-validated commitment through canonical materialization to verified execution.

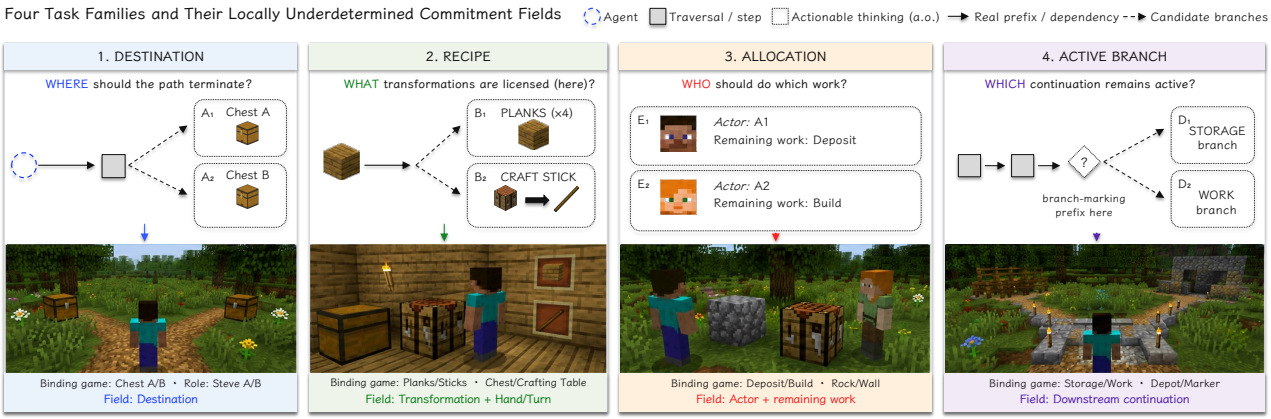

Figure 3: Four Minecraft families vary the task consequence carried across the agent boundary.

paired with its SELF+REQ target, and a receiver context augmented with the delivered REQ paired with its downstream SELF target. The Goal × Capability diagnostic uses a separate requester-initial, transparent-response, and requester-revision dataset.

Task templates yield 480 semantic training clusters under two role permutations and 1,920 forward rows: 960 sender-proposal rows and 960 receiver-after-request rows. Training uses two epochs and 240 optimizer steps; full optimization settings appear in the supplement. Input tokens are masked from the causal-LM loss. The single-step control adds 960 intermediate sender rows, for 2,880 rows and 360 optimizer steps. Controlled free-form receives 2,400 rows and 300 optimizer steps under the same two-epoch schedule. Main learned conditions use three independent seeds.

The submission-scale representation pool contains 128 held-out semantic clusters—16 per template—with both role permutations and three seeds, yielding 768 views per representation and 2,304 closed-loop episodes across the three communication surfaces. Mechanism evaluations use 160 clusters for request interventions, 80 held-out-template clusters for commitment horizon, and 60 unseen binding combinations. The Goal × Capability suite contains 40 semantic clusters—two goals, two workcell modes, and ten world variants—organized as 20 matched capability-counterfactual pairs. A-only planning, correct feedback, counterfactual feedback, and centralized full information each evaluate all 40 clusters under two role permutations and three independently trained seeds: 240 episodes per condition and 960 total. The counterfactual condition injects the bounded response from the paired workcell mode while preserving the requester input, initial proposal, and executable world. Semantic cluster remains the independent unit, with seed and role as repeated observations; the supplement additionally reports a 20-pair block-bootstrap sensitivity. A separate 128-cluster pool supports the matched backend comparison.

The pools separate distinct transfers: new bindings within known templates, held-out templates, unseen factor cross-products, and

unseen transform-order motifs. Each retains its own independent cluster count.

5.3 Comparators

The representation comparison matches the Qwen3.5-0.8B base model, semantic clusters, canonical task fields, two-epoch budget, skill interface, checker, materializer, executor, and verifier. GenCoord–DSL emits Section 4.4's grammar; GenCoord–JSON serializes the same fields; controlled multi-turn free-form expresses the same bilateral task semantics in natural language until a shared extractor recovers the commitment. It is a communication-surface comparator under the same model, task, and executor; MINDcraft and MineCollab remain system-level related work.

Mechanism conditions remove `REQ`, replace it with an executable same-template alternative, inject counterfactual feedback from the paired capability mode, or shorten the commitment horizon. Rule-Minimal-Request forwards the sender's private binding with zero model calls and uses the public composer to materialize both agents' paths. The Explicit-Factor Composer receives canonical factors directly. Factor-Code + Rule uses the same 1,920 rows, three seeds, two epochs, 240 steps, and forward stages as Direct Short DSL. In this structured suite, model-visible private-fact labels align one-to-one with the shared binding codebook; generated target length is the measured representation difference.

Each comparison changes one scientific layer. Request intervention changes delivered task content while preserving schedule and surface validity. Horizon changes whether the joint route is committed before execution or regenerated at an intermediate state. Short DSL versus JSON is a serialization comparison over the same canonical fields, two learned stages, and execution stack. Controlled free-form is an end-to-end communication-surface comparison under the same base model, task, and executor; it also changes turn structure, context growth, commitment extraction, calls, and training budget. Backend comparison changes the learned target while preserving the peer-facing Short DSL message. Centralized full information changes where local contexts are assembled.

5.4 Metrics and Statistics

Terminal success requires the world-state goal predicate after the complete episode. Raw validity records parsing. Commitment validity checks actors, skills, arguments, bindings, and dependencies against the canonical task schema and, when present, the delivered request or feedback. In feedback interventions this validity is response-conditioned; terminal success separately tests consistency with the unchanged true-world capability. Verified handoff requires the intended inventory transfer. Online metrics are model calls, input/output tokens, peer-directed wire bytes, and time-to-commitment (TTC). For structured methods, wire bytes count the inter-agent `REQ` payload; for controlled free-form, they sum all agent-to-agent dialogue before commitment extraction. Complete episode time ends at terminal readback and is reported descriptively.

Calls, tokens, wire bytes, and TTC are measured on a 40-cluster same-card sample with hot batch-1 models on an NVIDIA RTX 4090D, totaling 720 episode runs and 1,560 model calls. Semantic cluster is the independent unit; seed and role views are repeated observations. Paired comparisons use cluster-bootstrap 95% intervals, and all-success pools receive exact binomial lower bounds.

6 Results

Private information creates a 50% ambiguity ceiling; task content selects the resolved path in both directions of the edge; and a longer commitment horizon removes online replanning. Under this mechanism chain, Table 2 reports the quality-matched headline result: all three communication surfaces complete 128/128 held-out clusters, while Short DSL has the lowest online coordination cost.

6.1 Feedback Content Selects Requester Revision

The Goal × Capability suite contains 40 semantic clusters arranged as 20 matched capability-counterfactual pairs. Within each pair, the requester's model-visible input and initial proposal are byte-identical while the peer's private workcell mode changes the feasible actor, handoff object, and suffix. Four conditions evaluate all 40 clusters under two role permutations and three independently trained seeds, yielding 240 episodes per condition and 960 total: A-only planning, correct bounded feedback, counterfactual feedback, and centralized full information.

Every seed reproduces the 50/100/0/100 profile with descriptive seed-level SD 0.0. All 960 runs are parse-valid, planning-complete, and fallback-free. Correct feedback and centralized full information each succeed on 240/240. A-only succeeds on 120/240 and chooses the wrong craft actor on the remaining 120. In the feedback conditions, every resolved commitment is valid relative to the injected response: correct feedback also matches the true workcell, whereas counterfactual feedback is inconsistent with it and fails all 240 episodes with the wrong actor. Cluster-bootstrap effects are +50.0 points over A-only (95% CI $[35, 65]$), +100.0 over counterfactual feedback (95% CI $[100, 100]$), and 0.0 versus centralized full information (95% CI $[0, 0]$).

The intervention isolates feedback semantics. With the requester view, initial proposal, executable world, and two-call schedule unchanged, final commitments follow counterfactual feedback in 240/240 episodes: craft actor, handoff item and count, and peer suffix match the paired workcell mode, while goal fields remain fixed. The bounded capability consequence therefore causally controls the inter-agent cut in this diagnostic. Centralized planning uses one call and aggregates a mean 956 B of state; feedback resolution uses two local calls and a mean 103 B response (p95 118 B). Figure 4(a) summarizes the causal and full-information references.

6.2 Delivered Binding Determines Peer Execution

On 160 semantic clusters, the true multi-step request reaches 100% terminal success. Removing `REQ` while retaining the sender self-plan yields 50%; each template contains two balanced admissible bindings, making this a construction-level ambiguity ceiling. Replacing the request with the executable alternative from the same template—the request that would be correct under the other private fact—preserves message presence, protocol schedule, schema, approximate length, and call count, yet success falls to 0%. The paired

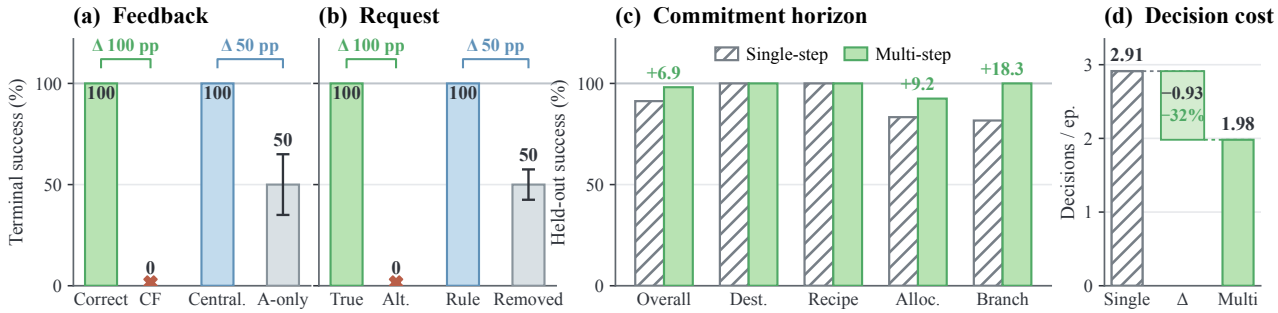

Figure 4: Mechanism evidence: feedback content selects requester revision, request content selects peer execution, and longer commitments reduce online decisions.

gains are +50.0 points over request removal (95% CI [42.5, 57.5]) and +100.0 points over the same-template alternative (95% CI [100, 100]).

A sender-local deterministic rule also reaches 100% with zero model calls by forwarding the correct private binding; the public composer materializes the two role-local paths. Under template shift, the rule remains at 100% while GenCoord reaches 98.1%, a paired difference of −1.9 points with 95% CI [−4.2, −0.2]. Receiver execution follows the delivered binding one-to-one whenever an executable plan is formed: same-template alternatives redirect all 960 confirmatory seed-role episodes and all 471 executable held-out episodes to the alternative path. GenCoord's nine held-out failures all occur before executable-plan formation, with zero wrong-binding failures. Within this intervention, the delivered binding is the causal control variable for peer execution; the backend and protocol surface determine whether a complete executable commitment is formed. Figure 4(b) summarizes the four conditions.

6.3 Multi-Step Commitments Reduce Online Decisions

Under their respective closed-loop training protocols, multi-step commitments reach 98.1% held-out template success, compared with 91.3% for the separately trained single-step controller; the paired difference is 6.9 points with a 95% interval of [2.9, 10.8]. Multi-step uses 1.98 model decisions per episode versus 2.91, a 32% reduction; the paired difference (multi-step minus single-step) is −0.931 with 95% CI [−0.971, −0.892]. The advantage persists even though the single-step model receives 2,880 supervision rows and 360 optimizer steps, compared with 1,920 rows and 240 steps for multi-step, including an explicit intermediate sender-after-obtain stage.

