Beyond the All-in-One Agent: Benchmarking Role-Specialized Multi-Agent Collaboration in Enterprise Workflows

Tao Yu[1,2,3*§], Hao Wang[2,*†], Changyu Li[1,4§], Shenghua Chai[2†], Minghui Zhang[2†], Zhongtian Luo[2†], Yuxuan Zhou[5], Haopeng Jin[2†], Zhaolu Kang[4],, Jiabing Yang[2,3], YiFan Zhang[2,3], Xinming Wang[2,3], Hongzhu Yi[3], Zheqi He[1‡], JingShu Zheng[1], Xi Yang[1], Yan Huang[2,3‡], Liang Wang[2,3]

[1]BAAI [2]CASIA [3]UCAS [4]Peking University [5]Tsinghua University

Large language model (LLM) agents are increasingly expected to operate in enterprise environments, where work is distributed across specialized roles, permission-controlled systems, and cross-departmental procedures. However, existing enterprise benchmarks largely evaluate single agents with broad tool access, while existing multi-agent benchmarks rarely capture realistic enterprise constraints such as role specialization, access control, stateful business systems, and policy-based approvals. We introduce EntCollabBench, a benchmark for evaluating enterprise multi-agent collaboration. EntCollabBench simulates a permission-isolated organization with 11 role-specialized agents across six departments and contains two evaluation subsets: a Workflow subset, where agents collaboratively modify enterprise system states, and an Approval subset, where agents make policy-grounded decisions. Evaluation is based on execution traces, database state verification, and deterministic policy adjudication rather than natural-language response judging. Experiments with representative LLM agents show that current models still struggle with end-to-end enterprise collaboration, especially in delegation, context transfer, parameter grounding, workflow closure, and decision commitment. EntCollabBench provides a reproducible testbed for measuring and improving agent systems intended for realistic organizational environments.

Date: May 9, 2026
Code: https://github.com/yutao1024/EntCollabBench
Data: https://huggingface.co/datasets/Kirito-Lab/EntCollabBench

1 Introduction

Large language model (LLM) agents [1] are increasingly expected to operate in enterprise environments [2, 3, 4, 5, 6], where work is distributed across specialized roles, permission-controlled systems, and cross-departmental procedures. Completing a business request therefore requires more than isolated tool use: agents must infer responsibility boundaries, delegate to the right role, preserve context, and update stateful systems correctly.

Recent enterprise agent benchmarks have advanced realistic workplace evaluation by using service platforms,

*Equal contribution.
†Work done during an internship at CASIA.
§Work done during an internship at BAAI.
‡Corresponding author.

CRM systems, code repositories, and collaboration tools [7]. However, most still assume a single agent with broad tool access, which differs from real organizations where HR, IT, customer support, engineering, and approval roles have separate responsibilities and permissions. Conversely, existing multi-agent benchmarks study communication and coordination, but usually in games, puzzles, or abstract distributed-information settings rather than enterprise workflows with access control, persistent records, and policy constraints [8, 9, 10].

We introduce ENTCOLLABBENCH, a benchmark for evaluating enterprise multi-agent collaboration. It simulates a permission-isolated organization with 11 role-specialized agents across six departments: IT, Human Resources, Customer Service, Shared Services, Engineering, and an Approval Center. Given a natural-language instruction and a designated starting agent, agents must complete tasks through cross-departmental delegation and communication, using only tools within their assigned responsibility scope.

ENTCOLLABBENCH contains two evaluation subsets. The **Workflow subset** covers operational tasks that modify enterprise system state, such as creating incidents, updating HR cases, scheduling meetings, revising knowledge articles, and submitting pull requests. The **Approval subset** covers policy-grounded decisions by finance, legal, and procurement specialists. Workflow tasks are evaluated through execution traces and database state diffs, while approval tasks are evaluated against deterministic policy adjudications.

Experiments show that current LLM agents still struggle with enterprise collaboration. Although many models can perform local role-specific actions, end-to-end success drops markedly when tasks require multi-hop delegation, context transfer, final-stage coordination, and parameter-level grounding in stateful systems.

Our contributions are:

We formulate enterprise multi-agent collaboration as an evaluation setting centered on role specialization, permission isolation, implicit routing, context transfer, and stateful cross-departmental workflows.

We introduce ENTCOLLABBENCH, a benchmark with 11 role-specialized agents across six departments, covering both operational workflow execution and policy-based approval decisions with objective verification.

We evaluate representative LLM agents on ENTCOLLABBENCH and identify key bottlenecks in delegation, parameter grounding, workflow closure, decision commitment, and coordination cost.

2 Related Works

Single-Agent Enterprise Benchmarks. In recent years, agent evaluation benchmarks targeting enterprise scenarios have advanced rapidly. AgentBench [3] was the first to systematically evaluate LLMs as agents across diverse environments. WorkArena [4] and WorkArena++ [5] built web-based task suites for knowledge workers on the ServiceNow platform. TheAgentCompany [6] simulated a corporate environment equipped with tools such as GitLab and RocketChat to assess agents on realistic enterprise tasks. EntWorld [11] further scaled to 1,756 GUI tasks spanning six enterprise domains including CRM and ITSM. In addition, Finch [12] and CRMArena [13] constructed domain-specific benchmarks for finance & accounting and CRM, respectively. However, all of the above adopt a single-agent paradigm—one agent assumes every role—without involving inter-agent communication or cross-role collaboration, leaving a significant gap with the multi-departmental division of labor found in real enterprises. The differences are shown in Figure 1.

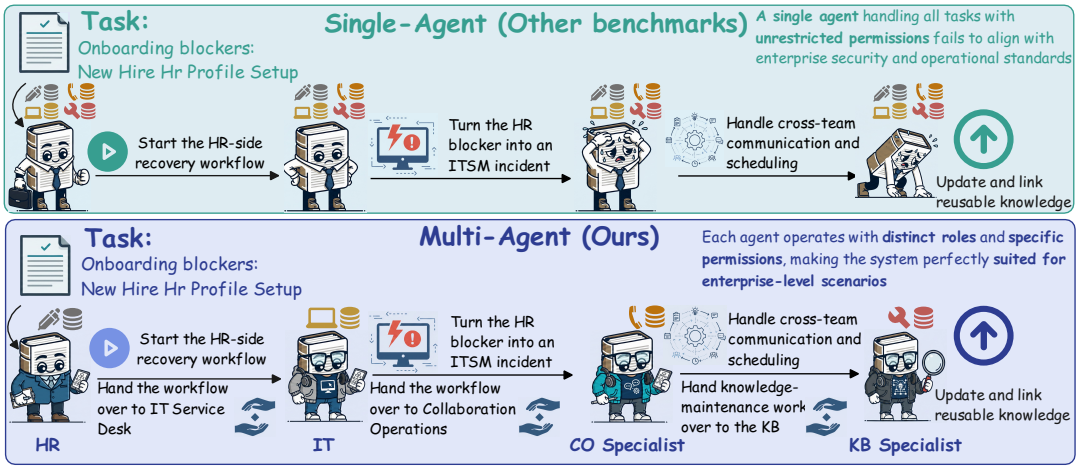


Figure 1. Comparison of EntCollabBench with other enterprise benchmarks.

Multi-Agent Collaboration Benchmarks. Early work on multi-agent evaluation centered on reinforcement learning, where SMAC [8] and Overcooked-AI [9] assessed cooperative strategies in game environments. In the LLM era, COMMA [10] evaluated communicative collaboration among multimodal agents under information asymmetry, MultiAgentBench [14] measured cooperative and competitive behaviors in scenarios such as Minecraft, and SILO-BENCH [15] tested multi-agent coordination under distributed information. Although these works involve inter-agent communication, their scenarios are limited to games, puzzles, or algorithmic tasks and do not address more complex, real-world settings such as enterprise workflows.

3 EntCollabBench

EntCollabBench (Figure 2) evaluates whether large language model agents can perform real enterprise work under the constraints that distinguish organisational settings from sandboxed task environments: role specialization, permission isolation, stateful business systems, and cross-departmental delegation.

3.1 Task Definition and Design Scope

We define each benchmark instance as a constrained collaboration task in the organizational environment introduced above. Given a natural-language instruction and a designated starting agent, agents must complete the task using only role-authorized tools and explicit cross-role delegation. Thus, success depends on responsibility inference, context transfer, and stateful workflow execution rather than isolated tool use.

This task definition induces four design constraints. First, **role isolation** is strictly enforced: agents cannot directly call tools or query data outside their own department, and must instead communicate necessary context to the appropriate downstream role. This creates information asymmetry, where insufficient delegation messages can cause execution failures, while excessive or noisy context may mislead downstream agents. Second, **collaboration dependency** is required: every task involves agents from at least two departments, ensuring that success depends on multi-agent planning and communication rather than single-agent tool use. Third, tasks require **implicit routing and decomposition**: natural-language instructions describe business

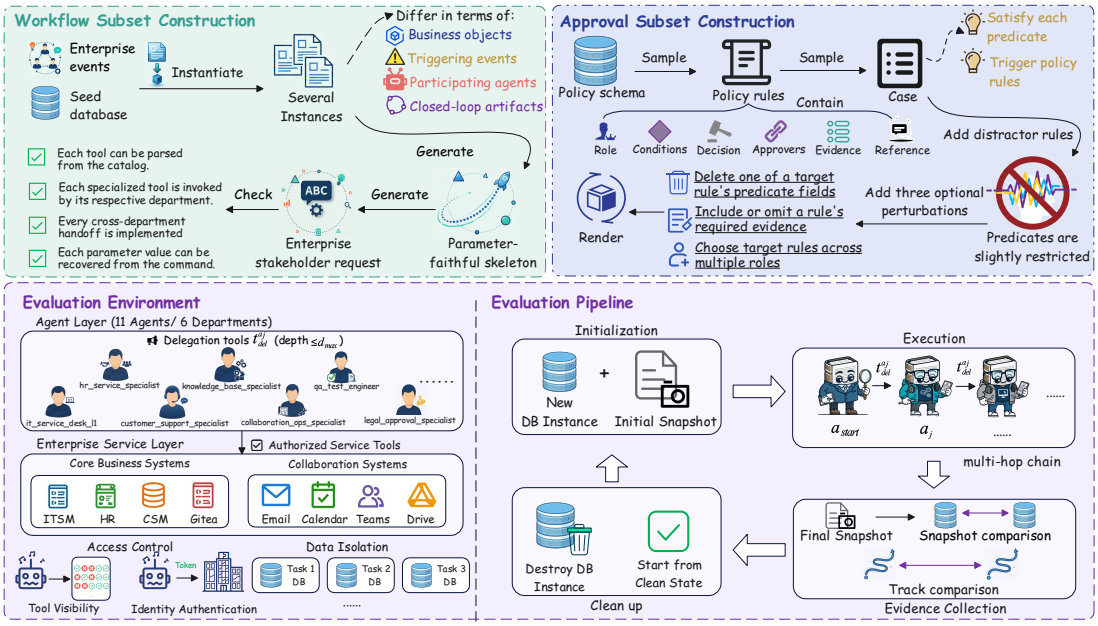


Figure 2. Overview of EntCollabBench. The Workflow Track generates tasks across business domains and process intents, producing instances with different objects, events, agents, and artifacts. The Approval Track constructs requests from sampled rules with predicate satisfaction and optional perturbations. The Evaluation Environment includes 11 agents over 6 departments with controlled access to enterprise systems. The Evaluation Pipeline proceeds through DB initialization, multi-hop execution starting from a designated agent, snapshot and trace event collection, and DB cleanup.

goals rather than explicit workflows, so agents must infer responsibility boundaries, identify the next role, and decide which actions to perform. Finally, the workflows are **long-horizon and stateful**, involving multiple tool calls and delegation rounds, where local errors in parameters, routing, or semantics can silently propagate and cause end-to-end failure.

The task suite covers realistic enterprise operations, including ticket handling, incident response, onboarding coordination, customer escalation, knowledge-base maintenance, document approval, and code review. Each task is paired with deterministic ground truth specifying the expected final system state, key parameter constraints, or policy decision outcome. Evaluation is therefore objective and reproducible: it is based on system-state changes, critical tool-call parameters, and policy-grounded decisions rather than the agents' natural-language responses.

3.2 Evaluation Subsets

Enterprise work in our benchmark is divided into two structurally distinct subsets. The dataset statistics are shown in Figure 3.

Workflow subset covers operational departments, including IT, HR, Customer Service, Shared Services, and Engineering. Tasks in this subset require agents to modify states in external enterprise systems, such as creating incidents, updating HR cases, sending emails, scheduling meetings, revising knowledge articles, or submitting pull requests. Evaluation is performed by extracting the trace events of each agent and comparing them with the structured ground-truth specification. To further verify that actions are truly executed rather than merely proposed, we also collect initial and final snapshots of the relevant databases and examine their

diffs for the expected state transitions.

Approval subset covers the Approval Center, whose three specialists (finance, legal, procurement) emit policy-grounded decisions rather than mutate external system state. Given a submitted request, each involved specialist must determine whether the request complies with internal policies and external regulations, cite the rules that justify the decision, and flag cases where the request is missing information required by an applicable rule. We evaluate this subset by *policy-based verification*: each case is scored against a deterministic per-specialist reference decision derived from a curated policy schema, comparing the decision label, the supporting rule citations, and any required-information flags against a reference adjudication.

Both subsets share the same cross-agent delegation protocol, they differ only in the verification target: operational workflow tasks are evaluated by final system state, while approval tasks are evaluated by policy-grounded adjudication.

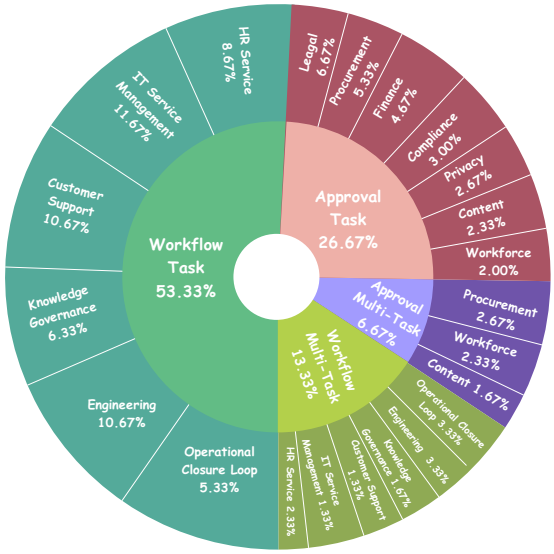


Figure 3. Dataset statistics. The benchmark contains 300 tasks: 160 workflow, 40 workflow multi-task, 80 approval, and 20 approval multi-task tasks, spanning six workflow and seven approval categories.

3.3 Data Construction

The two subsets share a common goal: producing executable, verifiable enterprise tasks at scale, but their construction pipelines reflect their different verification semantics. The Workflow subset is built bottom-up from a fixed enterprise tool catalog and seed system state; the Approval subset is built top-down from a structured policy schema of policy rules extracted from authoritative sources.

3.3.1 Source Materials

The Workflow subset is built from a *tool catalog* and a *seed database*. The tool catalog defines the enterprise services available to agents, including the permitted tools and parameter schemas for each service. The seed database populates the simulated enterprise systems with concrete business objects, ensuring that all task parameters and ground-truth trajectories refer to valid objects in the environment. The design of the tool catalog and seed database is informed by EnterpriseOps-Gym [7].

The Approval subset is built from a curated policy corpus consisting of 60 pages from the GitLab Handbook [16] and eleven GDPR articles [17]. The GDPR articles supplement the Handbook with broader data-protection rules. After removing navigation and footer boilerplate, the corpus contains approximately 840K characters of policy text and is processed using the same downstream pipeline.

3.3.2 Workflow Task Construction

Workflow tasks are generated category-first. We predefine business categories from common enterprise events and instantiate cases from seed-database parameters, varying the business object, triggering event,

agents, and closure artifact. For each case, we generate the ground-truth trajectory from one of 20 manually specified domain templates, each covering one business domain and iterated over five trigger types and five governance rules. The trajectory starts from the agent whose backend owns the trigger and records each step's executing agent, MCP server, tool name, and arguments. Cross-department actions are modeled as explicit delegation events that transfer a subtask and its supporting context to the appropriate downstream role. These events are implemented as role-specific delegation tools, such as `ask_<peer>_by_http`. A worked-out template appears in Appendix D.

We further verify that each accepted trajectory uses cataloged tools, respects tool ownership, delegates cross-department transitions explicitly, and recovers all argument values from the user instruction. We further verify that each accepted trajectory uses cataloged tools, respects tool ownership, delegates cross-department transitions explicitly, and recovers all argument values from the user instruction.

Cases with a natural multi-stage structure are further partitioned into multi-step tasks. Such cases typically consist of a record-establishment phase, a technical-resolution phase, and a coordination-and-closure phase. Each phase is converted into a sub-task with its own starting agent, its own instruction conditioned on context produced by upstream phases, and its own ground-truth fragment containing only the steps executed within that phase. We require each multi-step task to include at least three cross-agent delegations across the full chain, ensuring that the task cannot be reduced to sequential tool calls by a single agent.