The gain concentrates where the hidden fact changes actor or continuation: Allocation improves from 83.3% to 92.5%, and Active Branch from 81.7% to 100%, while both variants reach 100% on Destination and Recipe. The result identifies commitment horizon as an online control variable: committing farther both improves transfer and removes a replanning stage. Figure 4(c) summarizes success and online decisions.

6.4 Quality-Matched Coordination Cost

Across 2,304 closed-loop episodes, Short DSL, JSON, and controlled free-form each complete 128/128 held-out semantic clusters with 100% raw validity, commitment validity, and verified handoff and with no repair or fallback. The exact-binomial 95% lower bound is 0.972, and role-swap success differs by 0.0 for every surface. This pool locks closed-loop quality before comparing online representation cost. Table 2 separates the 128-cluster quality and episode measurements from the 40-cluster same-card online-cost sample.

With closed-loop quality fixed, Short DSL improves on controlled free-form on every reported online cost. Free-form receives 2,400 training rows and 300 updates, versus 1,920 and 240 for Short DSL; Short DSL reduces calls by 20%, output tokens by 70.4%, wire bytes by 92.8%, median TTC by 68.2%, and descriptive median episode time by 17.4%. Relative to JSON, it reduces output tokens by 70.6%, wire bytes by 74.9%, median TTC by 67.9%, and descriptive median episode time by 17.0%. The structured rows share sender-proposal and receiver-after-request calls and canonical fields, isolating serialization cost. Free-form measures the full surface, including multi-turn context, extraction, calls, and training budget. Median per-call latency is 1.14 seconds for Short DSL, 3.53 for JSON, and 3.27 for free-form; the shared executor takes approximately 22.6 seconds.

6.5 Specialized Commitment Backends

Composition reference. On 60 unseen binding cross-products, an enumerated case table reaches 0%, while the Explicit-Factor Composer and Direct Short DSL each reach 100%, confirming factor-wise construction beyond memorized complete cases.

Matched model backends. Direct Short DSL predicts complete role-local paths; Factor-Code + Rule predicts stage-specific binding codes and delegates path materialization to the deterministic composer. Both complete 128/128 semantic clusters (768/768 seed-role views) with perfect parsing, commitment validity, and verified handoff. On the separate 40-cluster same-card subset, Factor-Code + Rule uses slightly longer input context but reduces mean output by 75.5% and TTC p50 from 2.453 to 0.824 seconds. Both transmit the same 76.6-byte peer request. Table 3 reports the matched profile.

The byte-identical peer payload separates the coordination interface from its model-side realization: the latency gain comes from

Table 2: Quality-matched communication surfaces. Quality and episode time use 128 held-out clusters; online cost uses the 40-cluster same-card subset. Wire counts the peer request for structured methods and all inter-agent dialogue for free-form.

Method	Success ↑	Calls ↓	Input tok. ↓	Output tok. ↓	Wire (B) ↓	TTC p50/p95 (s) ↓	Episode p50/p95 (s) ↓
GenCoord–DSL	128/128	**2.00**	**949.1**	**78.6**	**76.3**	**2.27 / 2.62**	**24.86 / 27.06**
GenCoord–JSON	128/128	**2.00**	1154.7	267.6	303.4	7.07 / 7.86	29.94 / 31.82
Controlled free-form	128/128	2.50	1585.2	265.8	1061.1	7.13 / 7.86	30.09 / 31.92

Table 3: Matched model backends for the same peer-facing commitment interface. Quality uses 128 clusters; cost uses a separate 40-cluster same-card subset.

Metric	Direct Short DSL	Factor-Code + Rule
Success ↑	**128/128**	**128/128**
Input tok. ↓	**948.9**	980.2
Output tok. ↓	79.1	**19.4**
Wire (B) ↓	**76.6**	**76.6**
TTC p50/p95 (s) ↓	2.453 / 2.723	**0.824 / 0.997**

shortening the autoregressive target while preserving the same validated commitment.

6.6 Task-Structure Visibility

Across 96 diagnostic clusters, Full DAG and Skeleton views each preserve the familiar post-handoff motif at 100%; Skeleton removes instance-specific predecessors and edges while retaining coarse stages. Node-only additionally removes stage cues and changes node arrangement, reducing familiar-motif success to 18.2%. Both unseen transform-order motifs remain at 0% under all views. Coarse order therefore supports familiar path composition without the complete instance DAG, while new transformation orders remain the structural boundary; the full matrix appears in the supplement.

7 Discussion

Task consequence as the coordination primitive. The two intervention families expose the same content-to-route causal link in opposite directions: replacing `REQ` redirects receiver execution, while counterfactual feedback changes requester actor, handoff object, and suffix. The zero-call rule reproduces the content effect, identifying the binding as the cross-boundary control variable in the current task families; GenCoord couples it to the executable paths and discharge conditions that realize the joint route.

Commitment horizon as online control. A skill-path commitment fixes both who acts and how far the route proceeds before another model decision. `SELF+REQ` exposes the sender prefix, handoff boundary, peer suffix, typed arguments, and dependencies. Multi-step commitments remove an intermediate deliberation stage and improve template-shift success despite the single-step controller's additional supervision, linking horizon directly to online replanning.

Stable semantics, optimized realization. Tables 2 and 3 expose two orthogonal layers. Surface compression replaces JSON or dialogue with Short DSL; formation compression replaces full-path decoding with a stage code and deterministic composition. Rule, factor, and direct backends all terminate at the same peer-facing interface. At that boundary, the typed checker produces a validated commitment, materialization instantiates the dispatch plan, and world predicates verify discharge. Explicit task fields therefore support causal intervention, modular backend replacement, and failure localization without changing coordination semantics.

Shared structure and information interfaces. Coarse stages preserve a familiar handoff motif after instance-specific edges are removed, whereas new transform orders remain unresolved. Let $b(z)$ be serialized byte length, u_i^t uploaded local state, d_i^t dispatched decision, E_t active coordination edges, γ_{ij}^t an edge exchange, and R_c, R_l the respective round counts:

$$B_{\text{central}} = \sum_{t=1}^{R_c} \sum_{i=1}^{n} [b(u_i^t) + b(d_i^t)],$$

$$B_{\text{local}} = \sum_{t=1}^{R_l} \sum_{(i,j)\in E_t} b(\gamma_{ij}^t).$$

Centralized traffic follows synchronized state volume, participating agents, and rounds; local traffic follows active edges, commitment size, and local rounds. Both interfaces resolve all 40 Goal × Capability clusters across three seeds, while exposing different information flows.

Limitations. The evaluation covers bilateral coordination over a shared executable prior in static Minecraft with reliable messaging. Dynamic multi-edge consistency and open-ended task decomposition remain outside the current evidence.

8 Conclusion

Joint skill paths become ambiguous when their determining facts are distributed across agents. GenCoord carries each fact's executable consequence in a `SELF+REQ` commitment: requests transmit sender-local bindings, while bounded feedback returns peer-local capability consequences. Counterfactual interventions make receiver execution and requester revision follow the delivered task consequence in their respective directions. Multi-step commitments reduce online decisions and improve held-out-template success. At matched closed-loop quality, Short DSL lowers every reported online cost relative to controlled free-form communication, while a factor-coded backend accelerates the same peer-facing interface. Skill-path commitments thus connect local generative reasoning to verified multi-agent action.

References

[1] Cristian-Paul Bara, Sky CH-Wang, and Joyce Chai. 2021. MindCraft: Theory of Mind Modeling for Situated Dialogue in Collaborative Tasks. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 1112–1125. doi:10.18653/v1/2021.emnlp-main.85

[2] Qi Chai, Zhang Zheng, Junlong Ren, Deheng Ye, Zichuan Lin, and Hao Wang. 2025. CausalMACE: Causality Empowered Multi-Agents in Minecraft Cooperative Tasks. In *Findings of the Association for Computational Linguistics: EMNLP 2025*. Association for Computational Linguistics, 14410–14426. doi:10.18653/v1/2025.findings-emnlp.777

[3] Weize Chen, Chenfei Yuan, Jiarui Yuan, Yusheng Su, Chen Qian, Cheng Yang, Ruobing Xie, Zhiyuan Liu, and Maosong Sun. 2024. Beyond Natural Language: LLMs Leveraging Alternative Formats for Enhanced Reasoning and Communication. In *Findings of the Association for Computational Linguistics: EMNLP 2024*. Association for Computational Linguistics, 10626–10641. doi:10.18653/v1/2024.findings-emnlp.623

[4] Weize Chen, Jiarui Yuan, Chen Qian, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2024. Optima: Optimizing Effectiveness and Efficiency for LLM-Based Multi-Agent System. *arXiv preprint arXiv:2410.08115* (2024).

[5] Han-Lim Choi, Luc Brunet, and Jonathan P. How. 2009. Consensus-Based Decentralized Auctions for Robust Task Allocation. *IEEE Transactions on Robotics* 25, 4 (2009), 912–926. doi:10.1109/TRO.2009.2022423

[6] Amit K. Chopra, Matteo Baldoni, Samuel H. Christie V, and Munindar P. Singh. 2025. Azorus: Commitments over Protocols for BDI Agents. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 490–499. https://www.ifaamas.org/Proceedings/aamas2025/pdfs/p490.pdf

[7] Philip R. Cohen and Hector J. Levesque. 1991. Teamwork. *Noûs* 25, 4 (1991), 487–512. doi:10.2307/2216075

[8] Yubo Dong, Xukun Zhu, Zhengzhe Pan, Linchao Zhu, and Yi Yang. 2024. VillagerAgent: A Graph-Based Multi-Agent Framework for Coordinating Complex Task Dependencies in Minecraft. In *Findings of the Association for Computational Linguistics: ACL 2024*. Association for Computational Linguistics, 16290–16314. doi:10.18653/v1/2024.findings-acl.964

[9] Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. 2016. Learning to Communicate with Deep Multi-Agent Reinforcement Learning. In *Advances in Neural Information Processing Systems*, Vol. 29.

[10] Brian P. Gerkey and Maja J. Matarić. 2004. A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems. *The International Journal of Robotics Research* 23, 9 (2004), 939–954. doi:10.1177/0278364904045564

[11] Ran Gong, Qiuyuan Huang, Xiaojian Ma, Yusuke Noda, Zane Durante, Zilong Zheng, Demetri Terzopoulos, Li Fei-Fei, Jianfeng Gao, and Hoi Vo. 2024. MindAgent: Emergent Gaming Interaction. In *Findings of the Association for Computational Linguistics: NAACL 2024*. Association for Computational Linguistics, 3154–3183. doi:10.18653/v1/2024.findings-naacl.200

[12] Barbara J. Grosz and Sarit Kraus. 1996. Collaborative Plans for Complex Group Action. *Artificial Intelligence* 86, 2 (1996), 269–357. doi:10.1016/0004-3702(95)00103-4

[13] Shashwat Gupta, Anson Bastos, Mayukh Das, Supriyo Ghosh, Nagarajan Natarajan, Chetan Bansal, and Saravan Rajmohan. 2026. Learning Optimal Message Representations for Agentic Communication. In *Findings of the Association for Computational Linguistics: ACL 2026*. Association for Computational Linguistics, 28849–28879. doi:10.18653/v1/2026.findings-acl.1441

[14] HuaDong Jian, Chenghao Li, Haoyu Wang, Jiajia Shuai, Jinyu Guo, Yang Yang, and Chaoning Zhang. 2026. Gated Coordination for Efficient Multi-Agent Collaboration in Minecraft Game. *arXiv preprint arXiv:2604.18975* (2026). https://arxiv.org/abs/2604.18975

[15] Hojoon Kim, Yuheng Wu, and Thierry Tambe. 2026. AgenticCache: Cache-Driven Asynchronous Planning for Embodied AI Agents. In *Proceedings of Machine Learning and Systems*, Vol. 8.

[16] Qian Long, Zhi Li, Ran Gong, Ying Nian Wu, Demetri Terzopoulos, and Xiaofeng Gao. 2024. TeamCraft: A Benchmark for Multi-Modal Multi-Agent Systems in Minecraft. *arXiv preprint arXiv:2412.05255* (2024). https://arxiv.org/abs/2412.05255

[17] Zhao Mandi, Shreeya Jain, and Shuran Song. 2024. RoCo: Dialectic Multi-Robot Collaboration with Large Language Models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 286–299. doi:10.1109/ICRA57147.2024.10610855

[18] PrismarineJS. 2026. Mineflayer: Create Minecraft Bots with a High-Level JavaScript API. Software repository. Version 4.37.1 used in the project; accessed 2026-08-08. https://github.com/PrismarineJS/mineflayer

[19] Qwen Team. 2026. Qwen3.5-0.8B. Hugging Face model card. Accessed 2026-08-08. https://huggingface.co/Qwen/Qwen3.5-0.8B

[20] Reid G. Smith. 1980. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Trans. Comput.* C-29, 12 (1980), 1104–1113. doi:10.1109/TC.1980.1675516

[21] Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. 2016. Learning Multiagent Communication with Backpropagation. In *Advances in Neural Information Processing Systems*, Vol. 29.

[22] Milind Tambe. 1997. Towards Flexible Teamwork. *Journal of Artificial Intelligence Research* 7 (1997), 83–124. doi:10.1613/jair.433

[23] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An Open-Ended Embodied Agent with Large Language Models. *arXiv preprint arXiv:2305.16291* (2023). https://arxiv.org/abs/2305.16291

[24] Rundong Wang, Xu He, Runsheng Yu, Wei Qiu, Bo An, and Zinovi Rabinovich. 2020. Learning Efficient Multi-agent Communication: An Information Bottleneck Approach. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 9908–9918.

[25] Isadora White, Kolby Nottingham, Ayush Maniar, Max Robinson, Hansen Lillemark, Mehul Maheshwari, Lianhui Qin, and Prithviraj Ammanabrolu. 2025. Collaborating Action by Action: A Multi-agent LLM Framework for Embodied Reasoning. *arXiv preprint arXiv:2504.17950* (2025). https://arxiv.org/abs/2504.17950

[26] Juheon Yi, Jinglu Wang, Xiaoyi Zhang, and Yan Lu. 2026. Multi-agent Framework for Time-Sensitive Complementary Collaboration in Minecraft. *arXiv preprint arXiv:2606.15684* (2026). https://arxiv.org/abs/2606.15684

Supplementary Material

GenCoord: Skill-Path Commitments under Private Information

Reader map. Protocol definitions and exact resolution appear in Sections S1 and S4; causal feedback evidence appears in Section S10; the grounded trace and cost results appear in Sections S4 and S5; backend formation and naturalized private-fact results appear in Section S8; reproducibility materials appear in Section S11.

S1 Protocol Lifecycles and Stage Attribution

Takeaway. Forward request resolution and bounded feedback resolution use the same executable commitment object while carrying task-relevant consequences of local information in opposite directions.

Figure S1 gives the protocol-level reading order for the two routes. Table S1 aligns each route with its learned stages and transparent operations, while Table S2 fixes the symbols used by Algorithms S1–S2.

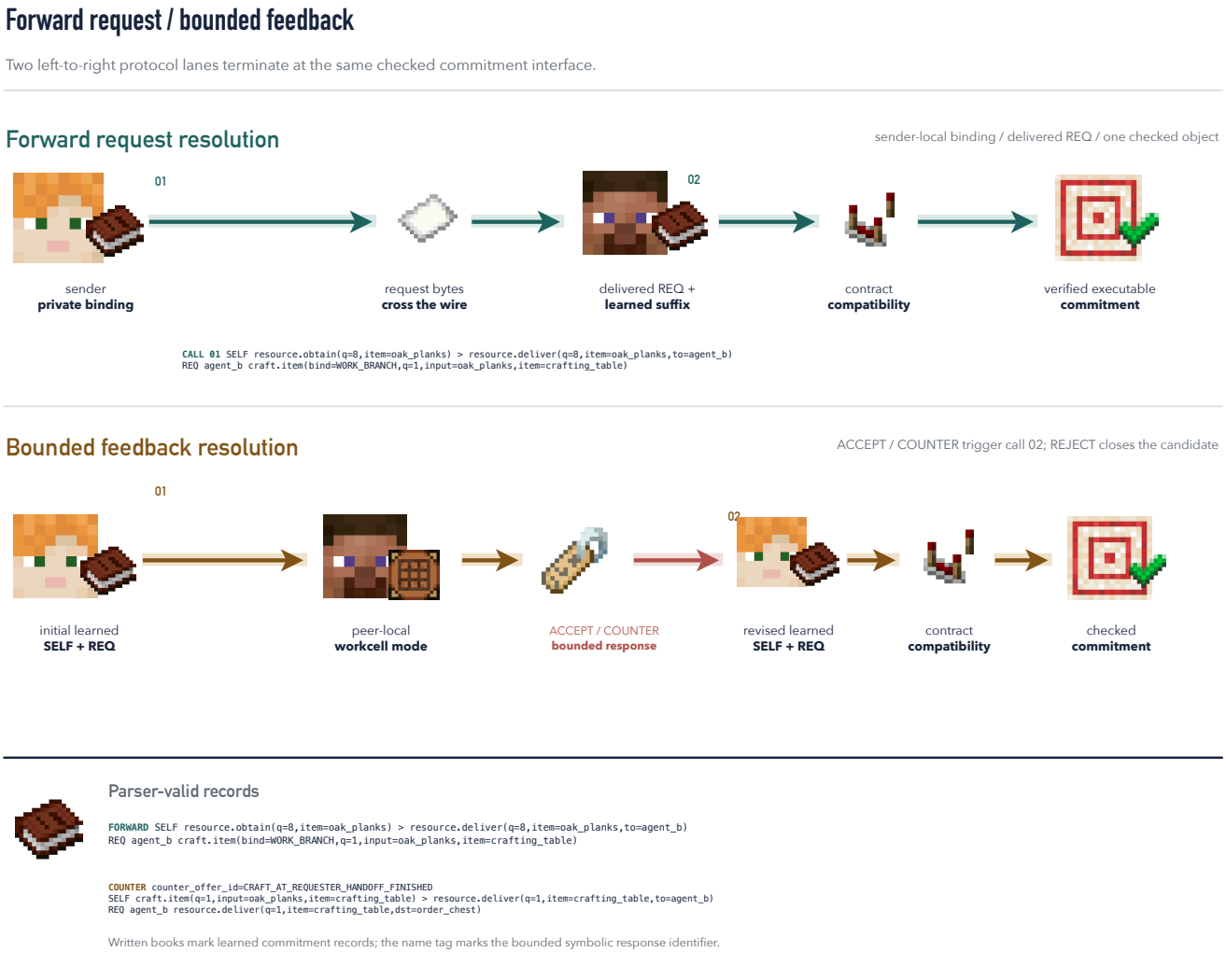

Figure S1. Forward and bounded-feedback lifecycles ending in the same contract-validated commitment interface.

The protocol result type is the tagged union

$$\mathcal{R} = (\{\texttt{SUCCESS}\} \times \mathcal{C} \times \mathcal{D}) \uplus (\{\texttt{NO_COMMITMENT}\} \times \mathcal{E}) \uplus (\{\texttt{FAIL}\} \times \mathcal{F}).$$

We write its values as $\mathrm{Success}(C, D)$, $\mathrm{NoCommitment}(e)$, and $\mathrm{Fail}(f)$.

Table S1. Lifecycle stages and learned-call attribution. Goal–Capability uses two learned calls on the evaluated Accept/Counter branches; Reject terminates after the first call. For the factor backend, `active_binding_id` is emitted at `sender_initial` and `accepted_binding_id` at `receiver_after_request`.

Setting	First learned output / stage	Boundary object	Second learned output / stage	Deterministic / non-learned operation	Learned calls
Main forward route	`sender_initial`	delivered `REQ`	`receiver_after_request`	canonical joint resolution	2
Goal–Capability	`requester_initial`	bounded capability response	`requester_after_response`	`ACCEPT`/`COUNTER` policy	2
Single-step route	`sender_initial`	request and state transition	`sender_after_obtain`, then receiver	task transition	episode-dependent
Factor backend	`active_binding_id` (sender)	delivered `REQ`	`accepted_binding_id` (receiver)	deterministic composer	2

Table S2. Core notation used by Algorithms S1–S2.

Symbol	Meaning and domain
$\mathcal{A} = \{a_1, \ldots, a_n\}$; $i, j \in \{1, \ldots, n\}$	agent team and indices; protocol roles are written as *snd*, *rcv*, *req*, and *peer*
$\mathcal{K} = (\mathcal{S}, G, \Theta, \Gamma)$	reusable executable prior: skill catalog $\mathcal{S}$, public task DAG G, typed schema Θ, and grounding/checker contract Γ
ξ_{pub}, ω_t, x_i^t	public episode instantiation, current executable world state, and agent i's model-visible local context at round t; ω_t is visible only to the runtime verifier
$\hat{y}_{\text{snd}}, q_{\text{snd}}$; $\hat{y}_{\text{rcv}}, q_{\text{rcv}}$	raw model strings and parsed sender/receiver proposals; $q = \perp$ denotes parse failure
$r_{\text{snd}\to\text{rcv}}$, u_{rcv}	requested peer path and receiver-local realization suffix
c_{peer}, ρ, α	peer capability, bounded response, and selected `counter_offer_id`
$N_{\mathcal{K}}^{\text{path}}$, $N_{\mathcal{K}}^{\text{prop}}$, Actorize	canonical normalization for paths/proposals and explicit instantiation of an implicit task actor
$\mathcal{C}$, $\mathcal{D}$, $\mathcal{E}$, $\mathcal{F}$	spaces of resolved commitments, dispatch plans, no-commitment reasons, and protocol/runtime failure codes

Algorithm S1 Forward resolution

Require: sender context x_{snd}, receiver context x_{rcv}, prior $\mathcal{K}$, public episode instantiation ξ_{pub}, executable world ω_t

$\hat{y}_{\text{snd}} \leftarrow \textsc{Generate}(x_{\text{snd}}, \mathcal{K}, \xi_{\text{pub}})$ ▷ `SELF+REQ`
$q_{\text{snd}} \leftarrow \textsc{Parse}(\hat{y}_{\text{snd}})$
if $q_{\text{snd}} = \perp$ **then**
  return $\textsc{Fail}$(`PARSE_FAILURE`)
if $\neg\textsc{ContractCheck}(q_{\text{snd}}, x_{\text{snd}}, \mathcal{K}, \xi_{\text{pub}})$ **then**
  return $\textsc{Fail}$(`CONTRACT_REJECT`)
if $|q_{\text{snd}}.\texttt{peer_requests}| \neq 1$ **or** $q_{\text{snd}}.\texttt{peer_requests}[0].\texttt{requested_plan} = []$ **then**
  return $\textsc{Fail}$(`CONTRACT_REJECT`)
$r_{\text{snd}\to\text{rcv}} \leftarrow q_{\text{snd}}.\texttt{peer_requests}[0]$; deliver $r_{\text{snd}\to\text{rcv}}$
$\hat{y}_{\text{rcv}} \leftarrow \textsc{Generate}(x_{\text{rcv}}, \mathcal{K}, \xi_{\text{pub}}, r_{\text{snd}\to\text{rcv}})$
$q_{\text{rcv}} \leftarrow \textsc{Parse}(\hat{y}_{\text{rcv}})$
if $q_{\text{rcv}} = \perp$ **then**
  return $\textsc{Fail}$(`PARSE_FAILURE`)
if $\neg\textsc{ContractCheck}(q_{\text{rcv}}, x_{\text{rcv}}, \mathcal{K}, \xi_{\text{pub}})$ **or** $q_{\text{rcv}}.\texttt{self_plan} = []$ **or** $q_{\text{rcv}}.\texttt{peer_requests} \neq []$ **then**
  return $\textsc{Fail}$(`CONTRACT_REJECT`)
$C \leftarrow \textsc{Resolve}_{\to}(q_{\text{snd}}, q_{\text{rcv}}, \mathcal{K}, \xi_{\text{pub}})$
if $C = \perp$ **then**
  return $\textsc{Fail}$(`RESOLUTION_CONFLICT`)
$D \leftarrow \textsc{Materialize}(C, \mathcal{K}, \xi_{\text{pub}})$
if $D = \perp$ **then**
  return $\textsc{Fail}$(`MATERIALIZATION_FAILURE`)
`status` $\leftarrow \textsc{ExecuteAndVerify}(D, \omega_t)$
if `status` $\neq$ `SUCCESS` **then**
  return $\textsc{Fail}$(`status`)
return $\textsc{Success}(C, D)$