3.3.3 Approval Task Construction

Approval tasks are constructed from structured policy rules. Each task is generated by sampling target rules from a policy schema, constructing a case that triggers those rules, and rendering the case into a submission package with deterministic ground truth.

Policy schema construction. We transform the policy corpus into a structured schema through four stages: a heading-aware chunker; an LLM classifier (GPT-5.4-Mini) that labels each chunk with its primary role (finance, legal, or procurement) and flags chunks containing approval-relevant policy rules; an LLM extractor (GPT-5.4) that emits rules in a strict JSON schema; and a finalization stage that filters low-quality extractions, deduplicates entries, and normalizes field names.Each extracted rule specifies a primary role, conjunctive typed-field conditions, a decision label in {`approve`, `reject`, `require_docs`, `require_preapproval`, `not_applicable`}, an approver chain, fulfillment-evidence slugs, and a source citation. The citation includes verbatim policy text and must be a contiguous substring of the source chunk, providing a falsifiable grounding signal that prevents hallucinated rules from entering the schema. The finalized policy schema contains 290 rules: 42 finance rules, 154 legal rules, and 94 procurement rules. A policy schema generation example appears in Appendix B.

Approval Task synthesis. Each Approval task is generated deterministically from the structured policy schema. We first sample one or more target rules and instantiate a case whose fields satisfy their conditions. We then add plausible but non-applicable distractor rules to reduce reliance on shallow pattern matching while preserving the ground truth. Three optional perturbations operate at the decision level: (i) deleting one of a target rule's predicate fields forces the agent to recognize missing information; (ii) including or omitting a rule's required evidence controls whether its preapproval requirement is discharged; and (iii) choosing target rules across multiple roles forces cross-departmental adjudication. The constructed case is then rendered into an intake form summarizing the business context and submitted parameters, optional supporting-evidence documents, and a role-specific directive that asks each specialist to review the case against named policy documents. Ground truth is computed by a deterministic *decision engine* that produces per-specialist decisions with supporting rule citations. A task generation example appears in Appendix C.

The resulting case is rendered into an intake form, optional evidence documents, and role-specific review

instructions. A deterministic decision engine then computes the expected per-specialist decision and its supporting rule citations. A task generation example appears in Appendix C.

Multi-step tasks in the Approval subset repeat this construction across stages that share a single case identifier. Later stages receive a summary of the upstream specialists' decisions in their user prompt, testing whether the agent can maintain stage independence rather than simply carrying upstream outcomes forward.

4 Evaluation

4.1 Multi-Agent System Formulation

We formulate enterprise multi-agent collaboration as a constrained distributed task execution problem, defined by a four-tuple $(\mathcal{A}, \mathcal{T}, \pi, \mathcal{D})$. The *agent set* $\mathcal{A} = \{a_1, \ldots, a_N\}$ (N=11) is partitioned into operational agents $\mathcal{A}_{\text{op}}$ (8 agents) and approval agents $\mathcal{A}_{\text{appr}}$ (3 agents). The *tool set* $\mathcal{T} = \mathcal{T}_{\text{svc}} \cup \mathcal{T}_{\text{del}}$ consists of enterprise service tools $\mathcal{T}_{\text{svc}}$ distributed across 8 service systems and delegation tools $\mathcal{T}_{\text{del}} = \{t_{\text{del}}^{a_j} : a_j \in \mathcal{A}\}$, where $t_{\text{del}}^{a_j}$ represents the tool for delegating a subtask to agent a_j. The *permission mapping* $\pi \colon \mathcal{A} \to 2^{\mathcal{T}}$ assigns each agent a_i its accessible tool set $\pi(a_i) \subseteq \mathcal{T}$; we write $\pi_{\text{svc}}(a_i) = \pi(a_i) \cap \mathcal{T}_{\text{svc}}$ for the enterprise service tool subset. The *delegation mapping* $\mathcal{D} \colon \mathcal{A} \to 2^{\mathcal{A}}$ specifies the agents to which a_i may delegate. The four-tuple satisfies $\{t_{\text{del}}^{a_j} : a_j \in \mathcal{D}(a_i)\} \subseteq \pi(a_i)$, i.e., each agent's permission set includes the delegation tools for its delegation targets.

Given a natural language instruction q and a starting agent a_{start}, the system produces an execution trajectory $\tau = [e_1, \ldots, e_K]$, where each step $e_k = (a^{(k)}, t^{(k)}, \theta^{(k)})$ denotes agent $a^{(k)}$ invoking tool $t^{(k)} \in \pi(a^{(k)})$ with arguments $\theta^{(k)}$. A step is an enterprise service tool call when $t^{(k)} \in \mathcal{T}_{\text{svc}}$, and a delegation action when $t^{(k)} \in \mathcal{T}_{\text{del}}$. A task q may comprise multiple subtasks $\mathcal{U}(q) = \{u_1, \ldots, u_L\}$, each corresponding to a trajectory segment. We define $\tau|_{a_i, u_l}$ as the subsequence of τ consisting of steps belonging to u_l and executed by a_i, preserving the original order. Evaluation compares τ against a reference trajectory τ^*, supplemented by system state verification.

4.2 Execution Environment

Enterprise service layer. The environment comprises 8 enterprise service systems exposing standardized tool interfaces via the MCP protocol, collectively constituting $\mathcal{T}_{\text{svc}}$. ITSM, HR, CSM, and Gitea serve as core business systems for IT operations, human resources, customer service, and code management respectively; Email, Calendar, Teams, and Drive serve as collaboration systems providing cross-departmental communication and document management capabilities.

Agent layer. The 11 agents in $\mathcal{A}$ are organized into six functional departments (Appendix Table 2). Each agent uses an LLM as its reasoning core and equipped with a role-specific system prompt (see Appendix E.1). Agent a_i has access to $\pi(a_i)$, which includes its authorized enterprise service tools and delegation tools pointing to agents in $\mathcal{D}(a_i)$. When a_i invokes $t_{\text{del}}^{a_j}$, the subtask and its context are sent to a_j, which processes the request independently and returns the result. Delegation may occur recursively, but the chain depth is bounded by a preset limit $d_{\max}$ (see Appendix E.2). Notably, each agent operates as an independent service, engaging in peer-to-peer communication and maintaining an isolated memory throughout the entire task execution lifecycle.

Instantiation of π_{svc}. π_{svc} is realized through two layers. The first layer is *tool visibility*: core business system tools are assigned to agents by functional domain (e.g., $\pi_{\text{svc}}(\texttt{hr\_service\_specialist})$ includes HR tools), collaboration system tools are available to all $a_i \in \mathcal{A}_{\text{op}}$, and approval agents $a_i \in \mathcal{A}_{\text{appr}}$ have $\pi_{\text{svc}}(a_i)$ restricted to local workspace document reading tools only. The second layer is *identity authentication*: each agent holds an independent identity token for each service system, and calls without a valid token are rejected at the server side, ensuring that even accidentally constructed calls to tools $t \notin \pi_{\text{svc}}(a_i)$ cannot succeed.

Data isolation. Before each task execution, an independent database instance is created for each involved service system and seeded with predefined data, establishing a deterministic initial state S_0. Database instances are fully isolated across tasks, ensuring inter-task independence.

4.3 Evaluation Procedure

Each task undergoes four phases. **(1) Initialization:** seed the databases and capture the initial state snapshot S_0. **(2) Execution:** send instruction q to a_{start}, which autonomously invokes tools in $\pi_{\text{svc}}(a_{\text{start}})$ or delegates via $t_{\text{del}}^{a_j}$ to $a_j \in \mathcal{D}(a_{\text{start}})$; downstream agents continue likewise, forming a multi-hop collaboration chain that produces τ. For multi-step tasks, subtasks are executed sequentially; if a preceding subtask fails, all subsequent subtasks are judged as failed. **(3) Evidence collection:** capture the final state snapshot $S_{u_l}^f$ after each subtask u_l and compute the normalized state difference $\Delta(S_{u_l}^0, S_{u_l}^f)$ by comparing record-level changes table by table, filtering non-semantic fields to retain only business state changes; simultaneously collect execution events from all participating agents and merge them into τ. For approval tasks, the external service-state difference is empty by design; the emitted approval decision is recorded as the terminal execution event and compared against the reference outcome. **(4) Cleanup:** destroy the isolated database instances to ensure the next task starts from a clean state.

4.4 Judgment Mechanism

Per-agent judgment. Let $\mathcal{A}(u_l) = \{a_i : \tau^*|_{a_i,u_l} \neq \emptyset\}$ be the set of agents involved in the reference trajectory of subtask u_l. For each $a_i \in \mathcal{A}(u_l)$, the judgment module constructs a three-part input: (1) $\tau^*|_{a_i,u_l}$, the reference steps that a_i should execute within u_l; (2) $\tau|_{a_i,u_l}$, the actual execution events of a_i within u_l; and (3) $\Delta(S_{u_l}^0, S_{u_l}^f)$, the state difference as objective evidence of whether tool calls produced expected side effects. The judge model compares $\tau^*|_{a_i,u_l}$ against $\tau|_{a_i,u_l}$, tolerating equivalent implementations and reasonable ordering differences, but ruling $\text{pass}(a_i, u_l) = \text{false}$ when key actions are missing or tool arguments contradict the state evidence. Approval agents $a_i \in \mathcal{A}_{\text{appr}}$ are judged by comparing the terminal decision event with the expected approval outcome, including the decision label, supporting rule citations, and missing-information flags.

Judgment consistency. To mitigate single-model judgment bias, the system uses a three-model majority vote: Gemini-3.1-Pro, GPT-5.4, and Claude-Sonnet-4.6 independently determine $\text{pass}(a_i, u_l)$, with the final result decided by majority vote. Appendix F reports the consistency between the voting results and human annotations.

Aggregation and metrics. We define subtask-level pass as $\text{pass}(u_l) = \bigwedge_{a \in \mathcal{A}(u_l)} \text{pass}(a, u_l)$, i.e., subtask u_l passes iff all involved agents pass. Let $\mathcal{A}_{\text{eval}} = \bigcup_{u_l \in \mathcal{S}} \{(a, u_l) : a \in \mathcal{A}(u_l)\}$ be all evaluated (agent, subtask) pairs, $\mathcal{S} = \bigcup_{q \in \mathcal{Q}} \mathcal{U}(q)$ be all subtasks, and $\mathcal{Q}$ be all tasks. We report three levels of pass rates: $R_{\text{agent}} = |\{(a, u_l) \in \mathcal{A}_{\text{eval}} : \text{pass}(a, u_l)\}|/|\mathcal{A}_{\text{eval}}|$ measures single-step execution accuracy, $R_{\text{subtask}} = |\{u_l \in \mathcal{S} :$

pass$(u_l)\}|/|\mathcal{S}|$ measures local collaboration success, and $R_{\text{task}} = |\{q \in \mathcal{Q} : \forall\, u_l \in \mathcal{U}(q),\ \text{pass}(u_l)\}|/|\mathcal{Q}|$ measures end-to-end workflow completion.

5 Experiments

5.1 Experimental Setup

We evaluated both closed-source and open-source models, including Claude-Sonnet-4.6 [18], Gemini-3.1-Pro-Preview [19], Gemini-3.1-Flash-Lite-Preview [19], GPT-5.4 [20], GPT-5-mini[20], DeepSeek-V4-Pro [21], DeepSeek-V4-Flash [21], Qwen3.5-122B-A10B [22], Qwen3.5-35B-A3B [22], Qwen3.5-9B [22], MiniMax-M2.7 [23], and MiMo-V2-Flash [24]. We report the overall accuracy, as well as the step-wise and multi-step accuracy for workflow and approval. In addition, we recorded the accuracy of each agent and computed both the average token cost per task and the average token cost for successfully completed tasks. Details of the settings are provided in Appendix E.2.

5.2 Main Results

EntCollabBench remains challenging even for the strongest models. As shown in Table 1, the best overall result is achieved by DeepSeek-V4-Pro with 62.00% average accuracy, followed by DeepSeek-V4-Flash at 57.33% and Claude-Sonnet-4.6 at 52.67%. Most evaluated models remain below 50%, indicating that realistic enterprise multi-agent collaboration is far from solved.

End-to-end collaboration is substantially harder than solving individual subtasks. As shown in Table 1, on multi-step workflow tasks, DeepSeek-V4-Pro reaches 78.33% subtask accuracy but only 50.00% task accuracy, while Claude-Sonnet-4.6 drops from 69.17% to 50.00%. This gap suggests that errors accumulate across delegation chains, where a single routing, execution, or communication failure can cause the full workflow to fail.

Approval tasks are easier in isolation but still difficult in multi-step settings. As shown in Table 1, in single-step approval tasks, DeepSeek-V4-Flash reaches 80.00%, and both Claude-Sonnet-4.6 and DeepSeek-V4-Pro reach 78.75%. However, the best multi-step approval task accuracy is only 40.00%, showing that policy reasoning must be combined with evidence preservation, role-specific judgment, and consistent cross-stage coordination.

Role-level success is much higher than full-task success. Tables 4 and 5 show that strong models often exceed 80% average accuracy at the role level, while their end-to-end task accuracy is much lower. This indicates that the main bottleneck is not isolated tool execution, but cross-agent routing, context transmission, and coordination under permission isolation.

Higher token usage does not necessarily translate into better collaboration. Tables 6 and 7 show that multi-step tasks consume substantially more tokens, especially in the Workflow Track. However, models with larger token usage are not always more accurate, suggesting that successful enterprise collaboration depends more on concise delegation, faithful parameter preservation, and effective planning than on longer context alone.

Table 1. Experimental results of closed-source and open-source models on EntCollabBench. Avg. is the task-level average.

Model	workflow	workflow multi-task		workflow avg.	approval	approval multi-task		approval avg.	avg.
		subtask	task			subtask	task		
Closed-source Models									
Claude-Sonnet-4.6	42.50	69.17	50.00	44.00	78.75	52.73	35.00	70.00	52.67
Gemini-3.1-Pro-Preview	41.88	63.33	45.00	42.50	63.75	49.09	35.00	58.00	47.67
Gemini-3.1-Flash-Lite-Preview	1.88	29.17	2.50	2.00	52.50	41.82	20.00	46.00	16.67
GPT-5.4	40.62	41.67	5.00	33.50	67.50	45.45	30.00	60.00	42.33
GPT-5-mini	19.38	43.33	7.50	17.00	67.50	58.18	40.00	62.00	32.00
Open-source Models									
DeepSeek-V4-Pro	61.25	78.33	50.00	59.00	78.75	41.82	25.00	68.00	62.00
DeepSeek-V4-Flash	52.50	70.00	45.00	51.00	80.00	47.27	30.00	70.00	57.33
Qwen3.5-122B-A10B	30.63	56.67	15.00	27.50	51.25	30.91	20.00	45.00	33.33
Qwen3.5-35B-A3B	25.62	52.50	22.50	25.00	52.50	25.45	5.00	43.00	31.00
Qwen3.5-9B	0.00	0.00	0.00	0.00	27.50	16.36	5.00	23.00	7.67
MiniMax-M2.7	19.38	54.17	15.00	18.50	45.00	20.75	5.00	37.00	24.67
MiMo-V2-Flash	23.75	41.67	12.50	21.50	8.75	0.00	0.00	7.00	16.67

5.3 Further Analysis

To complement aggregate metrics, we analyze representative execution traces to uncover recurring mechanisms behind successful and failed runs. We organize the analysis around enterprise-collaboration behaviors and note model-specific manifestations where especially pronounced.

Role difficulty is strongly affected by position in the delegation chain. The Knowledge Base Specialist is the weakest operational role, but much of this weakness comes from its frequent position at the end of delegation chains. When it is the starting agent, its pass rate is considerably higher; when it is downstream, failures often originate from missing delegation, incomplete content, or incorrect upstream context. By contrast, Developer and QA roles perform better because repository operations are more structured and often occur earlier in the workflow. Details are provided in Appendix G.1.

Multi-step tasks fail through prefix decay and final handoff errors. Multi-step tasks show clear degradation as the chain proceeds. Approval workflows mainly drop at the second step, while workflow tasks often fail at the final step. Details are provided in Appendix G.2. This pattern is especially visible in Qwen3.5-122B-A10B, which often completes earlier subtasks but fails when the last step requires binding previous artifacts, the current role, and the downstream role.

Collaboration tools become bottlenecks during workflow closure. Email, Calendar, and Teams operations often cause failures, despite appearing simpler than core business systems. Agents may misuse sender identities, confuse account keys with email addresses, misformat payloads, or resolve pronouns as literal team/channel names. GPT-5.4 shows this issue most prominently in the `collaboration_ops_specialist` role: many multi-step workflows fail at the final communication stage, contributing to its relatively low accuracy on workflow multi-task. See Appendix G.3.