Algorithm S2 Bounded feedback resolution

Require: requester context x_{req}, peer capability c_{peer}, prior $\mathcal{K}$, public episode instantiation ξ_{pub}, executable world ω_t

$\hat{y}_0 \leftarrow \textsc{Generate}(x_{\text{req}}, \mathcal{K}, \xi_{\text{pub}})$
$q_0 \leftarrow \textsc{Parse}(\hat{y}_0)$
if $q_0 = \bot$ **then**
  return $\textsc{Fail}$(`PARSE_FAILURE`)
if $\neg\textsc{ContractCheck}(q_0, x_{\text{req}}, \mathcal{K}, \xi_{\text{pub}})$ **then**
  return $\textsc{Fail}$(`CONTRACT_REJECT`)
if $|q_0.\texttt{peer_requests}| \neq 1$ **or** $q_0.\texttt{peer_requests}[0].\texttt{requested_plan} = []$ **then**
  return $\textsc{Fail}$(`CONTRACT_REJECT`)
$r_{\text{req}\to\text{peer}} \leftarrow q_0.\texttt{peer_requests}[0]$
$\rho \leftarrow \textsc{ResponsePolicy}(r_{\text{req}\to\text{peer}}, c_{\text{peer}}, \mathcal{K}, \xi_{\text{pub}})$
if $\neg\textsc{ResponseValid}(\rho, r_{\text{req}\to\text{peer}}, \mathcal{K}, \xi_{\text{pub}})$ **then**
  return $\textsc{Fail}$(`CONTRACT_REJECT`)
if $\rho = \textsc{Reject}$ **then**
  return $\textsc{NoCommitment}(\rho.\texttt{reason_code})$
$\hat{y}_1 \leftarrow \textsc{Generate}(x_{\text{req}}, \mathcal{K}, \xi_{\text{pub}}, q_0, \rho)$ ▷ ACCEPT or COUNTER
$q_1 \leftarrow \textsc{Parse}(\hat{y}_1)$
if $q_1 = \bot$ **then**
  return $\textsc{Fail}$(`PARSE_FAILURE`)
if $\neg\textsc{ContractCheck}(q_1, x_{\text{req}}, \mathcal{K}, \xi_{\text{pub}})$ **then**
  return $\textsc{Fail}$(`CONTRACT_REJECT`)
if $|q_1.\texttt{peer_requests}| \neq 1$ **or** $q_1.\texttt{peer_requests}[0].\texttt{requested_plan} = []$ **then**
  return $\textsc{Fail}$(`CONTRACT_REJECT`)
if $q_1.\texttt{peer_requests}[0].\texttt{target} \neq r_{\text{req}\to\text{peer}}.\texttt{target}$ **then**
  return $\textsc{Fail}$(`CONTRACT_REJECT`)
$C \leftarrow \textsc{Resolve}_{\leftarrow}(q_0, q_1, \rho, \mathcal{K}, \xi_{\text{pub}})$
if $C = \bot$ **then**
  return $\textsc{Fail}$(`RESOLUTION_CONFLICT`)
$D \leftarrow \textsc{Materialize}(C, \mathcal{K}, \xi_{\text{pub}})$
if $D = \bot$ **then**
  return $\textsc{Fail}$(`MATERIALIZATION_FAILURE`)
`status` $\leftarrow \textsc{ExecuteAndVerify}(D, \omega_t)$
if `status` $\neq$ `SUCCESS` **then**
  return $\textsc{Fail}$(`status`)
return $\textsc{Success}(C, D)$

The response policy selects among admissible alternatives from the grounded skill hierarchy. A response is $\rho \in \{\textsc{Accept}, \textsc{Reject}, \textsc{Counter}(\alpha)\}$. The bounded alternative α is serialized in the E1/E2 artifact as `counter_offer_id`; its observed counter value is `CRAFT_AT_REQUESTER_HANDOFF_FINISHED`. `ACCEPT` retains the proposed branch without an alternative, `COUNTER` names exactly one admissible alternative, and `REJECT` closes the candidate without a second generation or dispatch. Accepted and countered proposals both condition a second requester call. The transparent policy contributes zero learned calls and exposes the bounded capability consequence explicitly; Table S3 gives the complete response contract. The world state ω_t remains outside the model and response-policy inputs and is accessed by execution and verification.

Algorithms S1–S2 keep parsing, contract validation, resolution, materialization, and world discharge distinct. The terminal verifier returns one of `SUCCESS`, `EXECUTION_FAILURE`, `HANDOFF_FAILURE`, or `TERMINAL_FAILURE`; earlier stages return the failure codes shown in the algorithms.

S1.1 Resolver semantics

As in the main paper, a grounded task is $\tau = (a, k, \mathbf{v}, \ell, P)$: actor a, skill identifier k, typed arguments $\mathbf{v}$, grounding location ℓ, and predecessor set P. A resolved commitment has the common operational form

$$C = (S_i, S_j, b, P_C) \in \mathcal{C},$$

where S_i and S_j are the two normalized role-local paths, b is the selected binding identifier from the public codebook, and P_C is the merged predecessor relation. Protocol-specific proposals, responses, and realization records remain in the trace; C contains only the operative obligations that enter materialization.

The function Actorize(z, a) instantiates every implicit actor in path z as a. We use $N_{\mathcal{K}}^{\text{path}}$ for canonical path normalization and $N_{\mathcal{K}}^{\text{prop}}$ for proposal normalization; both canonicalize aliases, typed defaults, grounded references, bindings, actors, and predecessor semantics.

Table S3. Finite response schema and stage-local failure codes.

Decision / stage	Valid fields or condition	Protocol consequence
Accept	reason code; no `counter_offer_id`	second requester call preserves the normalized initial branch
Counter(α)	reason code and exactly one admissible `counter_offer_id`=α	second requester call realizes the selected bounded branch
Reject	reason code; no counter alternative	candidate closes; no second generation, commitment, or dispatch
Invalid response	unknown decision, forbidden field, missing alternative, or inadmissible ID	`CONTRACT_REJECT`
Parse	raw string cannot be decoded	`PARSE_FAILURE`
Contract	schema, role, skill, argument, target, or response check fails	`CONTRACT_REJECT`
Resolution	realization, boundary fields, branch, or dependency merge conflicts	`RESOLUTION_CONFLICT`
Materialization	validated commitment cannot instantiate a dispatch	`MATERIALIZATION_FAILURE`
World discharge	runtime action, inventory handoff, or terminal predicate fails	`EXECUTION_FAILURE`, `HANDOFF_FAILURE`, or `TERMINAL_FAILURE`

Forward resolution. Define

$$S_{\mathrm{snd}} = N_{\mathcal{K}}^{\mathrm{path}}(\mathrm{Actorize}(q_{\mathrm{snd}}.\texttt{self_plan}, \mathrm{snd}))\,,$$

$$R_{\mathrm{rcv}} = N_{\mathcal{K}}^{\mathrm{path}}(\mathrm{Actorize}(r_{\mathrm{snd}\to\mathrm{rcv}}.\texttt{requested_plan}, \mathrm{rcv}))\,, \quad U_{\mathrm{rcv}} = N_{\mathcal{K}}^{\mathrm{path}}(\mathrm{Actorize}(q_{\mathrm{rcv}}.\texttt{self_plan}, \mathrm{rcv}))\,.$$

Within the evaluated canonical contract, the receiver realizes the request exactly when

$$\mathrm{Realizes}_{\mathcal{K}}(U_{\mathrm{rcv}}, R_{\mathrm{rcv}}) \iff U_{\mathrm{rcv}} = R_{\mathrm{rcv}}. \tag{S1}$$

This equality preserves ordered task count, skills, typed values, bindings, actors, locations, and dependency semantics after normalization. Textual argument order and accepted aliases may differ. Canonical realization contains exactly the requested model-owned tasks; deterministic executor bookkeeping is added after resolution.

Let $\mathrm{HandoffCompatible}_{\mathcal{K},\xi_{\mathrm{pub}}}(S_i, S_j)$ denote compatibility of the producing/receiving actors, item, quantity, source/destination references, and boundary dependency under the public episode instantiation. Resolve$_{\to}$ returns

$$C^{\to} = (S_{\mathrm{snd}}, U_{\mathrm{rcv}}, b, P_C)$$

only when (i) the receiver identity equals the request target; (ii) q_{rcv} has a nonempty `SELF` suffix and no outbound request; (iii) Equation S1 holds; (iv) $\mathrm{HandoffCompatible}_{\mathcal{K},\xi_{\mathrm{pub}}}(S_{\mathrm{snd}}, U_{\mathrm{rcv}})$ holds; and (v) b yields a complete acyclic predecessor merge P_C. The merge preserves each role-local order, retains public predecessors on the selected branch, and adds a cross-agent dependency from the contract-validated handoff producer to the first consuming receiver node. The handoff becomes *verified* only after execution. Missing fields, incompatible handoffs, multiple selected branches, or cycles yield $\perp$.

Feedback resolution. Let $\mathcal{B}_{\mathcal{K}}$ be the public branch set, $B : \mathcal{Q}_{\mathrm{legal}} \to \mathcal{B}_{\mathcal{K}} \cup \{\perp\}$ extract the normalized branch selected by a proposal, and $\mathrm{AltBranch}_{\mathcal{K},\xi_{\mathrm{pub}}} : \mathcal{I}_{\mathrm{alt}} \to \mathcal{B}_{\mathcal{K}} \cup \{\perp\}$ map an admissible alternative identifier to its unique branch. The response-conditioned branch is

$$B^{\mathrm{resp}}_{\mathcal{K},\xi_{\mathrm{pub}}}(\rho, q_0) = \begin{cases} B(q_0), & \rho = \textsc{Accept}, \\ \mathrm{AltBranch}_{\mathcal{K},\xi_{\mathrm{pub}}}(\alpha), & \rho = \textsc{Counter}(\alpha), \\ \perp, & \rho = \textsc{Reject}. \end{cases}$$

For Accept, the second record may use accepted surface variants while preserving the normalized operative branch, target peer, and requester goal fields of q_0. For Counter(α), q_1 preserves the requester goal fields and target peer, replaces the initial branch with the selected alternative, and realizes that branch in its operative peer request. Reject closes the candidate after the first call.

For accepted and countered proposals, define

$$S_{\mathrm{req}} = N_{\mathcal{K}}^{\mathrm{path}}(\mathrm{Actorize}(q_1.\texttt{self_plan}, \mathrm{req}))\,,$$

$$R_{\text{peer}} = N_{\mathcal{K}}^{\text{path}}(\text{Actorize}(q_1.\texttt{peer_requests}[0].\texttt{requested_plan}, \text{peer}))\,.$$

The response itself is the peer-side resolution record, and the route proceeds directly to requester revision. Resolve$_{\leftarrow}$ requires

$$B(q_1) = B^{\text{resp}}_{\mathcal{K},\xi_{\text{pub}}}(\rho, q_0),$$

HandoffCompatible$_{\mathcal{K},\xi_{\text{pub}}}(S_{\text{req}}, R_{\text{peer}})$, and a complete acyclic merge P_C, then returns

$$C^{\leftarrow} = (S_{\text{req}}, R_{\text{peer}}, b, P_C).$$

The protocol trace retains q_0 and ρ as provenance, while $C^{\leftarrow}$ contains only the resolved operative paths. The materializer constructs

$$D = \text{Materialize}(C, \mathcal{K}, \xi_{\text{pub}}).$$

It preserves the complete high-level obligation set while instantiating validated actors, locations, task-instance identifiers, dependencies, and checker attachments. The compiler then adds deterministic navigation, inventory-transfer mechanics, timeouts, claims, and runtime metadata required to realize D as Mineflayer calls.

S2 Task Suite and Programmatic Supervision

Takeaway. Four forward-request task families place the path-determining binding in the sender's local view; the delivered request disambiguates the receiver's two admissible branches.

Table S4 enumerates the eight templates and the commitment field changed by each balanced binding. Table S5 then shows how every template produces reciprocal sender-proposal and receiver-realization supervision.

Table S4. Complete inventory of the eight core task templates and the commitment field varied by each binding.

Family	Template	Source material	Balanced bindings	Binding-dependent request path	Changed field
Destination	Build Site A/B	oak planks	`SITE_A`/`SITE_B`	`build.component`	destination
Destination	Chest A/B	cobblestone	`CHEST_A`/`CHEST_B`	`resource.deliver`	destination
Recipe	Chest/Crafting Table	oak planks	`CHEST`/ `CRAFTING_TABLE`	`craft.item`	transform; handoff item
Recipe	Planks/Sticks	oak planks	`PLANKS`/ `STICKS`	`craft.item`/ `resource.deliver`	transform; handoff item
Allocation	Deposit/Build	cobblestone	`BUILD`/`DEPOSIT`	`build.component`/ `resource.deliver`	actor and remaining work
Allocation	Dual Build	oak planks	`ROOF`/`WALL`	`build.component`	actor and component
Active branch	Active Order	oak planks	`STORAGE`/`WORK`	`craft.item`	downstream continuation
Active branch	Remaining Terminal	oak planks	`DEPOT`/ `MARKER`	`build.component`/ `resource.deliver`	downstream continuation

Each template has two equally represented admissible bindings. The active binding appears in the sender-local private field and is absent from the receiver-local view. Paired binding worlds keep the receiver's decision context matched until the request arrives, so removing the delivered request leaves the receiver unable to distinguish the two executable branches and produces the constructive 50% ceiling. Every instance requires a verified material handoff and a binding-specific terminal world-state predicate.

Table S5. Programmatic supervision records for the forward route.

Record stage	Target object	Rows
Sender proposal	multi-step `SELF` plus peer `REQ`	960
Receiver after request	downstream `SELF`; empty `REQ`	960
Intermediate sender (single-step)	next local action after obtain	960

Targets are instantiated from task-template specifications, sampled local facts, and canonical task objects. Public Minecraft resources support ontology and scenario design; the reciprocal targets come from the executable template contract. The data builder writes the model-visible context and target separately, then validates that each target maps back to the intended binding and terminal predicate.

S3 Training and Checkpoint Provenance

Takeaway. The checkpoint inventory makes stage coverage and optimization budget explicit. Multi-step training uses 1,920 rows and 240 updates per seed, while single-step receives an additional intermediate stage, 2,880 rows, and 360 updates.

Tables S6 and S7 report the matched model configuration, stage counts, update budgets, data hashes, and checkpoint identities.

Table S6. Training budgets and stage coverage per random seed.

Model or condition	Rows	Stage counts	Epochs	Steps	Seeds
Multi-step / direct / surface	1,920	960 sender + 960 receiver	2	240	3
Self-plan-only	1,920	960 sender + 960 receiver	2	240	3
Matched-SFT free-form	2,400	1,200 sender + 1,200 receiver	2	300	3
Single-step	2,880	960 sender + 960 intermediate sender + 960 receiver	2	360	3
Factor backend	1,920	960 sender code + 960 receiver code	2	240	3

Table S7. Shared training configuration and horizon-model provenance.

Field	Multi-step	Single-step
(a) Common configuration		
Base model	Qwen3.5-0.8B [Qwen Team(2026)]; tree `1ba0a4ab...`	
Optimizer	AdamW; LR 10^{-5}; weight decay 0.01; cosine schedule; warm-up 0.03; grad-norm 1.0	
Batch / precision	micro 4; accumulation 4; effective 16; bfloat16 on A100 40GB	
Length / decoding	maximum 2,048; deterministic decoding; structured cap 256 tokens	
Objective / selection	target-and-EOS causal loss; final-step checkpoint; DEV reserved for implementation checks	
(b) Model-specific data and checkpoint identity		
Training rows / updates	1,920 / 240	2,880 / 360
Training-data SHA-256 prefix	`f31ba0b5`	`8bdbf220`
Config SHA-256 prefix	`d965b3e7`	`d965b3e7`
Seed IDs	`2026073101-03`	`2026073101-03`
Final checkpoint prefixes	`66e1f8c0`, `4b99b075`, `48df1656`	`9c0bd589`, `f82d702f`, `42705cf5`

The project-side training record stores the complete data-manifest hash, configuration hash, base-model tree hash, seed, runtime versions, token exposure, checkpoint identity, and final checkpoint tree hash. The public arXiv artifact retains the scientific configuration and content hashes while removing host, scheduler, process, and private-path identifiers.

S4 Commitment Schema and Grounded Runtime

Takeaway. The model emits a compact symbolic commitment. A typed checker validates its semantics, a canonical materializer instantiates the dispatch plan, and the compiler lowers that plan to Mineflayer skills.

S4.1 Short DSL grammar

```
program ::= self_line NEWLINE req_line
self_line ::= "SELF -" | "SELF " path
```

```
req_line ::= "REQ -" | "REQ " agent " " path
path ::= task (" > " task){0,4}
task ::= skill "(" [args] ")"
args ::= arg ("," arg)*
arg ::= key "=" value
key ::= ident
value ::= json_string | json_number
          | "true" | "false" | "null" | atom
skill ::= control.wait | resource.obtain
          | resource.deliver | craft.item
          | transform.supply_input
          | transform.supply_fuel
          | transform.collect_output
          | build.component
ident ::= [A-Za-z][A-Za-z0-9_]*
agent ::= atom
atom ::= [A-Za-z0-9_.:/+-]+
json_number ::= JSON numeric literal
json_string ::= RFC 8259 double-quoted string
```

The displayed BNF defines the canonical serializer output. Value recognition follows the displayed priority: a leading double quote starts a JSON string; a complete JSON number is recognized next; the three reserved literals precede the unquoted atom fallback. A quoted string is scanned atomically, so escaped quotes, commas, parentheses, and backslashes inside it do not terminate an argument or task. Standard JSON escapes are decoded before type checking, and unquoted atoms remain restricted to the ASCII class shown above. The canonical serializer emits exactly two LF-separated lines with normalized ASCII spacing and no leading or trailing whitespace; parser-tolerated noncanonical whitespace is outside the serializer invariant.

The serializer emits the compact keys `q`, `item`, `input`, `to`, `dst`, `bind`, `site`, `from`, `dst_role`, `loc`, and `station`. The parser also accepts their expanded compatibility names, including `count`/`quantity`, `target_agent_ref`/`to_agent_id`, `destination`, `destination_role`, `from_agent_id`, `location_ref`, and `binding_id`, then canonicalizes them before checking. Syntactic parsing accepts identifier-shaped keys and primitive JSON values; the skill-specific contract checker rejects unknown keys, invalid types, unknown agents, inadmissible bindings, and missing required fields. The codec rejects duplicate keys, paths outside the evaluated catalog, invalid atoms, trailing text, and an empty requested plan.

The parser requires exactly two lines and permits one to five ordered tasks per nonempty path. Quantities are positive integers; `to` names a peer; `dst`, `site`, `loc`, and `station` identify grounding references; `bind` selects an admissible branch from the public codebook. The separator `>` encodes commitment precedence; compilation expands the validated path into low-level action calls. `REQ -` is a legal record whose parsed `peer_requests` value is the empty list; $\perp$ is reserved for parse failure, while contract rejection is represented separately by `CONTRACT_REJECT`.

Let $\mathcal{Y}_{\text{parse}}$ be syntactically admissible strings, $\mathcal{Q}_{\text{parsed}}$ parsed proposal objects, and $\mathcal{Q}_{\text{legal}}(\mathcal{K}, \xi_{\text{pub}})$ the subset satisfying the typed contract. Define

$$\text{Parse} : \mathcal{Y}_{\text{parse}} \to \mathcal{Q}_{\text{parsed}} \cup \{\perp\}, \qquad \text{ContractCheck}_{\mathcal{K},\xi_{\text{pub}}} : \mathcal{Q}_{\text{parsed}} \to \{\texttt{true}, \texttt{false}\}.$$

The legal-string domain is

$$\mathcal{Y}_{\text{legal}}(\mathcal{K}, \xi_{\text{pub}}) = \left\{ y \in \mathcal{Y}_{\text{parse}} : \text{Parse}(y) \neq \perp,\ \text{ContractCheck}_{\mathcal{K},\xi_{\text{pub}}}(\text{Parse}(y)) = \texttt{true} \right\}.$$

Let $\mathcal{Y}_{\text{canon}}(\mathcal{K})$ be canonical serializer outputs. For legal strings, the semantic decoder and encoder are

$$\text{Decode}_{\mathcal{K},\xi_{\text{pub}}}(y) = N_{\mathcal{K}}^{\text{prop}}(\text{Parse}(y)), \quad y \in \mathcal{Y}_{\text{legal}}(\mathcal{K}, \xi_{\text{pub}}),$$

$$\text{Encode}_{\mathcal{K}} : \mathcal{Q}_{\text{legal}}(\mathcal{K}, \xi_{\text{pub}}) \to \mathcal{Y}_{\text{canon}}(\mathcal{K}).$$

For legal DSL strings y_1, y_2,

$$y_1 \equiv_{\text{sem}} y_2 \iff \text{Decode}_{\mathcal{K},\xi_{\text{pub}}}(y_1) = \text{Decode}_{\mathcal{K},\xi_{\text{pub}}}(y_2). \tag{S2}$$

For $Q \in \mathcal{Q}_{\text{legal}}(\mathcal{K}, \xi_{\text{pub}})$, the tested codec invariant is

$$\text{Decode}_{\mathcal{K},\xi_{\text{pub}}}(\text{Encode}_{\mathcal{K}}(Q)) = N_{\mathcal{K}}^{\text{prop}}(Q). \tag{S3}$$

The invariant is defined and verified over the fixed schema, evaluated task families, and shared executable prior used in this study. Within that contract, the decoded object retains the ordered self path, target peer, ordered requested path, typed arguments, bindings, actors, grounding references, and dependency semantics required to determine the resolved route.

```
SELF resource.obtain(q=8,item=oak_planks) >
     resource.deliver(q=8,item=oak_planks,to=agent_a)
REQ agent_a craft.item(bind=WORK_BRANCH,q=1,
     input=oak_planks,item=crafting_table)
```

Table S8 separates the checks that establish a valid commitment from the post-materialization and post-execution evidence that discharges it.

Table S8. Checks across parsing, materialization, and live verification.

Phase	Check	Accepted condition
Pre-materialization	Syntax and identity	two-line grammar; actors and peer match the role instance
	Skill and arguments	catalog path, required keys, types, and quantities are valid
	Binding and handoff	branch is admissible; item, quantity, sender, and receiver agree
	Dependency merge	selected predecessor graph is complete and acyclic
Post-materialization	Static plan consistency	resolved actors, fields, handoff edge, and checker attachment match the validated commitment
Post-execution	World verification	handoff event and declared terminal predicate both pass

The live route is

$$\begin{aligned}\text{generate} &\to \text{parse} \to \text{contract check}\\ &\to \text{canonical materialize} \to \text{compile}\\ &\to \text{execute} \to \text{verify}.\end{aligned}$$

Once the model-generated semantics equal the selected canonical task object, the materializer combines the selected binding with the shared task structure. Mineflayer 4.37.1 [PrismarineJS(2026)] then executes the compiled skills against the official Minecraft Java server 1.21.4.