Delegation failures remain a central source of end-to-end errors. Agents frequently omit required delegation, pass insufficient context, or delegate before prerequisites are ready. Downstream agents may receive tasks without the needed article body, branch, file, or business object, making failure unavoidable even when the downstream role itself is capable. These failures explain why role-level accuracy is much higher than full-task accuracy. Details are provided in Appendix G.4.

Stateful database operations trigger incorrect fallback actions. A recurring Workflow Track failure is that

agents perform a semantically similar but incorrect database operation after failing to locate the target record. For example, when asked to update an existing incident or knowledge article, agents may search with the wrong identifier field, fail to retrieve the record, and then create a new record instead. This behavior appears across multiple models and is especially harmful because it leaves persistent but wrong enterprise state. Details are provided in Appendix G.5.

Tool calls often fail at the parameter-semantics level. Many failures occur after the correct tool family is selected. Models choose wrong enum values, mix up relationship labels such as `applied` and `suggested`, assign work through incorrect user fields, or set the wrong task status/type. These errors show that enterprise tool use requires parameter-level grounding beyond high-level action selection. Details are provided in Appendix G.6.

Higher reliability can require much higher coordination cost. DeepSeek-V4-Pro achieves strong accuracy, but its successful runs often involve many more trace events and substantially higher token usage. The model appears to execute conservatively, repeatedly checking state and coordinating with downstream agents before finalizing actions. This improves robustness but exposes a cost-efficiency trade-off for enterprise collaboration. Details are provided in Appendix G.7.

Approval workflows expose weak decision commitment. In approval tasks, some models retrieve relevant policy evidence but fail to convert it into a final decision. MiMo-V2-Flash shows the most severe form of this behavior, repeatedly reading the same policy documents until token usage grows sharply or the context window is exhausted. This suggests that approval agents need not only retrieval ability, but also stopping criteria and decision discipline. See Appendix G.8 for details.

Small models are much weaker on executable workflow tasks. Small models perform substantially worse on MCP workflow tasks than on approval tasks. Their main bottleneck is executable tool grounding: some stop after listing tools or reading schemas, while others choose plausible tools but fail to produce valid JSON arguments. This suggests that state-changing workflows impose stricter interface and parameter requirements than policy-review tasks. Details are provided in Appendix G.9.

Some failures occur before real tool execution. MiniMax-M2.7 occasionally outputs textual pseudo-tool calls rather than valid executable calls. The model appears to intend an action, but no real tool invocation is recorded in the trace. This reflects weak grounding between natural-language planning and executable tool-use format. See Appendix G.10 for details.

6 Conclusion

We presented EntCollabBench, a benchmark for evaluating enterprise multi-agent collaboration under role specialization, permission isolation, and cross-departmental delegation. Experiments show that current LLM agents often handle local role-specific actions, but struggle with end-to-end workflows requiring routing, context transfer, and final-stage coordination. EntCollabBench provides a reproducible testbed for measuring and improving agents in realistic organizational environments.

References

[1] Tao Yu, Zhengbo Zhang, Zhiheng Lyu, Junhao Gong, Hongzhu Yi, Xinming Wang, Yuxuan Zhou, Jiabing Yang, Ping Nie, Yan Huang, and Wenhu Chen. Browseragent: Building web agents with human-inspired web browsing actions, 2025. URL https://arxiv.org/abs/2510.10666.

[2] Ankush Agarwal, Harsh Vishwakarma, Suraj Nagaje, and Chaitanya Devaguptapu. Enterpriselab: A full-stack platform for developing and deploying agents in enterprises, 2026. URL https://arxiv.org/abs/2603.21630.

[3] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents, 2025. URL https://arxiv.org/abs/2308.03688.

[4] Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. Workarena: How capable are web agents at solving common knowledge work tasks?, 2024. URL https://arxiv.org/abs/2403.07718.

[5] Léo Boisvert, Megh Thakkar, Maxime Gasse, Massimo Caccia, Thibault Le Sellier De Chezelles, Quentin Cappart, Nicolas Chapados, Alexandre Lacoste, and Alexandre Drouin. Workarena++: Towards compositional planning and reasoning-based common knowledge work tasks, 2025. URL https://arxiv.org/abs/2407.05291.

[6] Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z. Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. Theagentcompany: Benchmarking llm agents on consequential real world tasks, 2025. URL https://arxiv.org/abs/2412.14161.

[7] Shiva Krishna Reddy Malay, Shravan Nayak, Jishnu Sethumadhavan Nair, Sagar Davasam, Aman Tiwari, Sathwik Tejaswi Madhusudhan, Sridhar Krishna Nemala, Srinivas Sunkara, and Sai Rajeswar. Enterpriseops-gym: Environments and evaluations for stateful agentic planning and tool use in enterprise settings, 2026. URL https://arxiv.org/abs/2603.13594.

[8] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge, 2019. URL https://arxiv.org/abs/1902.04043.

[9] Constantin Ruhdorfer, Matteo Bortoletto, Anna Penzkofer, and Andreas Bulling. The overcooked generalisation challenge: Evaluating cooperation with novel partners in unknown environments using unsupervised environment design, 2025. URL https://arxiv.org/abs/2406.17949.

[10] Timothy Ossowski, Danyal Maqbool, Jixuan Chen, Zefan Cai, Tyler Bradshaw, and Junjie Hu. Comma: A communicative multimodal multi-agent benchmark, 2025. URL https://arxiv.org/abs/2410.07553.

[11] Ying Mo, Yu Bai, Dapeng Sun, Yuqian Shi, Yukai Miao, Li Chen, and Dan Li. Entworld: A holistic environment and benchmark for verifiable enterprise gui agents, 2026. URL https://arxiv.org/abs/2601.17722.

[12] Haoyu Dong, Pengkun Zhang, Yan Gao, Xuanyu Dong, Yilin Cheng, Mingzhe Lu, Zikun Zhu, Adina Yakefu, and Shuxin Zheng. Finch: Benchmarking finance & accounting across spreadsheet-centric enterprise workflows, 2026. URL https://arxiv.org/abs/2512.13168.

[13] Kung-Hsiang Huang, Akshara Prabhakar, Sidharth Dhawan, Yixin Mao, Huan Wang, Silvio Savarese, Caiming Xiong, Philippe Laban, and Chien-Sheng Wu. Crmarena: Understanding the capacity of llm agents to perform professional crm tasks in realistic environments, 2025. URL https://arxiv.org/abs/2411.02305.

[14] Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. Multiagentbench: Evaluating the collaboration and competition of llm agents, 2025. URL https://arxiv.org/abs/2503.01935.

[15] Yuzhe Zhang, Feiran Liu, Yi Shan, Xinyi Huang, Xin Yang, Yueqi Zhu, Xuxin Cheng, Cao Liu, Ke Zeng, Terry Jingchen Zhang, and Wenyuan Jiang. Silo-bench: A scalable environment for evaluating distributed coordination in multi-agent llm systems, 2026. URL https://arxiv.org/abs/2603.01045.

[16] GitLab Inc. The GitLab Handbook. https://handbook.gitlab.com/, 2026. Accessed May 2026.

[17] European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data (General Data Protection Regulation). Official Journal of the European Union, L 119, 4 May 2016, 2016. URL https://eur-lex.europa.eu/eli/reg/2016/679/oj.

[18] Anthropic. System card: Claude Sonnet 4.6. https://anthropic.com/claude-sonnet-4-6-system-card, 2026.

[19] Google DeepMind. Gemini 3. https://blog.google/products/gemini/gemini-3/, 2025.

[20] Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

[21] DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence. Technical report, 2026. URL https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro/blob/main/DeepSeek_V4.pdf.

[22] Qwen Team. Qwen3.5: Accelerating productivity with native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

[23] MiniMax. MiniMax M2.7: Early echoes of self-evolution. Technical report / blog post, March 2026. URL https://www.minimax.io/news/minimax-m27-en.

[24] Core Team, Bangjun Xiao, Bingquan Xia, Bo Yang, Bofei Gao, Bowen Shen, Chen Zhang, Chenhong He, Chiheng Lou, Fuli Luo, Gang Wang, Gang Xie, Hailin Zhang, Hanglong Lv, Hanyu Li, Heyu Chen, Hongshen Xu, Houbin Zhang, Huaqiu Liu, Jiangshan Duo, Jianyu Wei, Jiebao Xiao, Jinhao Dong, Jun Shi, Junhao Hu, Kainan Bao, Kang Zhou, Lei Li, Liang Zhao, Linghao Zhang, Peidian Li, Qianli Chen, Shaohui Liu, Shihua Yu, Shijie Cao, Shimao Chen, Shouqiu Yu, Shuo Liu, Tianling Zhou, Weijiang Su, Weikun Wang, Wenhan Ma, Xiangwei Deng, Bohan Mao, Bowen Ye, Can Cai, Chenghua Wang, Chengxuan Zhu, Chong Ma, Chun Chen, Chunan Li, Dawei Zhu, Deshan Xiao, Dong Zhang, Duo Zhang, Fangyue Liu, Feiyu Yang, Fengyuan Shi, Guoan Wang, Hao Tian, Hao Wu, Heng Qu, Hongfei Yi, Hongxu An, Hongyi Guan, Xing Zhang, Yifan Song, Yihan Yan, Yihao Zhao, Yingchun Lai, Yizhao Gao, Yu Cheng, Yuanyuan Tian, Yudong Wang, Zhen Tang, Zhengju Tang, Zhengtao Wen, Zhichao Song, Zhixian Zheng, Zihan Jiang, Jian Wen, Jiarui Sun, Jiawei Li, Jinlong Xue, Jun Xia, Kai Fang, Menghang Zhu, Nuo Chen, Qian Tu, Qihao Zhang, Qiying Wang, Rang Li, Rui Ma, Shaolei Zhang, Shengfan Wang, Shicheng Li, Shuhao Gu, Shuhuai Ren, Sirui Deng, Tao Guo, Tianyang Lu, Weiji Zhuang, Weikang Zhang, Weimin Xiong, Wenshan Huang, Wenyu Yang, Xin Zhang, Xing Yong, Xu Wang, Xueyang Xie, Yilin Jiang, Yixin Yang, Yongzhe He, Yu Tu, Yuanliang Dong, Yuchen Liu, Yue Ma, Yue Yu, Yuxing Xiang, Zhaojun Huang, Zhenru Lin, Zhipeng Xu, Zhiyang Chen, Zhonghua Deng, Zihan Zhang, and Zihao Yue. Mimo-v2-flash technical report, 2026. URL https://arxiv.org/abs/2601.02780.

Appendix

A Agent Roster

Table 2 lists all eleven agents in the EntCollabBench organization, grouped by department. Each row reports the agent identifier used in task specifications and ground-truth trajectories, the persona name surfaced to the agent's own system prompt, and the agent's dedicated service scope. The eight operational agents additionally share four common collaboration services (Teams, Email, Calendar, Drive) which are not repeated per row. Cross-agent handoffs are issued through the typed delegation primitive `ask_<agent>_by_http`, where `<agent>` is the target identifier in this table.

Department	Agent identifier	Persona	Dedicated services
IT	`it_service_desk_l1`	Ivan Park	ITSM
	`it_change_engineer`	Nina Patel	ITSM
Human Resources	`hr_service_specialist`	Helen Zhou	HR
Customer Service	`customer_support_specialist`	Carlos Mendez	CSM
Shared Services	`knowledge_base_specialist`	Priya Nair	ITSM, HR, CSM (kb tools)
	`collaboration_ops_specialist`	Olivia Chen	— (common tools only)
Engineering	`developer_engineer`	Ethan Walker	Gitea
	`qa_test_engineer`	Mia Kim	Gitea
Approval Center	`finance_approval_specialist`	Sophia Lin	local workspace docs
	`legal_approval_specialist`	Daniel Wu	local workspace docs
	`procurement_approval_specialist`	Grace Liu	local workspace docs

Table 2. Full EntCollabBench agent roster.

B Policy Schema Example

To make the policy schema of Section 3.3.3 concrete, we show one finalized rule end to end. The rule below is extracted from the GitLab Handbook by the four-stage pipeline (chunker → classifier → extractor → finalize) and is one of the two target rules used by the task example in Appendix C.

Example: Rule `LEG-EVAL-0001` (*Evaluation Agreement Approval*)

Identity.

- `rule_id`: `LEG-EVAL-0001`
- `primary_role`: `legal`
- `category`: `evaluation_agreement_approval`
- `severity`: `mandatory`

Conditions (conjunctive predicates over typed fields; all must hold for the rule to fire):

- `customer_declines_trial_process == true`
- `requested_agreement_type == "Evaluation Agreement"`

Decision and approver chain.

- `decision`: `require_preapproval`
- `approver_chain`: `["Area Sales Manager or higher"]`
- `fulfillment_evidence`: `[area_sales_manager_or_higher_preapproval]`

The decision is `require_preapproval` by default; the rule engine promotes it to `approve` when an evidence document keyed by the listed slug is present in the case submission.
Cross-domain links (added by the finalize stage):

- `cross_refs`: `[PROC-AGREE-0001]`

The link records that this legal rule *references* a procurement rule on related agreement handling, computed by an LLM pass over rule pairs that share categories or fields.
Source citation (verbatim substring of the original chunk; non-substring extractions are dropped during finalize):

- `source.doc_id`: `gitlab/legal_customer_negotiations`
- `source.section`: "Sales Guide | Collaborating with GitLab Legal > OPERATIONAL > Request for a Trial or Evaluation Agreement"
- `source.verbatim`:
  "If a customer declines the trial process and is adamant to have a separate Evaluation Agreement, the sales team member or solutions architect should: Open a Legal Request to request an Evaluation Agreement with Request Form. The Legal Request should (i) include a request for approval from the Area Sales Manager or higher; and (ii) set forth applicable details to complete the Request Form, such as customer contact information, length of evaluation, number of users, etc."

The schema-level guarantees illustrated by this rule — typed conjunctive predicates, an enumerated decision class, named approver chain, evidence slugs, finalized cross-domain links, and a verbatim-grounded citation — are uniform across all 290 rules in the corpus, so any sampled subset can be fed to the deterministic decision engine without rule-specific glue.

C Approval Task Example

We walk through one approval task end to end to show how target-rule sampling, distractor injection, case construction, submission rendering, and rule-engine ground truth fit together. The example is task `T-0001` (`case_id`: `CONT-2026-0001`), a single-step cross-domain case that fires one legal rule and one procurement rule simultaneously.

Step 1 — Target rule sampling. The synthesis pipeline samples two target rules drawn from different roles, exercising the cross-departmental adjudication perturbation:

Sampled target rules

- `LEG-EVAL-0001` — category *evaluation_agreement_approval*, role `legal`, decision `require_preapproval`. Predicates:
  - `customer_declines_trial_process` = `true`
  - `requested_agreement_type` = `"Evaluation Agreement"`

  Fulfilled by `area_sales_manager_or_higher_preapproval`. Full definition in Appendix B.
- `PROC-BGSCRN-0007` — category *contractor_background_screening*, role `procurement`, decision `require_docs`. Predicates:
  - `worker_type` = `"contingent_worker"`

- `returning_to_service_at_gitlab` = `true`
- `days_since_contract_completion` ≤ 90

Fulfilled by `contractor_background_screening`.

Step 2 — Case construction. Field values are reverse-engineered from the predicates of all target rules so that every condition is satisfied; remaining business fields (project name, applicant department, application date) are filled from a fixture pool.

Constructed case parameters (`case_id`: `CONT-2026-0001`)

- `worker_type` = `"contingent_worker"`
- `returning_to_service_at_gitlab` = `true`
- `days_since_contract_completion` = 90
- `customer_declines_trial_process` = `true`
- `requested_agreement_type` = `"Evaluation Agreement"`
- `extension_beyond_initial_term` = `true`

Business header: `project_name` = `"Horizon Marketing Refresh"`; `applicant_department` = `"Infrastructure Operations"`; `application_date` = `"2026-05-03"`.

Step 3 — Distractor injection. Distractor rules from the same legal/procurement neighborhood are added to tighten the read-time decision boundary. Their predicates are pinned at near-miss values so they almost fire on the case but ultimately do not, leaving the ground truth unchanged:

Distractor rules (do not fire)

- `PROC-CW-0001` — category *contractor_extension_approval*.
  - Rule requires: `worker_type` = `"staff_augmentation_worker"`
  - Case has: `worker_type` = `"contingent_worker"`
  - Near-miss: the rule misses on `worker_type`, although the case's `extension_beyond_initial_term` = `true` would otherwise align with a later predicate.
- `PROC-CW-0002` — variant of the same category with the same near-miss on `worker_type`.

Step 4 — Submission rendering. The synthesizer renders the case into a self-contained submission package: a shared intake form, one evidence document per fulfilled rule, and a role-specific directive that names which specialists must adjudicate which sub-review.

Submission artifacts (excerpts)

(a) Approval intake form (excerpt).

"Approval Intake Form for case_id CONT-2026-0001. Project Name: Horizon Marketing Refresh. Applicant Department: Infrastructure Operations. Application Date: 2026-05-03. The submission flags Contractor Background Screening as the operative review area…
Submitted parameters: worker_type = contingent_worker; returning_to_service_at_gitlab = True; days_since_contract_completion = 90; customer_declines_trial_process = True; requested_agreement_type =

Evaluation Agreement; extension_beyond_initial_term = True."

(b) Pre-approval evidence (`LEG-EVAL-0001` fulfillment).

From: Area Sales Manager or higher. To: Infrastructure Operations Intake. Subject: Evaluation Agreement Approval — pre-approval — CONT-2026-0001.
"As Area Sales Manager or higher within the legal domain for the cited control I pre-approve this routing on the documented parameters."

(c) Background-check evidence (`PROC-BGSCRN-0007` fulfillment).

Subject: contractor background screening — completed for the contractor background screening workflow on Horizon Marketing Refresh.
"This report satisfies the documentation requirement keyed by contractor_background_screening for the routed approval."

(d) Directive (delegation prompt to the orchestrator).

"case_id CONT-2026-0001. Project Horizon Marketing Refresh is submitted by Infrastructure Operations on 2026-05-03…
First, ask legal_approval_specialist to perform Contractor Background Screening — Legal & Regulatory Review; the review should read the intake form, the background-check report, the pre-approval email, and the legal material-review policy document…
Then, ask procurement_approval_specialist to perform Contractor Background Screening — Procurement & Vendor Review; the review should read the same evidence and the procurement background-screening policy document…"

Step 5 — Ground-truth computation. The deterministic decision engine evaluates the case against the full policy schema. Two rules fire (one per role); both have their fulfillment evidence present in the submission, so the engine promotes `require_preapproval / require_docs` to `approve`. The finance specialist has no firing rule and resolves to `not_applicable`.

Ground truth (`ground_truth_approval_results`)

- `finance_approval_specialist`
  - Decision: `not_applicable`
  - Citations: `[]`
  - Reason: no finance rule is implicated by the case parameters.
- `legal_approval_specialist`
  - Decision: `approve`
  - Citations: `[LEG-EVAL-0001]`
  - Reason: the rule fires on the case parameters; the pre-approval evidence keyed by `area_sales_manager_or_higher_preapproval` discharges the pre-approval requirement, promoting the rule from `require_preapproval` to `approve`.
- `procurement_approval_specialist`
  - Decision: `approve`
  - Citations: `[PROC-BGSCRN-0007]`
  - Reason: the rule fires on the case parameters; the background-check evidence keyed by `contractor_background_screening` discharges the document requirement, promoting the rule from `require_docs` to `approve`.

Step 6 — Final task record. The five steps above are serialized into a single structured record that is appended to the released `tasks.json` file. The abridged record below shows the fields actually consumed by the runtime: the input parameters, the metadata used by analyses (e.g. which rules fire and which are distractors), the rendered submission package, and the rule-engine ground truth. Long string fields (e.g. `user_prompt`, per-specialist `rationale`) are elided because they appear verbatim in earlier steps.

Released task record (`tasks.json`, abridged)


```
{
  "task_id": "T-0001",
  "type": "contractor_background_screening",
  "description": "Contractor Background Screening review for
     Infrastructure Operations (case_id CONT-2026-0001).",
  "task_list": [{
    "sub_index": 1,
    "begin_agent": "approval_orchestrator",
    "user_prompt": "(see Step 4(d) above)",
    "request_fields": {
      "worker_type": "contingent_worker",
      "returning_to_service_at_gitlab": true,
      "days_since_contract_completion": 90,
      "customer_declines_trial_process": true,
      "requested_agreement_type": "Evaluation Agreement",
      "extension_beyond_initial_term": true
    },
    "satisfied_rule_ids": ["LEG-EVAL-0001", "PROC-BGSCRN-0007"],
    "distractor_rule_ids": ["PROC-CW-0001", "PROC-CW-0002"],
    "involved_roles": ["legal", "procurement"],
    "submission_package": [
      {"evidence_id": "E-001", "kind": "form"},
      {"evidence_id": "E-002", "kind": "background_check"},
      {"evidence_id": "E-003", "kind": "preapproval_email"}
    ],
    "ground_truth_approval_results": {
      "finance_approval_specialist":
        {"decision": "not_applicable", "rule_citations": []},
      "legal_approval_specialist":
        {"decision": "approve", "rule_citations": ["LEG-EVAL-0001"]},
      "procurement_approval_specialist":
        {"decision": "approve", "rule_citations": ["PROC-BGSCRN-0007"]}
    }
  }]
}
```


What the agent must do. The agent must (i) recognize that `LEG-EVAL-0001` and `PROC-BGSCRN-0007` are the two firing rules and that `PROC-CW-0001` / `PROC-CW-0002` are near-miss distractors that do not apply, (ii) verify that the corresponding fulfillment evidence is present in the submission package, and (iii) emit per-specialist decisions with rule citations matching the engine's ground truth. The finance specialist must correctly emit `not_applicable` with empty citations, since no finance rule is implicated.

D Workflow Task Template Example

The Workflow subset is generated from a fixed library of 20 domain templates, one per business domain. Each template is a Python function that takes a render context (trigger key, governance key, dates, timezone, fixture indices) and emits a `TaskDraft` containing (i) a parameter-faithful natural-language instruction (in English, with a Chinese translation), and (ii) an ordered tool sequence over typed enterprise services that

drives the ground-truth trajectory and the verifier's expected state changes. Each template is invoked under the cross-product of 5 triggers and 5 governance rules, yielding up to 500 raw task drafts per generation pass before deduplication and quality filtering.

We illustrate the structure with one template, `build_employee_onboarding_provisioning`, which renders an HR-led onboarding case that crosses People Ops, IT Service Desk, the hiring department, and finance shared services.

Example template: employee onboarding provisioning

Trigger fixtures (one of five sampled by trigger_index):

- *monitoring_alert* — "The onboarding readiness dashboard on ⟨trigger_date⟩ shows that the ⟨role⟩ start package for the ⟨office⟩ is missing identity and equipment tasks."
- *employee_request* — "The hiring coordination mailbox asked on ⟨trigger_date⟩ to accelerate onboarding for the ⟨role⟩ joining the ⟨department⟩ team in the ⟨office⟩."
- *audit_finding* — "An internal onboarding audit opened on ⟨trigger_date⟩ found that the approved ⟨role⟩ requisition for the ⟨office⟩ has no complete cross-functional provisioning trail."
- *contract_date_node* — start-milestone variant keyed to action_date.
- *external_partner_event* — payroll-feed handoff variant.

Governance fixtures (one of five sampled by governance_index):

- *sla_deadline* — the day-one provisioning package must close by the action date.
- *budget_cost* — Finance Controller sign-off above $3,200.
- *approval_chain* — People Ops Director and IT Operations Manager.
- *compliance_legal* — new-joiner access segregation.
- *continuity_min_disruption* — preserve the team's start-of-quarter operations.

Parameterized roles, departments, and offices (drawn from per-template fixture pools):

- Roles: APAC Revenue Operations Analyst; EMEA Support Enablement Specialist; North America Procurement Coordinator; Shared Services Payroll Analyst; Cloud Operations Planner.
- Departments: Revenue Operations; Customer Support; Procurement; Finance Shared Services; Cloud Operations.
- Offices: New York / Los Angeles / London / Shanghai / Singapore Hub.

Tool sequence (the structured trajectory; each step is parameter-checked against the seed database and is owned by a specific MCP server):

- `frappe.get_job_openings`
- `eops_hr.list_hr_cases`
- `eops_hr.create_new_hr_case` (or `update_hr_case` if the requisition already has an open case)
- `eops_itsm.create_incident`
- `eops_teams.create_group`
- `eops_email.send_email` *or* `eops_teams.send_message` (chosen by context)

Rendered instruction (one filled instance, after the parameter-faithful skeleton is rewritten by GPT-5.5 into stakeholder voice):

> *"The hiring coordination mailbox asked on 2026-05-04 to accelerate onboarding for the APAC Revenue Operations Analyst joining the Cloud Operations team in the Shanghai Hub. First, query the approved job opening for the analyst and the existing HR case queue for the Shanghai Hub onboarding stream. If an onboarding HR case is already open for the same requisition, update it so the access, laptop, and orientation checklist stay in one record; otherwise create a new onboarding HR case tagged to Cloud Operations. Then create a high-priority IT incident for workstation, identity, and collaboration provisioning, and create a Teams working group named onboarding-2026-05-04 for People Ops, IT Service Desk, and Cloud Operations. Send an onboarding handoff plan to it-ops@corp.example and hiring-ops@corp.example, then route the HR case and communication draft to the IT Service Desk so the day-one provisioning package closes within the agreed SLA."*

The same skeleton recurs across all 20 templates: a domain-specific scenario rooted in a triggering event, a parameterized cast of roles / offices / departments drawn from fixture pools, a governance clause that encodes the controlling deadline / budget / approver chain, and an ordered enterprise-service tool sequence whose typed arguments form the ground-truth trajectory. Because both the instruction values and the tool arguments are produced from the same render context, the verifier can mechanically check that every argument in the trajectory is recoverable from the user instruction.

E Agent-Layer and Benchmark Implementation Details

E.1 Agent Inference System Prompts

We abstract the 11 system prompts into two prompt templates. The first template is shared by the eight operational agents: `it_service_desk_l1`, `it_change_engineer`, `hr_service_specialist`, `customer_support_specialist`, `knowledge_base_specialist`, `collaboration_ops_specialist`, `developer_engineer`, and `qa_test_engineer`. The second template is shared by the three approval agents: `finance_approval_specialist`, `legal_approval_specialist`, and `procurement_approval_specialist`.

Operational Agent Prompt Template

```
# IDENTITY
- Role: <ROLE>
- Name: <NAME>
- Email: <EMAIL>

# ENVIRONMENT & ACCESS
- Environment: Corporate environment with strict access control limits.
- Common Services: teams, email, calendar, drive
- Dedicated Services: <DEDICATED_SERVICES>
- Authentication: Auth tokens are attached automatically. Do not pass them explicitly.

# RESPONSIBILITIES
- <RESPONSIBILITY>

# EXECUTION RULES
1. No Permission Required: You do not need to ask for confirmation or permission before
taking action. Do not ask for any information or confirmation from the user.
2. Task Delegation: Prioritize completing tasks independently. If a task exceeds your
responsibilities, use the ask_{role_name}_by_http tool to delegate only the out-of-scope
portion. Provide all required context.
3. Approval Request: For approval tasks, call the designated approver via
ask_{role_name}_by_http. Start only one approval item at a time, and initiate each
approval item only once. When initiating an approval request, include all already-provided
self-information required for that approval. As soon as all approval item results are
received, end immediately and return.
4. Error Fallback: If a colleague returns an error/timeout and you cannot provide
additional information to resolve it, do not ask again. Proceed to the next step. If you
cannot proceed, immediately summarize and return.
5. No Blind Repetition: If a task within your scope genuinely cannot be completed (for
example, a mandatory field is missing), immediately report the error and stop. Do not
repeat similar steps blindly.
6. Strict Return Format: Your final response must strictly follow this exact format:
DONE:<completed content>, UNDONE:<uncompleted content>, ERROR:<error message>.

# OPTIONAL ROLE-SPECIFIC GUIDE
- <OPTIONAL_ROLE_SPECIFIC_GUIDE>
```

The placeholder fields are instantiated as follows:

- `it_service_desk_l1`: Role = "IT Service Desk L1 Engineer"; Dedicated Services = `itsm`; Responsibility = "Use itsm for ticket handling."
- `it_change_engineer`: Role = "IT Problem/Change Management Engineer"; Dedicated Services = `itsm`; Responsibility = "Use itsm for change and incident workflows."
- `hr_service_specialist`: Role = "HR Service Specialist"; Dedicated Services = `hr`; Responsibility = "Use hr for HR operations."
- `customer_support_specialist`: Role = "Customer Support Specialist (CSM Case Agent)"; Dedicated Services = `csm`; Responsibility = "Use csm for customer support operations."
- `knowledge_base_specialist`: Role = "Knowledge Base Specialist (Cross ITSM/HR/CSM)"; Dedicated Services = `itsm`, `hr`, `csm` (primarily for knowledge entry maintenance); Responsibility = "Use itsm, hr, and csm for knowledge entry maintenance across ITSM/HR/CSM."
- `collaboration_ops_specialist`: Role = "Collaboration Operations Specialist (Meetings/Emails/Docs/Team Spaces)"; Dedicated Services = "None (Common services only)"; Responsibility = "Use common MCP tools for collaboration operations."; Optional guide = "email `send_message`: always set `payload.filename`."
- `developer_engineer`: Role = "Developer Engineer (Entry-level)"; Dedicated Services = `gitea`; Responsibility = "Use gitea for software collaboration."
- `qa_test_engineer`: Role = "QA/Test Engineer (Entry-level)"; Dedicated Services = `gitea`; Responsibility = "Use gitea for QA collaboration and code review tasks."

Approval Agent Prompt Template

```
# ROLE
<APPROVAL_ROLE>.

# RESPONSIBILITIES
- Review and approve <DOMAIN> requests by reading local workspace documents only.

# DOCUMENT ACCESS CONSTRAINT
Workspace layout reference:
- policy_docs/:  policy files for rule lookup.
- submission/:  case submission files (for example submission/T-0001/...).
- rulebook.md:  role rulebook at workspace root.

Do not request, browse, or infer from any submission file outside the explicitly requested
list.
Every decision and rationale must be strictly grounded in the allowed documents and cited
rules.
You must complete policy evaluation and submission evaluation before responding.

# DECISION POLICY
Choose exactly one decision based on document evidence and applicable rules:
- approve:  requirements are met.
- reject:  explicit prohibition or disqualifying condition is met.
- require_preapproval:  rule requires prior approval/escalation not yet fulfilled.
- require_docs:  mandatory evidence/documents are missing.
- out of scope:  no <DOMAIN>-applicable rule can be determined from provided materials.
- error:  technical/read failure.

# RATIONALE
```

Hyperparameter	Value
Inter-agent HTTP timeout	400 s
Task timeout	1000 s
Task LLM temperature	0
Maximum recursion depth $d_{\max}$	3
Summarization enabled	True
Summary trigger tokens	50,000
Judge LLM temperature	0

Table 3. Main hyperparameters used in the agent layer and benchmark evaluation.

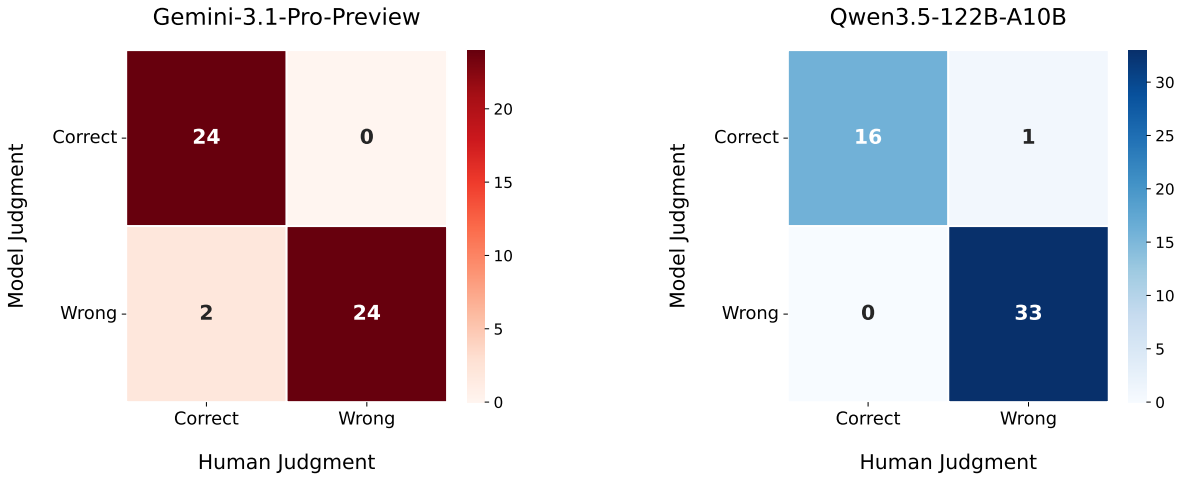


Figure 4. Confusion Matrix of Consistency Between Model Evaluation and Human Judgments

```
Provide concise rule-grounded reasoning aligned with the selected decision.
If require_docs, name missing documents.
If require_preapproval, name required approver/escalation path when available.

# OUTPUT FORMAT
DECISION:<approve|reject|require_preapproval|require_docs|out of scope|error>,
RATIONALE:<reason>, ERROR:<error message>
```

The placeholder fields are instantiated as follows:

- `finance_approval_specialist`: <*APPROVAL_ROLE*> = “Finance Approval Specialist”; <*DOMAIN*> = “finance”.
- `legal_approval_specialist`: <*APPROVAL_ROLE*> = “Legal Approval Specialist”; <*DOMAIN*> = “legal”.
- `procurement_approval_specialist`: <*APPROVAL_ROLE*> = “Procurement Approval Specialist”; <*DOMAIN*> = “procurement”.