Figure S2 follows one retained episode from learned records to terminal world state; Table S9 reports the exact calls, actions, inventory deltas, and wall-clock time for the same trace.

Table S9. Representative live episode accounting.

Event or state	Observed value
Selected binding	`WORK_BRANCH`
Learned model calls	2
Executor actions	13
Sender oak planks, before → after	8 → 0
Receiver oak planks, before → after	0 → 8
Terminal crafting tables	1
Wall-clock episode time	22.36 s
Handoff / terminal verification	pass / pass

Figure S2 uses role instance `GCP-PAPER-REP-ACTIVE-ORDER-BRANCH-0015-R1`. The trace records inventory snapshots around the handoff, all issued skills, the terminal inventory, and the server-side checker result.

S5 Full Quality and Communication Results

Takeaway. At matched terminal quality, the three communication surfaces differ sharply in generated length, message size, and time-to-commitment while sharing the same executor.

Table S10 establishes the quality lock and reports the exact online costs. Figure S3 makes the resulting cost profile visually comparable, and Table S11 separates same-card call latency from quality-pool reliability.

The same-card timing pool contains 40 clusters, two role views, and three checkpoints per surface. Models run sequentially on the same device. TTC ends when the executable bilateral commitment is available; executor time begins after commitment and remains near 22.6 s p50 across surfaces.

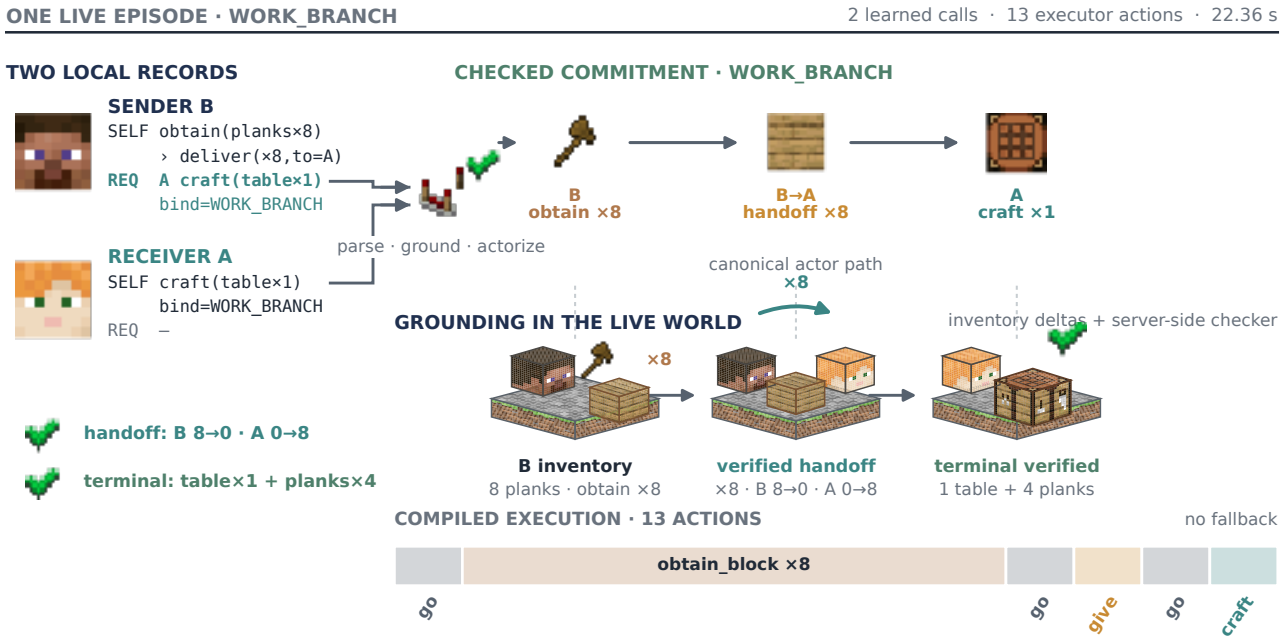

Figure S2. One grounded episode from bilateral Short DSL records to a checked commitment, verified handoff, and terminal world state.

Table S10. Matched-quality representation results. Success uses the 128-cluster quality pool; calls, token counts, protocol-specific Agent-to-Agent payload bytes, and TTC use the separate 40-cluster same-card timing pool.

Method	Success	Calls	Input tok.	Output tok.	Wire B	TTC p50	TTC p95
Short DSL GenCoord (0.8B)	100.0%	2.0	949.1	78.6	76.3	2.266 s	2.616 s
JSON GenCoord (0.8B)	100.0%	2.0	1,154.7	267.6	303.4	7.067 s	7.864 s
Controlled free-form + structured commit (0.8B)	100.0%	2.5	1,585.2	265.8	1,061.1	7.132 s	7.857 s

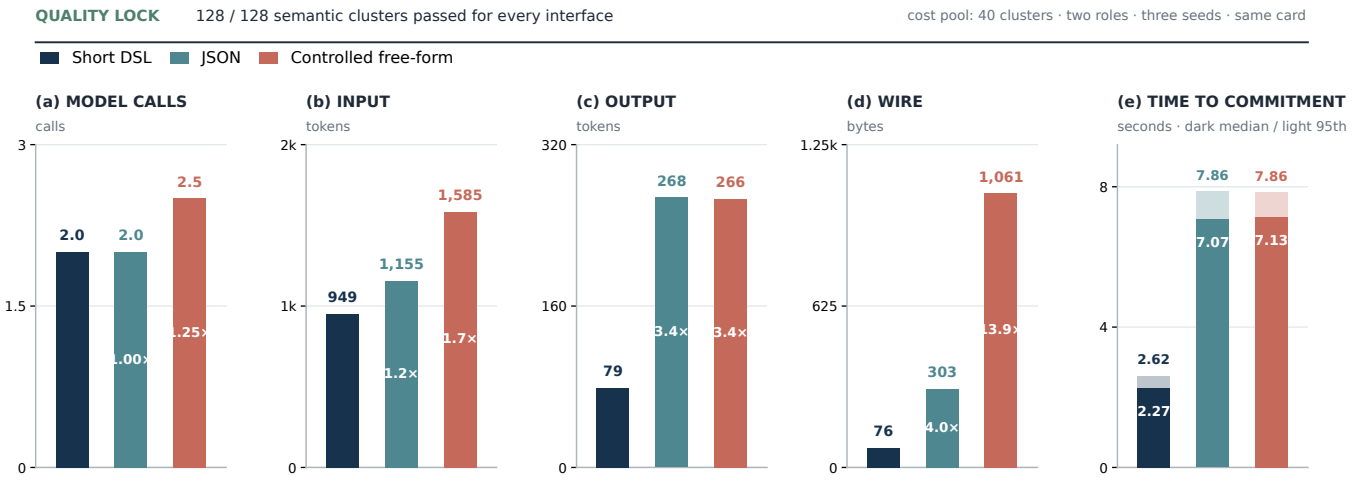

Figure S3. Quality-matched coordination costs on independent physical-unit axes. All three interfaces pass the 128-cluster quality pool; costs use the separate 40-cluster same-card timing pool. In the TTC panel, dark bars end at p50 (the median) and pale extensions end at p95 (the 95th percentile).

Table S11. Same-card latency and quality-pool reliability. The two panels retain their separate 40- and 128-cluster denominators.

(a) Call latency: 40-cluster same-card pool		
Surface	**p50**	**p95**
Short DSL call latency	1.143 s	1.660 s
JSON call latency	3.531 s	5.941 s
Controlled free-form call latency	3.274 s	3.746 s
(b) Reliability and symmetry: 128-cluster quality pool		
Strict successful clusters, each surface		128/128
Exact-binomial 95% lower bound		0.972
Role-swap success difference		0.0 pp
Missing / fallback rows		0 / 0

S6 Request Semantics and Strong Baselines

Takeaway. The delivered binding is causally active: removing it restores the paired 50% ceiling, while replacing it with the other executable option drives the receiver to the paired world-state branch.

Table S12 holds message presence, protocol schedule, schema, and learned-call count fixed while changing only the delivered task content; the deterministic row supplies an exact binding-selection ceiling for the enumerable library.

Table S12. Request intervention and deterministic binding reference on 160 clusters. Learned rows contain three seeds (960 episodes); the seedless deterministic rule runs once per role view (320 episodes). Paired differences are true request minus the listed condition. Bytes are canonical request-object JSON, distinct from the protocol-specific Short DSL request-line bytes in Table S10.

Condition	**Episodes**	**Success**	**Calls**	**Req.-object JSON B**	**True − condition**	**Mechanism readout**
True learned request	960	100.0%	2.0	304.4	0.0 pp	reference success
Request removed / self-plan-only	960	50.0%	2.0	0.0	+50.0 pp [42.5, 57.5]	paired ambiguity exposed
Same-template executable alternative	960	0.0%	2.0	304.4	+100.0 pp [100, 100]	receiver follows delivered alternative
Deterministic correct binding	320	100.0%	0.0	304.4	0.0 pp	exact binding ceiling

The alternative intervention selects the other executable request from the same template option set and preserves message presence, schedule, schema, approximate length, and learned-call count. The receiver follows the delivered alternative in 960/960 CONFIRM episodes and 471/471 executable held-out cases. Every executable held-out output preserves the delivered binding; the nine residual losses occur before an executable plan is formed.

S7 Commitment Horizon and Template Shift

Takeaway. Under the corresponding closed-loop training protocols, multi-step commitments improve held-out success and reduce online decisions; the gain concentrates in allocation and active-continuation templates.

Table S13 reports the paired aggregate effects and family decomposition; Figure S4 shows where the success gain appears and how much online burden remains.

The comparison evaluates complete systems under their corresponding training protocol. Multi-step training uses 1,920 rows and 240 optimizer updates per seed. Single-step training uses 2,880 rows, 360 updates, and an additional `sender_after_obtain` stage. This design gives the single-step system explicit intermediate-state supervision; the measured difference therefore combines commitment horizon with each protocol's online control pattern.

The deterministic correct-binding rule supplies an exact binding-selection ceiling on the same held-out pool: 100.0% versus 98.1% for multi-step GenCoord, a paired GenCoord-minus-rule difference of −1.9 pp (95% CI $[-4.2, -0.2]$). GenCoord preserves the correct binding in every executable output; its nine residual losses arise before plan formation. The comparison therefore separates exact binding sufficiency from learned commitment formation while retaining the same executable interface.

Table S13. Held-out-template horizon results across 80 independent clusters. Every difference is multi-step minus single-step. Family rows are descriptive; confidence intervals are reported for the aggregate paired comparisons.

Family / metric	Multi-step	Single-step	Δ	95% CI	Interpretive feature
Destination	100.0%	100.0%	0.0 pp	—	destination binding
Recipe	100.0%	100.0%	0.0 pp	—	transformation binding
Allocation	92.5%	83.3%	9.2 pp	—	actor / remaining workload
Active branch	100.0%	81.7%	18.3 pp	—	downstream continuation
Overall success	98.1%	91.3%	6.9 pp	[2.9, 10.8] pp	paired cluster bootstrap
Decisions / episode	1.981	2.913	−0.931 calls / ep.	[−0.971, −0.892] calls / ep.	online coordination cost
Input tokens / episode	1,146.5	1,689.1	−542.6 tok./ep.	[−567.8, −517.6] tok./ep.	repeated context avoided

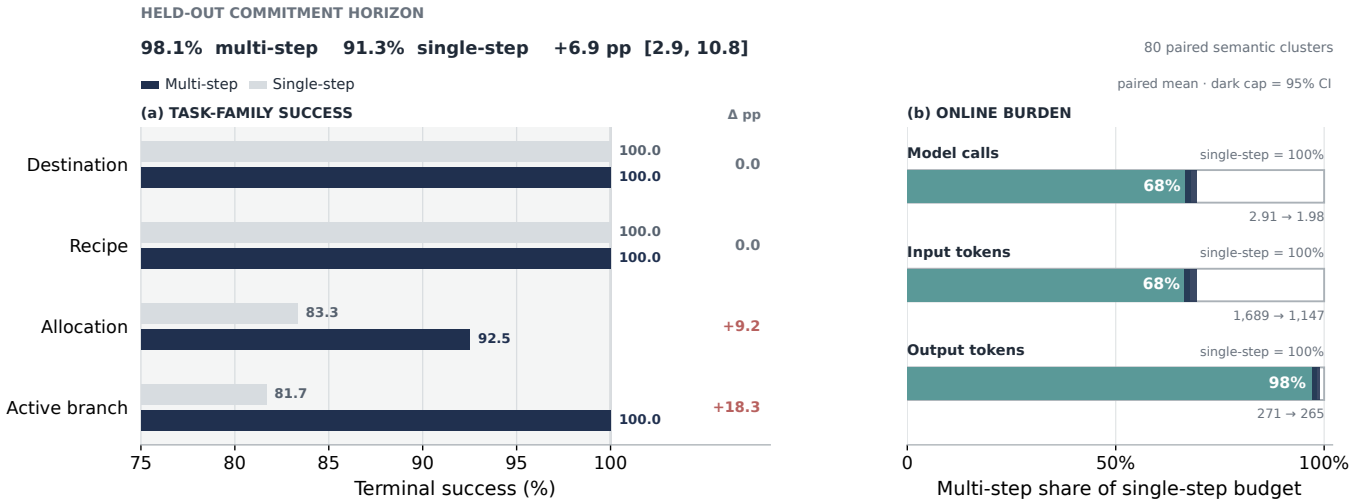

Figure S4. Paired commitment-horizon comparison over 80 held-out semantic clusters. (a) Task-family terminal success. (b) Multi-step online burden as a share of the corresponding single-step mean; dark caps map the paired absolute-difference confidence intervals from Table S13 onto that observed denominator.

S8 Backend and Composition Analyses

Takeaway. For the enumerable task library, a learned binding code plus deterministic rule is a faster quality-matched backend. Direct Short DSL retains an explicit executable path as its model output.

Table S14 isolates backend formation while holding the peer-facing commitment fixed. Table S15 then separates complete-case lookup from field-wise construction on unseen binding cross-products.

Table S14. Quality-matched direct and factor-code backends. Fresh success uses 128 clusters (two roles, three seeds); token counts and TTC use a separate 40-cluster same-card planning pool.

Metric	Factor + rule	Direct DSL
Fresh success	768/768	768/768
Input tokens	980.2	948.9
Output tokens	19.4	79.1
Protocol-specific peer payload	76.6 B	76.6 B
TTC p50	0.824 s	2.453 s
TTC p95	0.997 s	2.723 s

The factor target is the sender binding ID before delivery and the receiver accepted-binding ID after delivery. Both models receive the same model-visible information. A deterministic composer recovers actor, path, object, destination, dependencies, and the same peer-facing Short DSL request from the selected code.

Table S15. Composition reference on 60 unseen binding cross-products.

Method	Terminal success
Complete-case table	0/60
Explicit-factor composer	60/60
Direct Short DSL	60/60

The composition reference separates exact case lookup from field-wise construction. Explicit factor composition and Direct Short DSL recover all 60 tested cross-products, while exact complete-case lookup covers the observed cases only. Table S14 further shows that, when a public codebook uniquely determines the branch, factor-code generation provides a lower-latency path to the same peer-facing commitment.

S8.1 Naturalized private facts without binding IDs

The active binding field is next replaced by naturalized private-fact descriptions while the same 16 semantic task cells, public prior, and grounded executor are retained. Table S16 reports one Minecraft closed-loop condition with test-only syntactic recombinations and two stronger planning-only surface shifts.

Table S16. Naturalized private-fact formation after removing the active binding field. C1 is Minecraft closed-loop evaluation on held-out syntactic recombinations; C2 and C3 are planning-only lexical-paraphrase and paraphrase-plus-distractor conditions. Learned rows aggregate three seeds (960 episodes per condition); seedless model-free rows use 320 episodes.

Formation method	C1 live	C2 planning	C3 planning
Direct Short DSL (0.8B)	100.0%	65.8%	64.8%
Learned factor + rule (0.8B)	100.0%	**79.1%**	**80.1%**
Frozen factor parser rule	100.0%	0.0%	0.0%
TF–IDF nearest-neighbour rule	100.0%	59.4%	62.5%
Complete surface table	0.0%	0.0%	0.0%
Oracle factors + rule	100.0%	100.0%	100.0%

Under C1, both learned backends and the two stronger non-LLM semantic mappings retain 100% closed-loop success, while exact-string lookup covers the training surfaces only. The commitment interface therefore remains executable after the active answer field is replaced by test-only naturalized descriptions. Under stronger surface shifts, learned factor formation reaches 79.1% and 80.1%, exceeding TF–IDF by +19.7 pp (95% CI [11.8, 27.9]) and +17.6 pp ([9.2, 26.5]), respectively; Direct Short DSL reaches 65.8% and 64.8%. The result strengthens the interface/backend separation: compact factor prediction is the more robust realization under stronger lexical change.

S9 Task-Structure Visibility

Takeaway. Full and Skeleton views preserve the familiar post-handoff branch; the analysis cleanly separates that recovered motif from transformation-order motifs that require additional structural generalization.

Figure S5 reports the full 3-method × 3-view × 3-motif matrix with exact percentages in every cell.

Full DAG exposes the public nodes, typed fields, coarse stages, predecessor lists, and edges; its symbolic reference follows a topological ordering. Skeleton removes predecessor lists and edges while retaining coarse stage, and its symbolic reference uses that stage information. Node-only removes stages and edges, applies a stable answer-independent node arrangement, and uses a fixed task-path priority. The three views are categorical projections with distinct answer-independent ordering policies. Across 96 clusters, two role views, and three frozen checkpoints, the matrix shows that Direct Short DSL preserves the familiar post-handoff branch under Full and Skeleton views, while the two transformation-order motifs remain at 0%. The symbolic rule's non-monotone Node-only recovery reflects its fixed task-path priority under a different projection.

S10 E1/E2: Feedback Causality and Centralized Reference

Takeaway. Across 40 Goal–Capability semantic clusters (20 matched counterfactual pairs) and three independently trained seeds, requester-local planning reaches 50%, correct feedback reaches 100%, counterfactual feedback reaches 0%, and centralized full information reaches 100%.

S10.1 Paired construction and matched training

The evaluation crosses two private goals, two peer-local workcell modes, and ten world variants, producing $2 \times 2 \times 10 = 40$ semantic clusters. Table S17 records the matched training contracts; Table S18 reports the four-condition outcome

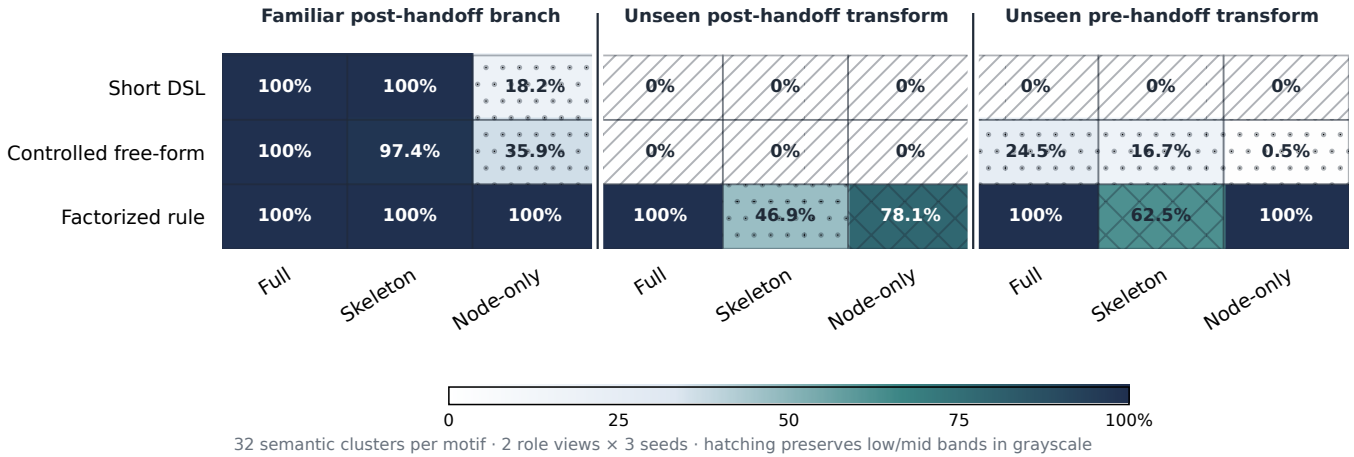

Figure S5. Annotated task-structure robustness matrix. Columns pair each topology motif with Full, Skeleton, and Node-only projections; cells show exact terminal success.

profile; Table S19 isolates the fields controlled by feedback; and Figure S6 traces the capability-conditioned routes and intervention. Holding goal and world variant fixed while changing the workcell mode creates 20 matched counterfactual pairs. Within each pair, the requester's model-visible input and initial proposal are byte-identical; the peer-local mode changes the feasible craft actor, handoff object, and downstream suffix. Each condition evaluates all 40 clusters under two role permutations and three training seeds, yielding 240 episodes per condition and 960 episodes overall. The primary analysis averages the six role–seed observations within semantic cluster and resamples 40 clusters. A pair-block sensitivity analysis additionally averages both capability worlds within each goal–world pair and resamples the 20 matched pair blocks.

Table S17. Training contracts for the Goal–Capability models. Train-token occurrences sum tokenized prompt, target, and EOS across all epochs.

Model	Visible information	Rows	Epochs	Updates	Train-token occurrences	Seeds
Distributed requester	320 initial local + 320 after-response local	640	2	80	868,720	3
Centralized full information	merged current local views	320	4	80	970,080	3

The distributed data contain 320 initial local records and 320 local records after the bounded response. The centralized data contain 320 joint-plan records. Both models use Qwen3.5-0.8B, effective batch size 16, learning rate 10^{-5}, maximum length 2,048, and final-step checkpoints, with DEV reserved for implementation checks. Matching optimizer updates gives the centralized model 1.117 times the distributed train-token occurrences. The reported totals include prompt, target, and EOS tokens over all sample occurrences; every row remains below the 2,048-token truncation limit.

S10.2 E1: feedback-content intervention

Table S18. E1/E2 outcomes on 40 semantic clusters, arranged as 20 matched counterfactual pairs. Intervals are primary cluster-bootstrap 95% CIs. Strict success requires all six role–seed observations of a cluster to succeed; the final column names the measured boundary component rather than total architecture traffic.

Condition	Seeds	Epis.	Success	95% CI	Strict	Calls	Measured boundary component
Requester-local (A-only)	3	240	50.0%	[35, 65]	20/40	1.0	0 B cross-boundary payload
Correct bounded feedback	3	240	100.0%	[100, 100]	40/40	2.0	response: mean 103 B
Counterfactual feedback	3	240	0.0%	[0, 0]	0/40	2.0	injected response: mean 103 B
Centralized full information	3	240	100.0%	[100, 100]	40/40	1.0	state aggregation: mean 956 B

Correct feedback improves success over requester-local planning by +50.0 percentage points (primary 40-cluster 95% CI [35, 65]). Replacing only the response with the paired counterfactual workcell mode changes the same two-call protocol from 100% to 0% (paired difference +100.0 points, 95% CI [100, 100]). The 20-pair block sensitivity gives +50.0 points [50, 50] and +100.0 points [100, 100], respectively. Every seed reproduces the 50/100/0 profile with descriptive seed-level SD 0.0. All 720 distributed episodes are parse-valid, condition-specific contract-valid, planning-complete, and

fallback-free. Counterfactual commitments remain internally valid under the injected response, while the unchanged executable world provides an independent terminal check.