Table 4. Experimental results of closed-source and open-source models on workflow and approval across 11 agents. Agent abbreviations in the header correspond to the full identifiers as follows: IT (IT Service Desk L1), Change (IT Change Engineer), HR (HR Service Specialist), Support (Customer Support Specialist), KB (Knowledge Base Specialist), Collab (Collaboration Ops Specialist), Dev (Developer Engineer), QA (QA Test Engineer), Finance (Finance Approval Specialist), Legal (Legal Approval Specialist), and Proc. (Procurement Approval Specialist).

Model	IT	Change	HR	Support	KB	Collab	Dev	QA	Finance	Legal	Proc.	Avg.
Closed-source Models												
Claude-Sonnet-4.6	74.40	96.97	79.71	86.21	67.33	91.24	97.62	80.95	88.46	88.68	77.27	83.09
Gemini-3.1-Pro-Preview	87.90	89.23	94.12	92.98	78.79	51.85	100.00	100.00	73.08	71.70	68.18	80.08
Gemini-3.1-Flash-Lite-Preview	22.40	39.39	21.74	15.52	42.57	35.77	57.14	66.67	53.85	67.92	59.09	39.06
GPT-5.4	74.40	90.91	92.75	79.31	55.45	83.94	92.86	88.10	84.62	81.13	72.73	79.55
GPT-5-mini	59.68	72.73	79.71	67.24	30.69	50.74	80.49	90.24	73.08	73.58	77.27	62.98
Open-source Models												
DeepSeek-V4-Pro	88.00	89.39	94.20	94.83	66.34	94.16	100.00	95.24	88.46	79.25	88.64	87.93
DeepSeek-V4-Flash	79.03	87.88	95.65	91.38	58.42	94.12	100.00	95.12	84.62	84.91	88.64	85.38
Qwen3.5-35B-A3B	64.80	72.73	78.26	86.21	45.54	67.15	95.24	83.33	65.38	67.92	52.27	68.42
Qwen3.5-122B-A10B	54.40	69.70	71.01	72.41	43.56	82.48	88.10	95.24	73.08	62.26	43.18	66.84
Qwen3.5-9B	–	–	–	–	–	–	–	–	42.31	35.85	45.45	40.65
MiMo-V2-Flash	63.20	76.92	66.67	59.65	35.00	83.82	80.95	90.48	80.00	30.00	16.67	58.48
MiniMax-M2.7	60.00	63.64	42.03	67.24	26.73	53.28	88.10	69.05	61.54	54.72	41.86	54.33

E.2 Hyperparameters

We report the main hyperparameters that govern agent execution and benchmark evaluation. All task-execution agents use deterministic decoding with temperature 0, and recursive delegation is bounded by a maximum depth of $d_{\max} = 3$. Inter-agent communication uses an HTTP timeout of 400 seconds, while each top-level task is assigned a timeout budget of 1000 seconds. Short-term memory summarization is enabled, with summarization triggered once the running context reaches 50,000 tokens. For evaluation, the judge LLM also uses deterministic decoding with temperature 0.

F Consistency Between Model Evaluation and Human Judgments

To assess the reliability of the automatic judgment mechanism, we compare the three-model majority-vote results with human annotations. As shown in Figure 4, the automatic judgments are highly consistent with human judgments on both evaluated models. For Gemini-3.1-Pro-Preview, the majority-vote judge agrees with human annotations in 48 out of 50 cases, achieving a 96.0% agreement rate. For Qwen3.5-122B-A10B, the agreement reaches 49 out of 50 cases, corresponding to 98.0%. These results indicate that the three-model majority voting mechanism provides judgments that are well aligned with human evaluation.

G Further Analysis

Table 5. Experimental results of closed-source and open-source models on workflow multi-task and approval multi-task across 11 agents. Agent abbreviations in the header correspond to the full identifiers as follows: IT (IT Service Desk L1), Change (IT Change Engineer), HR (HR Service Specialist), Support (Customer Support Specialist), KB (Knowledge Base Specialist), Collab (Collaboration Ops Specialist), Dev (Developer Engineer), QA (QA Test Engineer), Finance (Finance Approval Specialist), Legal (Legal Approval Specialist), and Proc. (Procurement Approval Specialist).

Model	IT	Change	HR	Support	KB	Collab	Dev	QA	Finance	Legal	Proc.	Avg.
Closed-source Models												
Claude-Sonnet-4.6	86.36	100.00	92.86	100.00	70.59	83.87	91.67	53.85	33.33	81.25	52.94	79.64
Gemini-3.1-Pro-Preview	94.44	100.00	55.56	100.00	84.62	56.67	85.71	92.31	66.67	75.00	44.44	76.10
Gemini-3.1-Flash-Lite-Preview	35.71	66.67	66.67	40.00	30.00	45.00	50.00	66.67	33.33	68.75	44.44	49.19
GPT-5-mini	71.43	100.00	100.00	85.71	50.00	50.00	76.92	66.67	66.67	64.71	64.71	67.91
GPT-5.4	68.42	80.00	83.33	100.00	15.38	35.48	85.71	92.31	50.00	76.92	47.06	60.84
Open-source Models												
DeepSeek-V4-Pro	90.00	83.33	100.00	81.82	66.67	78.12	100.00	93.33	50.00	58.82	43.75	79.12
DeepSeek-V4-Flash	86.96	83.33	100.00	92.31	82.35	75.86	92.31	64.29	100.00	60.00	47.06	77.98
MiMo-V2-Flash	80.00	100.00	33.33	66.67	42.86	80.00	90.91	100.00	–	0.00	0.00	75.68
Qwen3.5-35B-A3B	90.00	50.00	66.67	62.50	27.27	67.74	92.86	100.00	0.00	46.67	27.78	63.58
Qwen3.5-122B-A10B	68.18	90.91	66.67	75.00	35.29	58.06	78.57	75.00	100.00	50.00	33.33	61.21
Qwen3.5-9B	–	–	–	–	–	–	–	–	0.00	21.43	23.53	21.88
MiniMax-M2.7	52.38	100.00	36.36	81.82	53.33	67.86	71.43	83.33	0.00	46.15	20.00	59.48

Table 6. Input/Output token statistics of successful tasks across models.

Model	Input (K)					Output (K)				
	workflow	workflow multi-task	approval	approval multi-task	Avg.	workflow	workflow multi-task	approval	approval multi-task	Avg.
Closed-source Models										
Claude-Sonnet-4.6	318.36	2804.86	104.33	462.53	667.28	6.69	37.09	3.00	9.95	10.20
Gemini-3.1-Pro-Preview	204.86	1358.85	51.26	166.64	358.36	6.67	17.68	4.32	8.69	7.68
Gemini-3.1-Flash-Lite-Preview	248.43	1965.17	56.47	240.29	462.89	2.16	4.46	0.72	1.91	1.96
GPT-5.4	183.77	239.18	50.04	328.49	148.20	2.60	4.22	0.90	3.03	2.25
GPT-5-mini	500.27	1678.73	50.16	254.45	544.55	10.18	28.74	4.45	13.23	11.55
Open-source Models										
DeepSeek-V4-Pro	573.89	3587.46	147.72	472.74	801.01	5.91	25.61	2.37	7.45	7.69
DeepSeek-V4-Flash	783.06	5024.39	205.86	564.95	1131.73	6.52	21.67	3.07	8.63	7.68
Qwen3.5-122B-A10B	352.91	1968.20	56.85	209.80	464.32	4.85	21.56	1.78	8.90	6.33
Qwen3.5-35B-A3B	701.57	2310.42	46.70	202.07	626.02	6.58	28.78	2.44	11.67	8.04
Qwen3.5-9B	–	–	29.95	23.24	28.61	–	–	2.17	1.92	2.12
MiniMax-M2.7	454.26	1796.93	24.99	211.72	424.76	4.98	17.98	2.38	10.38	6.35
MiMo-V2-Flash	275.17	2851.05	261.66	–	581.35	2.69	25.30	0.85	–	5.53

G.1 Role difficulty is strongly affected by position in the delegation chain.

Case Study: Role Specialization and Chain-Position Effects

Overview. A consistent cross-model pattern is that agent performance is shaped not only by the difficulty of the assigned operation itself, but also by the agent's *position in the delegation chain*. In particular, the weakest-looking role in aggregate statistics is often `knowledge_base_specialist`, while code-oriented roles such as `developer_engineer` and `qa_test_engineer` are comparatively more reliable.

Knowledge Base Specialist: low average performance. `Knowledge_base_specialist` appears to be the weakest role. This is unsurprising at the task level: knowledge-base maintenance often requires multi-step database operations across several business systems, including article updates, metadata synchronization, and link creation between cases/incidents and knowledge records. These actions are operationally tedious and parameter-sensitive.

However, the more important finding is that its weak aggregate score is not purely a role-capability issue. It is

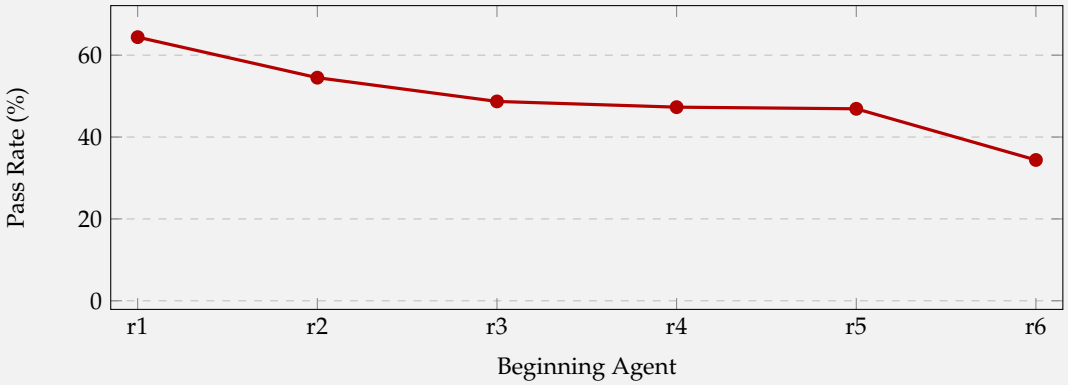
Table 7. Input/Output token statistics of all tasks across models.

Model	Input (K)					Output (K)				
	workflow	workflow multi-task	approval	approval multi-task	**Avg.**	workflow	workflow multi-task	approval	approval multi-task	**Avg.**
Closed-source Models										
Claude-Sonnet-4.6	304.99	2325.60	112.13	324.45	568.52	6.67	32.20	3.23	7.66	8.28
Gemini-3.1-Pro-Preview	210.85	1232.00	54.01	133.98	282.16	6.50	17.48	4.60	7.78	7.00
Gemini-3.1-Flash-Lite-Preview	437.77	2084.72	62.99	169.86	549.95	2.81	3.71	0.76	1.45	2.36
GPT-5.4	209.02	403.28	52.67	180.55	182.59	2.65	3.65	0.97	1.85	2.21
GPT-5-mini	646.23	1219.93	46.87	195.00	483.76	14.55	26.34	4.75	10.78	12.75
Open-source Models										
DeepSeek-V4-Pro	743.15	3992.73	165.05	339.86	1007.03	6.09	27.75	2.65	5.47	6.82
DeepSeek-V4-Flash	958.29	3639.38	200.68	467.57	1102.81	7.06	18.69	3.34	6.82	7.09
Qwen3.5-122B-A10B	472.71	1047.90	64.21	139.18	389.50	5.09	11.23	2.43	5.58	4.84
Qwen3.5-35B-A3B	802.58	2080.27	47.82	90.56	678.28	8.81	20.21	3.18	9.60	8.02
Qwen3.5-9B	–	–	31.76	44.65	34.33	–	–	2.06	4.00	2.45
MiniMax-M2.7	615.93	1313.39	24.20	56.02	455.27	5.35	11.53	2.36	3.56	4.71
MiMo-V2-Flash	754.61	4622.82	1854.55	3363.12	1484.42	3.39	21.27	1.22	1.83	4.34

also strongly affected by a chain-position effect. In many workflows, `knowledge_base_specialist` appears at the end of the delegation chain, which means its success depends heavily on whether upstream agents correctly preserve and relay the remaining instructions.
Across workflow tasks, the pass rate of `knowledge_base_specialist` varies substantially depending on the `beginning_agent`:

Beginning agent	Passed / Total	Pass rate
`knowledge_base_specialist (r1)`	163 / 253	64.4%
`it_change_engineer (r2)`	78 / 143	54.5%
`collaboration_ops_specialist (r3)`	37 / 76	48.7%
`customer_support_specialist (r4)`	114 / 241	47.3%
`it_service_desk_l1 (r5)`	98 / 209	46.9%
`hr_service_specialist (r6)`	64 / 186	34.4%

The following figure can be used to visualize the chain-position effect:



This distribution supports a more precise interpretation:

1. When `knowledge_base_specialist` is the starting agent (`beginning_agent`), its pass rate reaches **64.4%**, which is not exceptionally poor.
2. Its average performance is pulled down because it frequently appears as a *downstream* agent.

3. In these downstream cases, failures often originate *before* the specialist acts: upstream agents omit the handoff, pass incomplete instructions, bind the wrong object, or attempt the knowledge-base step themselves.

Thus, the knowledge-base role is penalized by both operation complexity and terminal-chain exposure.

Developer Engineer and QA Test Engineer: comparatively robust roles. By contrast, `developer_engineer` and `qa_test_engineer` are generally more reliable across models. Their relative strength comes from two structural advantages. First, repository operations are comparatively regularized. Code tasks often involve well-structured artifacts such as pull requests, issues, branches, commits, or review comments. The required operations are therefore more standardized than cross-system business workflows. Second, these agents are often positioned near the *start* of the delegation chain rather than at the end. As a result, they are less exposed to upstream instruction loss. In many cases, they serve as the initial orchestrator or as a directly instructed specialist, which reduces the risk of incomplete handoff.
This contrast suggests that role performance is jointly determined by:

1. **tool-space regularity** (whether the action schema is standardized), and
2. **chain position** (whether the agent starts the workflow or receives a late delegated remainder).

Interpretation. Overall, these results show that agent-level evaluation should not be interpreted in isolation. A low pass rate may reflect not only the intrinsic difficulty of a role, but also the role's structural location in the workflow graph. In this benchmark, `knowledge_base_specialist` is disadvantaged because it combines both hard operations and late-stage dependency on upstream delegation. Meanwhile, `developer_engineer` and `qa_test_engineer` benefit from cleaner tool interfaces and earlier workflow positions.

G.2 Multi-step tasks fail through prefix decay and final handoff errors.

Case Study: Prefix Accuracy Decay in Multi-Subtask Workflows

Case Selection. A central question in multi-step agentic evaluation is not only whether a model eventually solves the full task, but *where* it begins to fail along a chained workflow. To study this, we examine **prefix accuracy** in multi-subtask tasks: after subtask i, what fraction of tasks still have a fully correct prefix $(1, \ldots, i)$?
Setup. We compare two benchmark families of multi-subtask workflows:

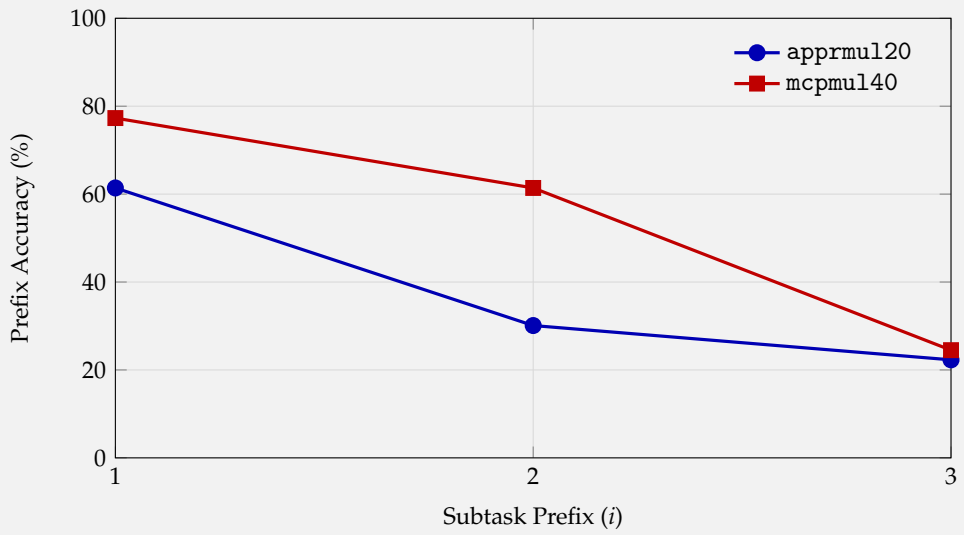
- **apprmul20**: multi-step approval tasks.
- **mcpmul40**: multi-step cross-system workflow tasks.

For a clean comparison, we focus on **three-subtask tasks**. The small number of two-subtask tasks in `apprmul20` are omitted from the main figure and discussion below.

Measurement. For each workflow family, we compute the prefix success rate after each subtask:

$$\text{PrefixAcc}(i) = \frac{\#\{\text{tasks whose first } i \text{ subtasks are all correct}\}}{\#\{\text{all 3-subtask tasks}\}}.$$

The figure shows the accuracy decay curves.



Model Behavior. The two workflow families exhibit different failure shapes. For approval tasks, the main loss appears in the middle of the workflow. Accuracy drops from **61.4%** after the first subtask to only **30.1%** after the second subtask, a decline of more than thirty percentage points. After that, the curve flattens somewhat, falling to **22.3%** at the full three-subtask prefix. For cross-system workflows tasks, the first two subtasks remain comparatively stable: accuracy decreases from **77.3%** to **61.4%**, which is a moderate drop. However, the final transition is much harsher: prefix accuracy collapses from **61.4%** to **24.5%** at the third subtask.

Interpretation. This case study shows that multi-subtask performance should not be summarized only by final task pass rate. Two workflow families may both end with low end-to-end success, yet fail in structurally different ways. Therefore, prefix-accuracy curves provide a more informative diagnostic than a single end-to-end metric. They reveal where the workflow becomes unstable, which is crucial for both system design and failure analysis.

Case Study: Weak Final-Stage Agent Binding in Qwen3.5-122B-A10B

Case Selection. We use a representative case from the multi-task MCP benchmark to illustrate a distinctive contrast between `Qwen3.5-122B-A10B` and `Qwen3.5-35B-A3B`. At the aggregate level, the larger model achieves a **higher subtask pass rate** but a **lower task pass rate**: `Qwen3.5-122B-A10B` reaches **6/40 = 15.0%** task success and **68/120 = 56.7%** subtask success, whereas `Qwen3.5-35B-A3B` reaches **9/40 = 22.5%** task success and **63/120 = 52.5%** subtask success. This means that the 122B model is somewhat better at making partial progress, but the 35B model is more effective at converting earlier progress into full end-to-end completion.

Question. The representative task follows a three-stage workflow. The first two subtasks establish the necessary upstream state. The third subtask, assigned to `collaboration_ops_specialist`, requires the agent to:

1. send a plain-text email to the specified recipients,
2. create a calendar event with the required scheduling parameters,
3. hand off the remaining knowledge-base work to `knowledge_base_specialist`
4. ensure the HR knowledge article is updated correctly as part of the final workflow completion.

This kind of final-stage prompt is harder than the first two stages because it must simultaneously bind together *prior workflow state*, *the current agent's responsibilities*, and *the downstream agent's responsibilities*.

Model Behavior. The most salient pattern is not that the 122B model misunderstands the overall objective. In fact, it often understands the workflow quite well. The problem is that, in the third subtask, it does not reliably bind the *last critical action* to the *correct specialist*.

In the representative case, the judge concluded that the model successfully completed the collaboration-operations actions: it sent the required email and created the calendar event correctly. However, the run

still failed because the model did not correctly relay the knowledge-base update instruction to the downstream specialist. Instead, after completing the email and calendar portions, it attempted to take over the knowledge-base update itself. Since that action belonged to a different specialist and required tools it did not have, the final step was left incomplete.
The 122B model tends to behave as if the final stage were a single integrated objective rather than a structured division of labor. As a result, three recurrent failure modes appear:

1. **Role confusion:** it mixes up what the current agent should do versus what should be delegated downstream.
2. **Handoff omission:** it completes the visible front-end actions (such as email and calendar) but fails to pass the remaining instruction bundle to the next specialist.
3. **Improper self-execution:** it tries to perform a delegated action itself, which then fails because the current agent lacks the required tools.

This pattern helps explain the aggregate metric gap. The 122B model is strong enough to carry the workflow through the earlier stages, so its subtask success rate is relatively high. But its final-stage conversion is unstable: once the workflow reaches the last subtask, the model is more likely to lose the precise alignment among prior artifacts, current-role actions, and downstream-role actions. By contrast, the 35B model appears weaker in some earlier subtasks, but it is more reliable at executing the final structured handoff needed for complete task success.

Result. The representative task failed at the full-task level even though the model completed substantial portions of the third subtask. The email and calendar actions were judged correct, but the workflow was still marked incomplete because the knowledge-base update was not stably assigned to the appropriate downstream specialist.
This is why the benchmark-level comparison looks counterintuitive at first glance: `Qwen3.5-122B-A10B` achieves more subtask wins overall, but those wins convert into fewer fully completed tasks.

Interpretation. This case reveals an important limitation of larger multi-agent models: improved local execution does not automatically produce better end-to-end workflow completion. In this benchmark, `Qwen3.5-122B-A10B` often succeeds in advancing the workflow through the first two stages, but in the third stage it is more prone to losing the exact mapping. As a result, the model frequently reaches a state where it has *understood the task globally* but failed to bind the final critical action to the correct executor. This explains why its subtask success rate can exceed that of the smaller model while its end-to-end task success remains lower.

G.3 Collaboration tools become bottlenecks during workflow closure.

Case Study: `collaboration_ops_specialist` as the Closing-Stage Bottleneck

Case Selection. We use a representative case (from GPT-5.4) to analyze why the `collaboration_ops_specialist` becomes the dominant bottleneck at the end of the pipeline. Across the GPT-5.4 benchmark run, this agent was evaluated **31** times but passed only **10** times, for a pass rate of **32.3%**. In many workflows it is responsible for the final subtask, so failures in this agent frequently convert an otherwise correct prefix into an end-to-end task failure.
Question. The representative task below already has the first two subtasks completed correctly: the customer case has been escalated, and the IT problem/change agent has created the required backend records. The final subtask is assigned to `collaboration_ops_specialist`.

> Keep the workflow moving using customer case 50, which involves the internal escalation for CS-0000050 related to NGINX Plus Variant 51, along with the associated problem record. First, send a plain-text email from Olivia Chen at `olivia.chen@techcorp.com` to Carlos Mendez, Nina Patel, and Priya Nair, addressing the subject as "CS-0000050 escalation review." In the email, please include a request for them to review CS-0000050 for NGINX Plus Variant 51, along with the containment path, current reproduction details, and any documentation updates.
>
> Next, create a calendar event on Alice's primary calendar with the summary "CS-0000050 escalation

review." This event should start on May 10, 2026, at 2:30 PM and end at 3:00 PM, both in the Asia/Shanghai time zone. Ensure that Carlos Mendez, Nina Patel, and Priya Nair are invited to this event, and send updates to all attendees.

After that, ask the knowledge base maintenance specialist to continue the workflow via HTTP. Update the CSM knowledge article with the knowledge ID 107, titled "NGINX Plus Variant 51 escalation triage," setting its state to published and visibility to internal. The product ID should be 71, and the owner ID is 795.

Finally, create a case knowledge link for case 50, associating it with knowledge ID 107 and marking it as suggested.

Model Behavior. The trajectory shows that the model was able to perform part of the subtask correctly, but failed because it did not internalize the precise contract of the calendar API.

(1) The email action was executed correctly. The trace shows a valid email tool call:

```
mcp_email_call_tool {... "userId":"olivia.chen@techcorp.com" ...}
```

and later confirms that the email was actually sent.

(2) The calendar action used the wrong identifier. Immediately afterward, the model invoked the calendar API with the wrong calendar identifier:

```
mcp_calendar_call_tool {... "calendarId":"alice@techcorp.com" ...}
```

However, the task explicitly required `alice-primary`. The calendar backend returned a clear error:

```
Calendar with id 'alice@techcorp.com' does not exist
```

(3) The model did not recover after the tool error. Instead of correcting the identifier from `alice@techcorp.com` to `alice-primary`, the model simply continued with the remaining workflow. The trace shows that it delegated the knowledge-base step, completed the article update, and then produced a final summary that omitted the failed calendar action:

```
DONE: Sent plain-text email ...  delegated the knowledge workflow via HTTP and
completed the CSM knowledge article update ...
```

In other words, once the calendar call failed, the model did *not* enter a repair step. It treated the task as mostly complete and moved on.

Result. The task failed even though the first two subtasks had already been completed successfully. The judge's conclusion was explicit:

```
The agent failed to create the calendar event because it used an incorrect
calendarId ('alice@techcorp.com' instead of 'alice-primary') and did not attempt
to correct it after receiving an error.
```

Thus, the final subtask failed not because of complex business logic, but because of a tool-contract mismatch: the model used a plausible-looking identifier that was semantically wrong for the target API.

Why This Agent Is the Bottleneck. This failure is not isolated. The result and trajectory files show that `collaboration_ops_specialist` repeatedly fails on three related classes of tool-contract errors.

(1) Calendar contract failures. The most frequent problem is confusion about the calendar identifier. In multiple traces, the model used `alice@techcorp.com` or `primary` where the API expected `alice-primary`. For example, another trace shows:

```
mcp_calendar_call_tool {... "calendarId":"alice@techcorp.com" ...}
```

followed by:

```
User not found with email:  alice@techcorp.com
```

The model then stopped without repairing the call. This means the agent often understands the *intent* (create a review meeting) but fails on the exact required API identifier.

(2) Email contract failures. A second recurring issue is sender identity and email payload formatting. In another representative trace, the model attempted to send from a user account that the session was not allowed to impersonate. The backend returned:

```
You can't access this email address.  Please try using "me" as your userId or
enter your own email address.
```

The model then aborted the rest of the collaboration workflow:

```
DONE:, UNDONE: Could not send the required email ...  so the calendar event ...
and the HR service specialist follow-up were not initiated
```

This indicates that the failure is not just in recipient generation, but in understanding the operational constraint that the email tool only permits certain sender identities.

(3) Teams contract failures. A third recurring issue appears in Teams-based coordination tasks. In these cases the model can often find the correct team and channel, and even send a message, but still fails because it omits a required parameter such as message importance. For instance, one trace shows a `send_channel_message` call whose JSON contains the body and routing fields but no explicit importance field. The corresponding judge decision states:

```
The agent failed to set the 'importance' parameter to 'high' ...  resulting in a
message with 'normal' importance instead of the requested urgency.
```

So even when the communication is sent, the agent still loses correctness on a missing contract field.

Interpretation. This case study suggests that the main weakness of `collaboration_ops_specialist` is not high-level reasoning, but **interface precision**. The agent usually understands the collaboration intent: send a message, schedule a meeting, or post a Teams notification. However, it repeatedly fails on the exact low-level contract required by the communication tools:

- using `alice@techcorp.com` or `primary` instead of `alice-primary`,
- using an unauthorized sender identity instead of the allowed `me` semantics,
- confusing a team display name with a true `teamId`,
- omitting required fields such as Teams message `importance`.

The practical consequence is severe: many workflows survive the first two subtasks but die at the closing communication step. This is why `collaboration_ops_specialist` becomes the dominant bottleneck in end-to-end multi-agent completion.

G.4 Delegation failures remain a central source of end-to-end errors.

Case Study: Delegation Failures as a Primary Source of Multi-Agent Errors

Case Selection. A major recurring failure mode in the benchmark is not tool invocation in isolation, but *delegation breakdown*. In many tasks, the model is explicitly required to hand work to a colleague with the appropriate tools or business role. When this delegation step is omitted, underspecified, or issued in the wrong order, the downstream subtask either fails immediately or becomes impossible to complete correctly.
Across the benchmark, delegation failures repeatedly fall into three broad categories:

- **delegation not initiated,**
- **delegation initiated with insufficient information,**
- **delegation issued at the wrong time or in the wrong dependency order.**

1. Delegation not initiated. The first and most direct failure mode is that the model simply does not call the required downstream specialist, even when the prompt explicitly says that it should.

This appears in tasks that mix domain-specific work with general collaboration actions such as email, calendar, or Teams operations. In these cases, the current agent often attempts to continue the workflow itself rather than handing off to `collaboration_ops_specialist` or another required role.
Representative benchmark judgments explicitly state missing actions such as:

> ‘‘Failed to call ask_collaboration_ops_specialist_by_http as required by the reference trajectory.’’

This pattern occurs across multiple models and starting agents:

- a task beginning from `hr_service_specialist`, where the workflow included email and meeting requirements but the required collaboration handoff was omitted;
- a task beginning from `it_service_desk_l1`, where the judge again marked the missed call to `ask_collaboration_ops_specialist_by_http`;
- a task beginning from `customer_support_specialist`, where the downstream collaboration step was not passed on correctly;
- a task beginning from `developer_engineer`, where the workflow advanced through engineering steps but failed to initiate the next-role handoff.

The key point is that these are not merely “missed helper calls.” In this benchmark, delegation is part of the required workflow semantics. If the current agent does not initiate the handoff, then later stages are structurally absent even if the upstream work is correct. In multi-agent workflows, failing to delegate is equivalent to failing to execute the task.

2. Delegation with insufficient information. A second recurring failure mode is that the upstream agent does delegate, but the delegated request is missing information that the downstream agent needs in order to act. A common pattern is knowledge-base creation or update tasks. The upstream agent hands off the work, but fails to include the actual article body, content payload, or complete action context. The downstream agent then either requests clarification or stalls because the request is underspecified.
A representative benchmark judgment describes exactly this pattern: the downstream agent requested the missing article content, but the upstream process terminated rather than supplying it, so the workflow never closed. Another nearby case shows the same structural issue in a different form: the upstream agent failed to pass the task to the correct role with a complete action bundle, and the downstream specialist therefore could not complete the required operation.
This means that the delegation failed even though the *call* itself happened. In practical terms, the handoff was syntactically present but semantically incomplete. A delegated subtask is only executable if the parent agent passes the missing state, not just the destination role.

3. Delegation at the wrong time or in the wrong dependency order. The third class of failures is especially important in code and repository workflows: the model delegates a subtask before the downstream preconditions exist.
One representative example occurs when a model asks the QA agent to modify a file on a branch that has not yet been created. The judge summary makes the failure explicit:

> ‘‘The agent delegated the QA task before creating the required branch, causing the QA task to fail. The agent did not retry the QA task after creating the branch.’’

This is a pure dependency-order failure. The downstream QA agent is not wrong; it is being asked to act in a state where success is impossible.
A closely related pattern appears when the developer agent performs part of the QA agent’s work *before* delegating it. For example, in some tasks, the developer agent created the QA checklist file itself and then handed the same step to QA. The delegated QA subtask subsequently failed because the file already existed. Here the problem is not missing delegation, but *incorrect ownership sequencing*: the parent agent collapses the role boundary before handing off.
A third variant appears in content-preparation workflows. The upstream developer agent creates the branch, creates the file, and delegates onward, but fails to fetch the actual source content from Drive or another artifact store. The downstream workflow therefore proceeds over placeholders or hallucinated content rather than the required material. In one representative case, the judge explicitly notes that the created file contained a placeholder comment rather than the actual SQL implementation from the referenced Drive file. In another, the model admitted that it lacked the content and substituted generic runbook text instead.
These examples show that timing and information dependencies are as important as role selection itself.

Model Behavior. Taken together, these cases reveal a common behavioral weakness: the models often treat delegation as a *surface routing step* rather than as a *state-sensitive contract*. In successful execution, the parent agent must do all of the following:

1. choose the correct downstream specialist,
2. delegate at the correct workflow point,
3. ensure all preconditions are already satisfied,
4. pass the exact content, identifiers, and action context the child needs, and
5. retry or repair the handoff if the first attempt fails for dependency reasons.

Many failures occur because one of these conditions is dropped. The result is often a workflow that looks partially correct from the top level but is broken at the coordination boundary.

Result. These delegation failures lead to several benchmark-visible outcomes:

1. the required downstream role never acts,
2. the downstream agent acts on incomplete context and terminates,
3. the downstream agent receives an impossible task because dependencies are not established,
4. the upstream agent performs work that should have remained delegated, causing duplication or downstream collision,
5. the task fails even when several individual tool actions were executed correctly.

Interpretation. This common pattern suggests that delegation should be understood as a first-class reasoning problem rather than a secondary implementation detail. The model must reason not only about *what* the overall task requires, but also about *who* should perform each step, *when* that handoff becomes valid, and *what information* must cross the boundary. The benchmark shows that many multi-agent failures are not single-tool errors; they are coordination-contract failures between agents. In this sense, delegation is one of the central bottlenecks in multi-agent enterprise workflows. A model may understand the global task correctly and still fail because it cannot maintain the correct sequence of ownership, preconditions, and information transfer across agent boundaries.

G.5 Stateful database operations trigger incorrect fallback actions.

Case Study: Models Execute a Similar but Wrong Operation After Failures

Case Selection. We select a representative failure case in which the model is asked to update an existing database record, but instead creates a new record that looks superficially similar. This pattern is especially important because the model does not simply stop after a failed retrieval. Instead, it falls back to a nearby but semantically incorrect action, producing a plausible-looking output while violating the true task objective.