Table S19. Commitment fields under the counterfactual-feedback intervention.

Field	Matches injected	Matches true world	Observed behavior
Craft actor	240/240	0/240	rewritten
Handoff item	240/240	0/240	rewritten
Handoff count	240/240	0/240	rewritten
Peer suffix	240/240	0/240	rewritten
Goal item	240/240	240/240	invariant
Goal count	240/240	240/240	invariant

The intervention preserves the requester input, initial proposal, executable world, model checkpoints, decoding contract, and two-call schedule. Counterfactual feedback controls the final commitment in all 240 episodes: craft actor, handoff item, handoff count, and peer suffix follow the injected capability consequence, while goal item and goal count remain unchanged. Each complete commitment therefore reaches the actor selected by the injected response, and the unchanged world rejects that counterfactual actor at terminal verification.

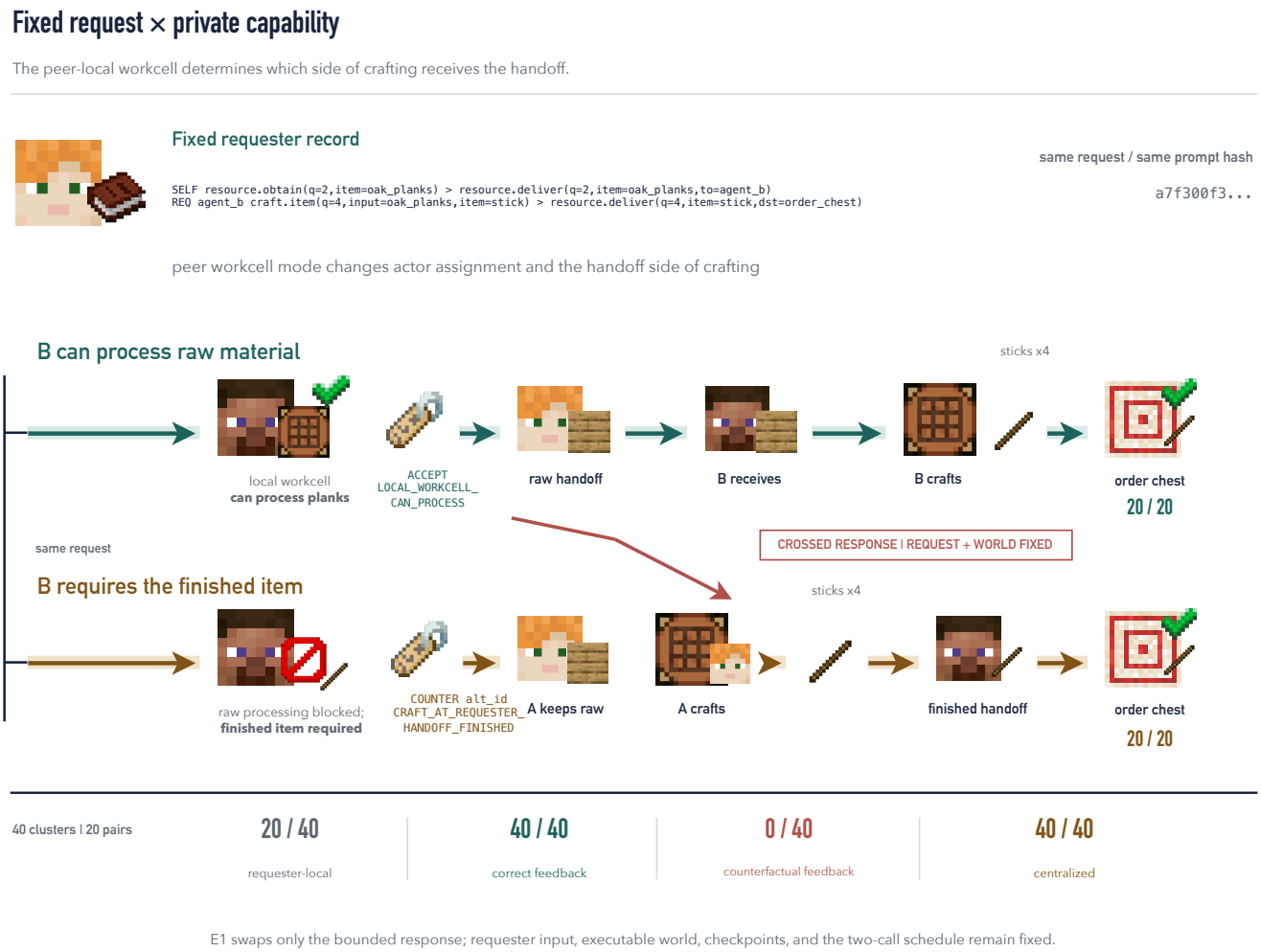

Figure S6. Capability-conditioned commitments and the counterfactual-feedback intervention with requester input and executable world held fixed.

Figure S6 illustrates a correct-feedback pair that shares one requester prompt hash. A `RAW_PROCESSOR` peer accepts the raw-material handoff. A `FINISHED_RECEIVER` peer counters with `CRAFT_AT_REQUESTER_HANDOFF_FINISHED`; the requester moves crafting into its own path and hands off the finished item. Both correct routes reach the terminal

checker. The E1 intervention injects the matched counterfactual response while keeping the requester input, initial proposal, and executable world fixed.

S10.3 E2: three-seed centralized reference

Centralized full information succeeds in 240/240 episodes across the same three seeds and 40 clusters, matching correct feedback with a paired difference of 0.0 points (95% CI $[0, 0]$). It assembles the two current local views before one learned call. The distributed route keeps the views separate and uses a bounded response with mean 103 B and p95 118 B before its second learned call. The centralized state-aggregation component has mean 955.5 B and p95 963 B (rounded to 956 B in Table S18). The 103 B value measures the bounded response, while 955.5 B measures centralized state aggregation. The architecture-level equations below account for every application message and make the component boundaries explicit.

For architecture accounting, let $\mathcal{A}_t \subseteq \mathcal{A}$ be the participating agents at round t, let $E_t \subseteq \mathcal{A} \times \mathcal{A}$ be the directed active-edge set, and let $\mathcal{M}_{ij}^t$ be the complete sequence of application messages sent on directed edge (i, j). The function $b(m)$ counts canonical UTF-8 payload bytes, including any application wrapper present in m but excluding transport framing. Let $m_{i,\uparrow}^t$ and $m_{i,\downarrow}^t$ denote centralized upload and download payloads. With R_c centralized rounds and R_l local rounds,

$$B_{\text{central}} = \sum_{t=1}^{R_c} \sum_{i \in \mathcal{A}_t} \left[b(m_{i,\uparrow}^t) + b(m_{i,\downarrow}^t) \right], \tag{S4}$$

$$B_{\text{local}} = \sum_{t=1}^{R_l} \sum_{(i,j) \in E_t} \sum_{m \in \mathcal{M}_{ij}^t} b(m). \tag{S5}$$

Feedback sent in the reverse direction belongs to the corresponding edge (j, i). These equations define total interface accounting and are distinct from the component measurements above. Both architectures depend on coordination rounds; local communication also depends on active-edge density and messages per active edge.

S11 Reproducibility Notes

Pre-release validation checks the bundled compact data, method identities, pool denominators, non-null horizon means, paired-difference direction, confidence intervals, bootstrap seeds, and 10,000 resamples. It also checks the corrected single-step inventory of 2,880 rows, three stage counts of 960, and 360 optimizer updates per seed. For E1/E2, the arXiv ancillary data contain twelve evaluation lanes and 960 episode metrics stripped of host, scheduler, process, and private-path identifiers, including all 240 counterfactual-feedback control episodes. The deterministic table builder regenerates Tables S17–S19 from the canonical analysis and matched training-budget files. Earlier representation, request, horizon, and visibility results are reconstructed from the bundled aggregate analyses and fixed paired-comparison records; E1/E2 additionally include de-identified episode-level metrics.

Statistical procedure. Representation, request, horizon, visibility, and the primary E1/E2 analyses use the semantic cluster as the independent unit. Where role views or training seeds repeat a cluster, their outcomes are averaged within cluster before inference. Paired effects use cluster-aligned differences. Reported bootstrap intervals use 10,000 percentile resamples with fixed analysis seeds; E1/E2 condition intervals use seeds 8901–8904 and primary paired-difference intervals use 8911–8914. The E1/E2 sensitivity aggregates both capability clusters within each goal–world pair and resamples 20 pair blocks with seeds 8921–8924. Exact-binomial intervals are two-sided 95% Clopper–Pearson intervals. Seed-level standard deviations summarize training-run stability, while cluster bootstrap intervals provide inferential uncertainty.

Table S20 links every reader-facing result family to its compact canonical source.

Packaged sources. The arXiv package contains the bibliography, canonical Supplement source, compact ancillary source data, final vector figure exports, and deterministic E1/E2 table builder. Reader-facing paths are package-relative. Experimental runtime dependencies include Qwen3.5-0.8B, Mineflayer 4.37.1, Node.js, Python, PyTorch, Transformers, and the official Minecraft Java server 1.21.4; each is governed by its upstream license or terms.

Build sequence. Regenerate the E1/E2 tables with:

```
python3 anc/scripts/build_e1e2_tables.py \
  --analysis anc/source_data/e1e2_analysis.json \
  --training anc/source_data/e1e2_training_budget.json \
  --tables-dir tables --data-dir anc/source_data
```

Then compile `supplement.tex` with the bundled bibliography. Final vector figure PDFs are included under `figures/`; editable authoring sources, render caches, and contact sheets are intentionally excluded from the arXiv upload package.

Table S20. Claim-to-artifact map for the compact arXiv ancillary bundle.

Result family	Canonical bundle artifact
Evidence-suite and task inventory	`anc/source_data/evidence_suites.csv`; `anc/source_data/task_templates.csv`
Representation quality and cost	`anc/source_data/representation_cost.csv`
Request intervention	`anc/source_data/request_intervention.csv`
Commitment horizon	`anc/source_data/horizon_summary.json`; `anc/source_data/horizon_family.csv`; `anc/source_data/paired_comparisons.json`
Structure visibility	`anc/source_data/visibility_motifs.csv`; `anc/source_data/visibility_overall.csv`
Grounded runtime trace	`anc/source_data/runtime_trace.json`; `anc/source_data/trace_registry.csv`
E1/E2 outcomes and field fidelity	`anc/source_data/e1e2_episode_metrics.jsonl`; `anc/source_data/e1e2_analysis.json`; `anc/source_data/e1e2_field_fidelity.csv`; `anc/source_data/e1e2_config.json`
Training budgets and runs	`anc/source_data/e1e2_training_budget.json`; `anc/source_data/training_runs.csv`
Backend and composition references	`anc/source_data/backend_comparison.csv`; `anc/source_data/composition_reference.csv`
Naturalized private-fact formation	`anc/source_data/naturalized_private_fact_summary.csv`; `anc/source_data/naturalized_private_fact_paired.csv`; `anc/source_data/naturalized_private_fact_scope.json`
Table regeneration	`anc/scripts/build_e1e2_tables.py`

References

[PrismarineJS(2026)] PrismarineJS. 2026. Mineflayer: Create Minecraft Bots with a High-Level JavaScript API. Software repository. Version 4.37.1 used in the project; accessed 2026-08-08. `https://github.com/PrismarineJS/mineflayer`

[Qwen Team(2026)] Qwen Team. 2026. Qwen3.5-0.8B. Hugging Face model card. Accessed 2026-08-08. `https://huggingface.co/Qwen/Qwen3.5-0.8B`