Question. The task requires strict alignment among an incident, an HR case, and a knowledge article. The key requirement is that the existing incident `INC_051` must be updated first, and all downstream actions must be attached to that same record.

> Align incident `INC_051`, HR case `HRC0000744` with HR case ID 34, and knowledge article `KB_018` for incident-to-knowledge alignment. First, update incident `INC_051` to in progress, assign it to `USER_022` in assignment group `GROUP_002`, and add the worknote: `Requesting knowledge alignment with KB_018 for network port failure.`
>
> Next, ask the knowledge base maintenance specialist to update knowledge article `KB_018`, titled "Slow Internet in Building B," by refreshing the recovery path for network port failure and documenting the validated owner handoff for payroll setup acceptance. Keep it published, make it internally visible, set the owner to `USER_039`, and then link it to incident `INC_051` as applied.
>
> After that, ask the HR service specialist to update HR case `HRC0000744`, which is HR case 34, to

work in progress, assign it to Helen Zhou, and add the worknote: `Using incident INC_051 and knowledge KB_018 for coordinated routing.` Then have them create an HR case task under parent case 34 with ready status, the short description "Track knowledge-aligned recovery for payroll setup acceptance," assigned to Helen Zhou as a checklist task, and add the worknote: `Follow the steps refreshed in KB_018 while closing HRC0000744.`

Finally, ask the collaboration operations specialist to send an email from `olivia.chen@techcorp.com` to `ivan.park@techcorp.com`, `priya.nair@techcorp.com`, and `helen.zhou@techcorp.com` with the subject "INC_051 knowledge alignment" and the message: `Knowledge article KB_018 has been refreshed for INC_051. Please align recovery actions and close remaining blockers.` Then create a calendar event on `alice-primary` titled "INC_051 knowledge alignment review" from `2026-05-02 15:00:00+08:00` to `2026-05-02 15:30:00+08:00` in the Asia/Shanghai time zone, with the description "Confirm the updated guidance in KB_018 and the remaining work on HRC0000744," and invite Ivan Park, Priya Nair, and Helen Zhou with updates sent to all.

Model Behavior. The model correctly completes several downstream actions, but the core failure occurs at the very beginning of the workflow, where it should have modified the existing incident. According to the judge, the model:

> `The agent failed to update the existing incident INC_051. It incorrectly searched for the incident using 'find_incident_by_number' with the ID format ('INC_051') instead of the number format, concluded it didn't exist, and created a new incident (INC_101) instead. Consequently, INC_051 was never updated.`

This trace reveals a three-step failure mechanism: **(1) Identifier confusion.** The task provides `INC_051` as the existing incident identifier. Instead of treating this as the canonical object to update, the model uses it as a search string for a number-based retrieval function. In other words, it confuses *an incident ID* with *the string format expected by a number-based lookup tool*; **(2) Retrieval failure is misinterpreted as nonexistence.** After the lookup fails, the model does not pause, retry with a different retrieval method, or explicitly report that the record could not be resolved. Instead, it assumes the failed search means that the incident does not exist; **(3) A wrong fallback action is executed.** Rather than stopping, the model performs a similar but incorrect operation. The runtime summary makes this explicit:

> `DONE: ... Incident INC_101 (originally requested as INC_051) -- Created and updated to in_progress, assigned to USER_022 ...`

This is a particularly dangerous failure mode. The model does not simply fail silently; it creates a new object that appears operationally valid, then continues the remainder of the workflow on top of that wrong object. As a result, downstream agents successfully update the HR case, send the email, create the calendar event, and link the knowledge article — but they do so relative to the new incident rather than the intended one.
Importantly, this behavior is not unique to one model or one object type. We observed the same pattern across multiple systems:

- In another incident-alignment task, a different model was judged to have

  > `incorrectly created a new incident instead of modifying the existing one`

  after failing to update `INC_049`.
- In a separate incident case, another model similarly

  > `failed to update the existing incident INC_057, instead creating a new incident (INC_101)`

Result. The overall task fails because the original incident was never updated. The judge explicitly records the missed requirements as:

- `Failed to update incident INC_051 to 'in_progress'.`
- `Failed to assign incident INC_051 to USER_022 in assignment group GROUP_002.`
- `Failed to add the requested worknote to incident INC_051.`

Thus, even though later subtasks were completed in a superficially coherent way, the workflow is still incorrect because it is anchored to the wrong database object.

Interpretation. This case illustrates a broader and more subtle failure pattern than ordinary tool misuse. The model does not merely fail to retrieve a record. Instead, after a failed update attempt, it substitutes a *similar but semantically different operation*: **create instead of update**. The result is an execution that looks productive but is actually inconsistent with the task specification.
Together, these examples suggest a general cross-model failure mode: when a database update fails, LLM agents may switch to a nearby creation operation rather than preserving the distinction. This is especially risky in enterprise workflows, because the resulting state can look internally consistent while still being attached to the wrong object.

G.6 Tool calls often fail at the parameter-semantics level.

Case Study: Business-Semantic Parameter Errors in Tool Calls

Case Selection. A recurring benchmark-wide failure mode is that the model *does call the correct tool*, but the *business semantics of the parameters are wrong*. This is not the same as failing to use tools at all. Instead, the execution reaches the tool layer, but the parameter values, field meanings, or target-object semantics do not match the task requirements.
This class of errors is especially important in enterprise workflows because many workflow tasks are evaluated not just on whether a record was touched, but on whether it was updated with the *correct business meaning*. Across the benchmark, we observe five especially common subtypes:

1. wrong enum value,
2. wrong relationship semantics,
3. updating the wrong object or creating a new object instead of updating an existing one,
4. wrong assignee semantics,
5. wrong status or task-type semantics.

1. Wrong enum values. The most common semantic error is a wrong enumerated value. A typical example is incident creation where the prompt explicitly requires `category = software`, but the model either omits the field or lets the system fall back to the default value `inquiry-help`.
This pattern appears clearly in the benchmark results for multiple models. In one representative trajectory, the task explicitly requests:

> ‘‘Create a new high-priority software incident ...’’

The tool call succeeds, but the resulting record contains:

```
"category": "inquiry-help"
```

rather than

```
"category": "software"
```

The judge therefore marks the subtask as failed even though the incident was otherwise created with the correct caller, channel, impact, urgency, priority, assignee, and worknotes.

2. Wrong relationship semantics. A second frequent pattern concerns relation fields such as whether a knowledge article is linked as an `applied` solution or only as a `suggested` one.
In several benchmark cases, the task requires the article to be linked as `applied`, but the model either passes the wrong field name (for example `link_type` instead of `used_as`) or omits the correct relation parameter. As a result, the system defaults to `suggested`. The article is linked successfully, but the link has the wrong business meaning.
A representative judge description makes this explicit:

> ‘‘The agent successfully updated the knowledge article but failed to link it ...as ‘applied’. It used the incorrect parameter name link_type instead of used_as, causing the system to default to ‘suggested’.’’

This kind of error is subtle but important. The database operation succeeds, yet the workflow still fails because the semantic relationship encoded in the record is not the one requested by the task.

3. Wrong target-object semantics Another major failure mode occurs when the task requires an *update* to an existing object, but the model instead creates a *new* one. This often happens when the model mishandles object identity during retrieval or parameter binding.
For example, the task may require updating an existing incident such as `INC_049` or `INC_057`. Instead, the model searches by the wrong identifier format, fails to find the target, and then creates a fresh incident such as `INC_101`. The same pattern appears for knowledge articles: instead of updating `KB_004` or `KB_023`, the model creates a new article.
Representative judge summaries repeatedly note failures of the form:

> ‘‘The agent failed to update the existing incident ...Instead, after failing to find it by number, it created a new incident.’’

and

> ‘‘The agent failed to update the existing article ...instead creating a new article.’’

The runtime summary makes this explicit:

> DONE: ... Incident INC_101 (originally requested as INC_051) -- Created and updated to in_progress, assigned to USER_022 ...

This is a deeper semantic problem than a bad field value. The model is not merely setting the wrong attribute on the right object; it is acting on the wrong object entirely.

4. Wrong assignment semantics. Assignment fields are another common source of business-semantic errors. The prompt may require assigning a task or case to Helen Zhou or to `USER_022`, but the model instead passes a hallucinated ID, an incorrect user, or fails to resolve the assignee at all.
In one representative case, the benchmark judge states that the model:

> ‘‘hallucinated the user ID for Helen Zhou ...resulting in no assignment change in the database.’’

In another, the HR case was assigned to the wrong person entirely. In yet another, the model claimed in its final answer that the assignment had been updated, but the actual `assigned_to` field was unchanged.
These errors matter because assignment is usually not cosmetic in enterprise workflows. It determines ownership, routing, and service accountability.

5. Wrong status or task-type semantics. A fifth common pattern concerns status and task-type fields, especially in HR task creation. The prompt may require a checklist-style child task with `status = ready` and `task_type = checklist`, but the model instead sets `draft`, leaves `task_type` null, or omits the field entirely.
Representative judge statements include failures such as:

> ‘‘Set the HR case task status to ‘ready’ (agent set it to ‘draft’).’’

and

> ‘‘Missed setting the ‘task_type’ parameter to ‘checklist’.’’

Again, the tool call may succeed and create a task object, but the created task does not carry the intended workflow semantics.

Interpretation. This common pattern suggests that MCP performance depends on more than planning and delegation. A model must also preserve the exact *business semantics* of the tool parameters it emits. That includes:

1. selecting the correct target object,
2. selecting the correct enum value,
3. selecting the correct relationship type,
4. selecting the correct assignee identity,
5. selecting the correct lifecycle state and task type.

Thus, a substantial portion of benchmark failure is best understood as *semantic parameterization error*. The model is often close enough to appear competent at the action level, but not precise enough to satisfy the workflow’s structured business contract.

Case Study: GPT-5.4 Early Semantic Misinterpretation Failure in Multi-task

Case Selection. We use a representative failure case from the benchmark to illustrate a weakness (from GPT-5.4) in multi-agent LLM workflows: an early semantic misinterpretation in the first subtask prevents all later subtasks from executing.

Task Description. The task consists of three sequential user prompts, each assigned to a different specialist agent.

1. **Subtask 1 of the task (to `it_change_engineer`).**

   Alice Carter from the networking team and Bob Davis from client relations are currently managing this crisis. Amid the pressing nature of customer escalations, particularly regarding the case identified as CS-0000048, it is imperative that a meticulous internal triage is initiated forthwith. Prior to any further actions, ensure that an incident is duly created, with the caller identified as the former, and the short description articulating the internal escalation pertaining to the aforementioned case under the Premium Support SLA, which operates continuously at Tier 25. This incident must be categorized under software, marked as new, and assigned the utmost levels of impact and urgency, with the same assignee designated for the task. Following this, a change record must be established, encapsulating the essence of containment planning for the same case, ensuring that it is set to assess status, with a medium impact and risk, yet still treated with high priority. The category for this record should reflect application software, and it must also be assigned to the same individual. Once these records are in place, a request should be dispatched via HTTP to the customer support specialist, urging them to proceed with the workflow.

2. **Subtask 2 of the task (to `customer_support_specialist`).**

   Sustain the same workflow by using the internal escalation incident for CS-0000048 related to the Premium Support SLA, which operates 24/7 at Tier 25. Update the customer case with ID 48, setting the priority to high, the state to in progress, and assigning it to the user with ID 795. Mark this as an escalation due to a breach risk. After that, ask the collaboration operations specialist to continue the workflow via HTTP.

3. **Subtask 3 of the task (to `collaboration_ops_specialist`).**

   Proceed with the same workflow by using the incident for internal escalation regarding CS-0000048, which falls under the Premium Support SLA (24x7, Tier 25). This includes the change record and customer case 48.

   First, send a plain-text email from Olivia Chen at `olivia.chen@techcorp.com` to Carlos Mendez, Nina Patel, and Priya Nair at their respective email addresses, with the subject line "CS-0000048 internal escalation review." The email should request a review of CS-0000048 for the Premium Support SLA (24x7, Tier 25), including the current symptom, containment path, and any documentation updates.

   Next, create a calendar event on Alice's primary calendar. The event should have the summary "CS-0000048 internal escalation review" and is scheduled to start on May 4, 2026, at 10:00 AM and end at 10:30 AM, all in the Asia/Shanghai time zone. The event description should cover the review of customer impact, containment, and next steps for the Premium Support SLA (24x7, Tier 25). Ensure that Carlos Mendez, Nina Patel, and Priya Nair are invited to this event.

   After that, ask the knowledge base maintenance specialist to continue the workflow via HTTP.

   Then, update the CSM knowledge article with the knowledge ID 326, titled "Premium Support SLA Tier 25 escalation path." Set the state to published, the visibility to internal, and associate it with product ID 217 and owner ID 795.

   Finally, create a case knowledge link for case ID 48, linking it to knowledge ID 326 and marking it as suggested.

Model Behavior. The model failed in the first subtask, and the error propagated to the rest of the task. The execution trace shows the following sequence:

1. The first assigned agent, `it_change_engineer`, began by inspecting the ITSM toolset and retrieving the schemas for `create_incident`, `create_change`, `get_user_using_name`, and `find_sla_definition_by_name`.

2. Instead of resolving the phrase "the former" from the prompt as a discourse reference, the model treated it as a literal person name and queried ITSM for a user named `Former Former`.
3. The user lookup failed with a `NOT_FOUND` error. In parallel, the model also attempted to resolve `Premium Support SLA` through an exact-name lookup, which likewise failed.
4. After these two lookup failures, the model stopped rather than recovering with an alternative interpretation or proceeding with partial execution. It did not create the incident, did not create the change record, and did not send the required HTTP request to the next agent.
5. Because the task was evaluated as a strictly sequential workflow, failure in the first subtask caused the remaining two subtasks to be skipped entirely.

The runtime summary explicitly reported:

```
DONE:, UNDONE: Incident and change creation could not be completed; workflow
request to customer support specialist not sent, ERROR: Mandatory caller/assignee
identifier is missing because user 'Former Former' was not found in ITSM; SLA
'Premium Support SLA' was also not found.
```

Result. The overall task failed. The judge concluded that the model:

```
''misunderstood the prompt's reference to 'the former' as a literal name 'Former
Former', failed to find the user, and subsequently aborted the task without
creating the incident, creating the change, or contacting the customer support
specialist.''
```

As a result, The first subtask failed. No incident, no change record, and no HTTP delegation were produced. The second subtask and the third subtask were skipped.

Interpretation. This case illustrates a key weakness of LLM-based multi-agent systems in enterprise workflows: a small reference-resolution error at the beginning of the pipeline can trigger a full cascade failure. The model was not primarily defeated by a complex business rule. Instead, it failed on a basic semantic interpretation problem, namely resolving "the former," and because the workflow was strictly sequential, that single mistake prevented all downstream agents from acting.

G.7 Higher reliability can require much higher coordination cost.

Case Study: DeepSeek Achieves Correctness Through Costly Cross-Agent Verification

Case Selection. We select a representative "high-accuracy but high-cost" case from DeepSeek-v4-pro. In our aggregate statistics, DeepSeek-v4-pro passes this task with **8,908,553** total tokens and **239** trace events, while Minimax M2.7 fails the same task with only **603,173** tokens and **56** trace events. This pattern is not isolated: the same model also shows extreme coordination cost on `task_35`, where it reaches **11,680,059** tokens and **314** trace events.

Question. The task is a three-stage cross-system workflow involving HR knowledge management, ITSM routing, HR case handling, and customer support escalation.

1. **Subtask 1 (to `knowledge_base_specialist`).**
   Update HR knowledge article ID 2 ("New Hire Orientation") by setting it to published, internal, and owned by user 39; send a plain-text email from Priya Nair to Ivan Park, Helen Zhou, and Carlos Mendez with the subject "HR and ITSM routing unification alignment"; then ask `it_service_desk_l1` to continue the workflow via HTTP.
2. **Subtask 2 (to `it_service_desk_l1`).**
   Use knowledge article 2 and the alignment email to update incident `INC_057`: set the status to `in_progress`, assign it to `USER_022`, place it in `GROUP_002`, add routing-cleanup worknotes, and then ask the HR service specialist to continue via HTTP.

3. **Subtask 3 (to `hr_service_specialist`).**
   Update HR case `HRC0000757` to `work_in_progress`, assign it to Helen Zhou, refresh the worknotes, create a checklist-style child HR task, ask the customer support specialist to continue via HTTP, and finally update customer case 27 to high priority, in progress, assigned to user 795, with escalation reason `customer_request`.

Model Behavior. DeepSeek does not execute this task as a short linear workflow. Instead, it repeatedly expands the scope, consults additional specialists, and performs explicit end-state verification after intermediate actions.
First, the initial knowledge-base step begins with explicit planning and broad tool inspection rather than immediate execution. The trajectory shows the model saying:

> "Let me start by understanding what needs to be done. I'll work through this systematically... Let me first check what tools are available across the relevant systems and look up the knowledge article."

This quote comes directly from the `knowledge_base_specialist` trace.
Second, although Subtask 1 is nominally assigned to `knowledge_base_specialist`, the trace already expands into a multi-agent coordination pattern. In the first subtask alone, the runtime trace records activity from `knowledge_base_specialist` (**65** events), `it_service_desk_l1` (**89**), `customer_support_specialist` (**54**), and `hr_service_specialist` (**31**), for a total of **239** events. This is the core reason why token usage grows so quickly: the model solves one user request by recursively activating several other specialists.
Third, the downstream agents continue the same conservative pattern. After updating the incident, `it_service_desk_l1` does not stop at the required state change; instead, it explicitly delegates again:

> "Incident updated successfully. Now let me delegate to the HR service specialist to continue the workflow.""

Later, the `customer_support_specialist` again reopens the context instead of treating the task as a narrow update:

> "Let me start by updating case CS-0000027 with the specified changes and then look up relevant information to continue the workflow."

The same agent ends with another explicit verification round:

> "Now let me verify the final state of the case and summarize everything."

Finally, the model repeatedly summarizes and re-checks completed work:

> "The email has been sent successfully. Now let me summarize everything that's been completed. Let me verify the final state of all items:"

This repeated "plan, delegate, verify, summarize" pattern is exactly what makes the run expensive. The token burden is also highly concentrated in downstream coordination: `customer_support_specialist` alone consumes **6,847,046** tokens, far more than the other agents in the same successful run.

Result. Despite the **high coordination cost**, the run is judged as **passed**. The final state shows that the required workflow was completed across all systems:

1. The HR knowledge article was successfully updated and remained in the required published/internal state, with owner changed to user 39.
2. Incident `INC_057` was updated from `new` to `in_progress`, reassigned to `USER_022`, and annotated with routing-unification worknotes.
3. HR case `HRC0000757` was updated to `work_in_progress`, assigned to Helen Zhou, and linked to the refreshed knowledge guidance; a new checklist task (`TASK-0001`) was also created under the parent HR case.
4. Customer case `CS-0000027` was updated and escalated, including assignment to user 795 and escalation reason `customer_request`.

Interpretation. This case shows that DeepSeek's stronger correctness can come from a very expensive execution strategy. Rather than solving the workflow with a minimal number of tool calls, it repeatedly re-enters the state space, asks other specialists to confirm context, and performs explicit final-state verification at multiple stages. In other words, the model reaches the correct answer partly by buying safety through additional coordination rounds. That strategy improves completion quality on this task, but it does so at an order-of-magnitude higher token cost.

G.8 Approval workflows expose weak decision commitment.

Case Study: Mimo-V2-Flash's Extreme Looping Behavior in Approval Workflows

Case Selection. We select a more extreme failure case from the approval-workflow benchmark to illustrate a dangerous execution pattern of the Xiaomi Mimo-V2-Flash Model. In this case, the initial relay is correct, but the delegated legal specialist enters a severe document-reading loop, repeatedly rereading the same policy files until the context explodes. Compared with a simple wrong answer, this behavior is more concerning because it wastes a massive amount of computation while still failing to produce a usable decision.

Question. The task is a delegated approval workflow. The first subtask of the task is reproduced below.

1. **Subtask 1 of the task (assigned to `collaboration_ops_specialist`).**

   > Project Beacon Analytics Rollout is submitted by Strategic Partnerships on 2026-05-03. The intake covers a publicity waiver release requirement review. The intake is routed to the named specialists for adjudication against the cited policy excerpts and the attached evidence. Controlling policy excerpt: "Where necessary, signed Publicity Waiver and Release agreements are in place with named individuals, and individuals appearing in audio or visual content." Business parameters: `case_id=PUBL-2026-0007;...` First, ask `legal_approval_specialist` to perform Review; the review should read `submission/T-0007/approval_intake_form.md` and ...

Model Behavior. The trajectory shows a striking contrast between the initial agent and the delegated legal specialist.

(1) The initial delegation is correct. The first assigned agent performs the expected relay action. The trace shows a direct HTTP handoff:

```
ask_legal_approval_specialist_by_http(...)
```

Thus, the failure does not begin at task understanding or delegation. The root agent successfully forwards both the request and the evidence file paths.

(2) The delegated legal specialist enters an extreme loop. Once the legal specialist takes over, the run becomes pathological. Instead of reading the submitted evidence and producing a compliance decision, it starts repeatedly traversing and rereading policy files. The trace for the delegated specialist contains 4,516 trace events, indicating prolonged uncontrolled execution.
Most importantly, the repetition is concentrated on the same policy files. The top file-related calls include:

- `tool_workspace_read_file(escalation_ethics.md)` 500 times
- `tool_workspace_read_file(policy_docs/material_review.md)` **42 times**
- `tool_workspace_list_files(policy_docs)` **24 times**
- `tool_workspace_read_file(publicity_waiver_release_policy.md)` **21 times**
- `tool_workspace_read_file(rulebook.md)` **20 times**

This is not a normal retry pattern. It is a highly repetitive loop over the same workspace documents. The trace repeatedly shows the model reading one policy file, printing part of its contents, and then reading another policy file again instead of synthesizing a final decision. Near the end of execution, the same file is still being reread:

`tool_workspace_read_file(policy_docs/material_review.md)`

followed immediately by another emission of the document header:

```
# Material Review (Internal & External) ...  This document is read by the
legal_approval_specialist during reviews involving this topic.
```

(3) Context growth becomes catastrophic. Because the model keeps reloading long policy documents into the running context, the token footprint explodes. The delegated legal specialist alone consumed 104,307,660 input tokens and produced only **24,414 output tokens**. By contrast, the root agent consumed only **335,544 input tokens**. This shows that the cost is overwhelmingly concentrated in the downstream specialist loop rather than the initial coordination step.
The root trace also records an explicit failure signal from the delegated specialist:

```
Error: delegation to 'legal_approval_specialist' failed: HTTP 500 from peer ...
maximum context length is 262144 tokens ...
```

Thus, the workflow does not merely become expensive; it crashes because the repeated document reads exhaust the context window.

Result. The workflow fails despite the correct initial delegation.

- The first agent successfully forwards the legal review request and the evidence paths.
- The delegated legal specialist fails to transition from document retrieval to final adjudication.
- The specialist enters a runaway loop of repeated policy reads, especially on `policy_docs/escalation_ethics.md`.
- The execution ends with a context-window error rather than a final legal decision or rationale.

In other words, the task is not defeated by a subtle compliance rule. It is defeated by an uncontrolled internal loop that turns a simple delegated review into a context-exhaustion failure.

Interpretation. This case demonstrates a particularly dangerous failure mode of the Xiaomi model in multi-agent settings. The model can perform the initial delegation correctly, which makes the run appear healthy at first. However, once a downstream approval specialist begins reasoning over local policy documents, it may enter a repetitive retrieval loop instead of producing a bounded decision. The result is extreme token waste, loss of termination behavior, and eventual system failure.
This pattern is not isolated: other approval cases show the same phenomenon with different policy files, such as hundreds of rereads of `policy_docs/nda_contracts.md` or repeated cycling between `policy_docs/sales_revenue.md` and `policy_docs/material_review.md`. The broader implication is that the model is not merely inaccurate under load; it is operationally unstable in document-heavy delegated workflows.

G.9 Small models are much weaker on executable workflow tasks.

Case Study: Sequence Robustness and Fragility in Different Models

Case Selection. This case study examines a benchmark-level contrast between two model families. The first contrast concerns *long-sequence workflow robustness*: how models behave when moving from single-round complex workflow tasks to multi-round workflows tasks. The second contrast concerns *tool-use fragility in small open models*: how much performance collapses when tasks require strict MCP-style tool orchestration rather than simpler approval-routing behavior.
Closed-source models are more robust on long-sequence workflows. A broad trend in the benchmark is that most models perform worse on multi-round workflows than on single-round complex tasks. Intuitively, this is expected: multi-round workflows require the model to preserve earlier state, maintain the current role, bind downstream responsibilities correctly, and continue acting after prior artifacts have already been created. For example, in the benchmark summary, `DeepSeek-v4-pro` drops from **61.25%** on the single-round setting to

50.00% on the multi-round setting. This is the common pattern: longer delegation chains and more stateful continuation lead to lower success.
However, two closed-source models stand out as exceptions:

1. `Claude Sonnet 4.6` increases from **42.50%** to **50.00%**.
2. `Gemini 3.1 Pro Preview` also improves in the multi-round setting; in our local benchmark results, it rises from **31.87%** to **45.00%**.

This reversal is important. It suggests that stronger closed-source models are not merely better at solving isolated subtasks; they are also better at *maintaining coherent workflow state across longer sequences*. In practice, this means they are more likely to:

1. preserve references to artifacts created earlier in the chain,
2. keep track of which role is currently active,
3. distinguish between current-agent actions and downstream delegated actions, and
4. continue execution without losing the logical structure of the workflow.

The key difference is not just raw task-solving ability, but the ability to remain structurally coherent as the workflow becomes longer and more stateful.

Small open models are substantially weaker on workflows than on approval-routing tasks. A second benchmark-wide pattern is that small open models degrade far more sharply on MCP tasks than on approval-routing tasks. The reason is not simply that MCP tasks are harder in the abstract; rather, MCP tasks require a combination of strict tool selection, precise parameter formatting, and correct API argument passing.
Qwen3.5-9B: schema-reading without execution. The clearest example is `Qwen3.5-9B`. On MCP tasks, it records **0/108** task success, while on simpler approval-routing tasks it performs substantially better (for example, **22/80 = 27.50%** on one approval benchmark). Its dominant MCP failure mode is not a wrong final answer after active execution. Instead, it often fails *before business execution begins*.
In a large share of MCP failures, the model first calls `list_tools`, sometimes follows with `get_tool_schema`, absorbs long tool descriptions into context, and then stops. Representative judge summaries repeatedly state that the agent:

> ‘‘only listed the available tools and then stopped without performing any of the requested actions’’

Instead, it frequently stalls at the schema-reading stage. Thus, the core weakness is not only poor reasoning over the business workflow, but also failure to transition from *tool understanding* to *tool execution*.
Gemini 3.1 Flash-Lite Preview: unstable parameter realization. A different but related failure pattern appears in `Gemini 3.1 Flash-Lite Preview`. This model is not as inert as Qwen-9B. It often does understand the intended action and does attempt tool usage. However, it struggles with the stricter interface discipline required by MCP tasks.
This gap suggests that the model is much more comfortable with lightweight routing and textual decision tasks than with structured multi-system action execution.
The failure reasons consistently point to parameter-level breakdowns. Representative examples include:

1. using `work_notes` instead of `worknotes`,
2. using `parent_case_id` instead of `parent_case`,
3. omitting required fields such as `task_type`,
4. failing to include required calendar parameters such as `sendUpdates`.

These are not high-level misunderstandings of the workflow. Rather, they are *interface-landing failures*: the model gets close to the correct operation semantically, but fails to satisfy the exact MCP contract. The model must also realize the action in exactly the right tool schema and parameter form.

Interpretation. Taken together, these results show a two-level capability divide.
First, among stronger models, the important distinction is whether they can remain coherent across long, stateful, delegated workflows. This is where some closed-source models show an advantage.
Second, among smaller open models, the bottleneck is often even earlier: they may fail to turn tool availability into actual execution, or they may approximate the intended action but fail at the strict parameter-binding stage required by MCP interfaces.
Therefore, the benchmark reveals two different notions of robustness:

1. **sequence robustness**, which matters for long multi-round workflows, and
2. **interface robustness**, which matters for precise MCP tool execution.

Closed-source models appear stronger on the first dimension, while small open models remain weak on the second.

G.10 Some failures occur before real tool execution begins.

Case Study: Minimax-M2.7 Emits Pseudo Tool Calls

Case Selection. We present a representative failure case for Minimax's weaker tool-use capability. This case is especially informative because the model succeeds in Stage 1 with a genuine structured delegation, but then fails in Stage 2 by outputting a *pseudo tool call* as plain text rather than producing a real executable tool invocation.
Question. The task is a three-stage approval workflow for case `MUL-2026-0022` involving different approval specialists.

1. **Subtask 1 (to `qa_test_engineer`).**
   Ask `procurement_approval_specialist` to perform a contractor background screening review based on `submission/MT-0022/s1_approval_intake_form.md`, and return after delegation is completed.
2. **Subtask 2 (to `it_change_engineer`).**
   Ask `finance_approval_specialist` and `procurement_approval_specialist` to perform an IT purchase review documentation check, reading four specified submission files, and return after delegation is completed.
3. **Subtask 3 (to `it_change_engineer`).**
   Ask `legal_approval_specialist` and `procurement_approval_specialist` to perform an anti-corruption prohibited-payment review, again by relaying the provided case context and file paths only.

Model Behavior. The crucial behavior appears in Subtask 2. In Subtask 1, Minimax demonstrates that it *can* produce a proper tool call when it remains in the structured calling regime. The trajectory for Subtask 1 shows a normal delegation sequence: the message contains a populated `tool_calls` field, followed by a real `tool_call` event, then `delegate_start` and `delegate_done`. The task is completed correctly.
However, in Subtask 2, the model switches from executable structure to textual imitation. The trace records the following `agent_message` from `it_change_engineer`:

```
"I'll delegate both approval tasks to the respective specialists.  Let me
initiate both approvals simultaneously."
<minimax:tool_call>
<invoke name="ask_finance_approval_specialist">...
```

This looks superficially like a tool invocation, but the trace metadata shows that the message had

```
tool_calls:  []
```

rather than a structured tool-call object. Correspondingly, there is **no real** `tool_call`, `delegate_start`, or `delegate_done` event for these supposed Subtask-2 delegations. The LLM response metadata further shows:

```
finish_reason:  stop
```

instead of `tool_calls`. In other words, the model did not enter tool-execution mode at all; it merely produced text that resembled an invocation syntax.
This failure mode is qualitatively different from ordinary planning errors. The model did not misunderstand the task content. It explicitly stated the correct next action, named the correct specialists, and even rendered a plausible invocation template. The problem is that the invocation remained *unexecuted text*. The system therefore observed only six trace events for Subtask 2, all local to `it_change_engineer`, and none from the downstream approval specialists.

Result. The overall task failed. The benchmark summary reports **task_passed = 0**, with **32,964** total tokens. Only one of the three subtasks passed. Subtask 1 succeeded, but Subtask 2 failed because the required finance and procurement delegations were never actually issued, and Subtask 3 was skipped entirely.
The failure is visible directly in the runtime summary for Subtask 2:

```
subtask_result_preview:  "I'll delegate both approval tasks ...
<minimax:tool_call> <invoke ...>"
```

This confirms that the benchmark recorded the raw pseudo-call text itself as the final output of the subtask.

Interpretation. This case illustrates a specific weakness in Minimax-M2.7's tool-use behavior: under some conditions, the model does not emit a structured tool call even though it appears to know one is required. Instead, it generates a textual simulation of tool syntax. This is more damaging than a normal reasoning error, because the output may look operationally correct to a casual reader while being functionally inert in the execution trace. In multi-agent workflows, such pseudo calls break the control flow immediately: no downstream specialist receives the request, no tool-side state changes occur, and later subtasks cannot proceed.

H Limitations

EntCollabBench is a simulated benchmark and cannot cover every aspect of real enterprise work. Although the environment includes stateful service systems, role-specific permissions, and cross-departmental delegation, it abstracts away many factors present in deployed organizations, such as human preferences, informal communication norms, and noisy or incomplete real-world records.

The benchmark also has finite coverage. Our organization contains 11 agents across six departments and focuses on workflow execution and approval decisions. These settings cover many common enterprise operations, but they do not exhaust all enterprise domains. Similarly, the Approval Track is based on selected policy sources and may not reflect the full ambiguity and jurisdictional variation of real enterprise policies.

For cases requiring semantic judgment, we use model-based judges with majority voting, which reduces but does not eliminate possible evaluator bias.

Finally, the experiments depend on contemporary LLM systems and their tool-use implementations. Closed-source models may change over time, and multi-agent runs can be expensive due to long traces and repeated delegation.

I Broader Impact

EntCollabBench is intended to support the development of more reliable enterprise agents by evaluating whether they can coordinate across roles, respect permission boundaries, and complete stateful workflows. Better evaluation in this setting may help organizations identify failure modes before deployment, reduce unsafe automation, and design agent systems with clearer accountability and access-control constraints.

At the same time, enterprise agents can create risks if deployed prematurely or with excessive permissions. Failures in routing, context transmission, or parameter grounding may lead to incorrect records, missed approvals, inappropriate customer communication, or unauthorized actions. More capable collaborative agents could also be misused to automate harmful organizational activity or bypass human review if access controls are poorly designed.

We mitigate these risks by using simulated enterprise data and sandboxed systems rather than real organizational records. The benchmark enforces role-based permission isolation and evaluates agents through controlled state verification. We recommend that real-world deployments include human oversight, audit logs, conservative permission scopes, and safeguards against irreversible actions.