MACRO: A Benchmark for Clarification-Aware Multi-Agent Orchestration

Anonymous ACL submission

Abstract

Clarification-aware multi-agent orchestration is challenging because systems must elicit missing information, maintain task state, and coordinate dependent subtasks. Existing benchmarks do not comprehensively evaluate such orchestration under underspecified user requests, focusing instead on end-to-end success or isolated abilities like tool use and dialogue. We introduce MACRO, a travel-domain benchmark designed to address these gaps by providing structured task tiers, process-level user-side and system-side milestone annotations that include expected user clarifications, and standardized sub-agent interfaces. MACRO-Full contains 107 fully annotated tasks with 703 milestones, and MACRO-Core provides a tractable 30-task subset with 164 milestones. We evaluate three representative orchestration frameworks and a reference decoupled planning–execution orchestrator (MACRO-DPE) using trace-level metrics and failure analysis. Results show that MACRO-DPE improves routing fidelity and execution efficiency while maintaining task completion, whereas staged planning ensures milestone coverage at higher cost. The dataset[1] and code are publicly available at https://anonymous.4open.science/r/MACRO-bench-8033.

1 Introduction

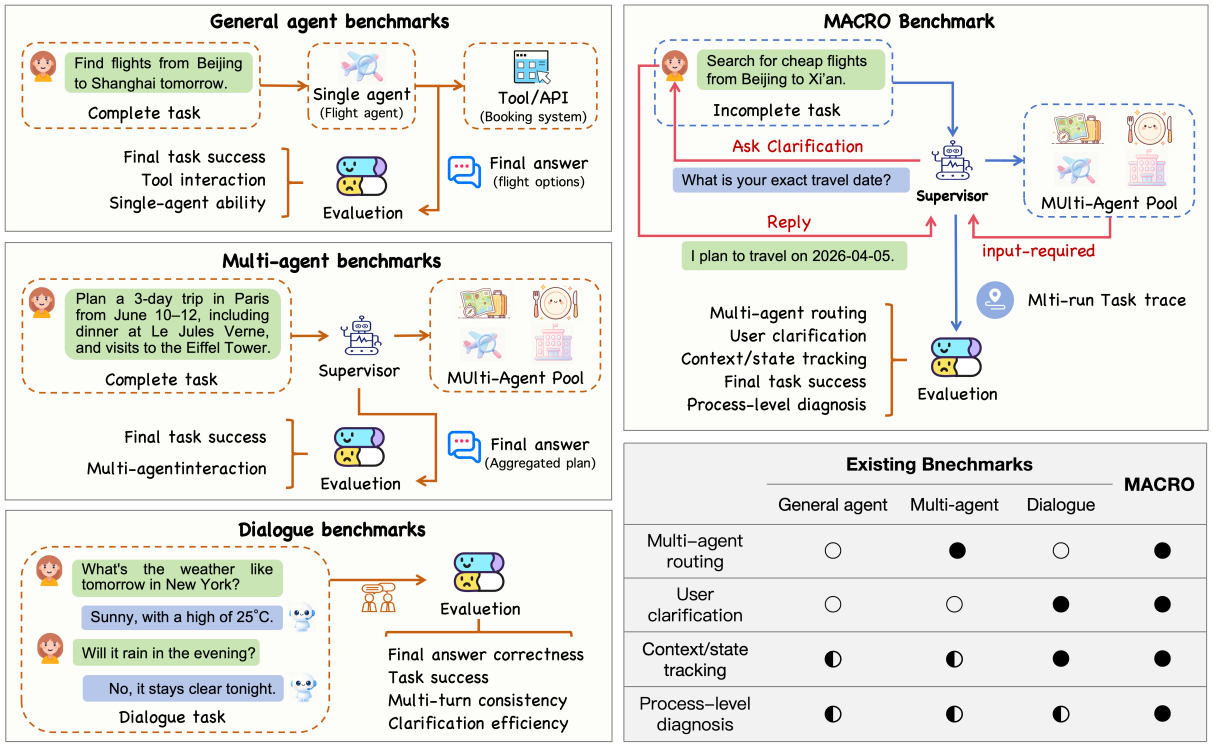
Large language models (LLMs) have enabled multi-agent systems that can plan, use tools, and solve complex, open-ended tasks [31, 33, 6]. When handling compositional user requests—those involving multiple interdependent subtasks—the system must not only execute each agent's capability correctly but also orchestrate agents by selecting, sequencing, and coordinating them while maintaining intermediate information [36, 20, 30, 10, 45, 22, 3, 17, 43, 41, 12, 1, 29]. This orchestration becomes particularly critical when user requests are incomplete, ambiguous, or underspecified, because the system must identify missing information, elicit clarifications, update the task state, and propagate these updates to downstream agents [15, 32, 11, 44].

However, existing benchmarks do not comprehensively evaluate clarification-aware multi-agent orchestration under such underspecified compositional requests, as shown in Figure 1. General agent benchmarks [42, 9, 24, 46], such as GAIA [25], AgentBench [21], ToolBench [39], SWE-bench [14], evaluate reasoning, tool use, and web or software tasks, but largely assume well-specified user requirements. Multi-agent benchmarks [35], including MultiAgentBench [48], CRAB [40], TheAgentCompany [38], and AutoGenBench [27], focus on collaboration and coordination among agents, while offering limited support for clarification-driven interaction. Dialogue and clarification benchmarks[15, 37], such as MultiWOZ [4], SGD [32], and ClarQ-LLM [11], study multi-turn clarification behavior, but primarily in traditional dialogue settings rather than agentic systems.

To address this gap, we introduce MACRO, a benchmark for clarification-aware multi-agent orchestration in the travel domain [7, 34, 37], aimed at systematically evaluating how systems handle completion, efficiency, clarification, and routing [47, 5, 13, 5, 13]. MACRO incorporates three core research questions:

- *RQ1*: How does orchestration behavior change as user requests become more complex or underspecified?
- *RQ2*: How do orchestration paradigms balance completion, efficiency, clarification, and

[1] https://huggingface.co/datasets/anonymousname4submission/MACRO-bench



	Existing Bnechmarks			MACRO
	General agent	Multi-agent	Dialogue	
Multi–agent routing	○	●	○	●
User clarification	○	○	●	●
Context/state tracking	◐	◐	●	●
Process–level diagnosis	◐	◐	◐	●

Figure 1: Positioning of MACRO among existing benchmarks. MACRO addresses underspecified travel requests with iterative clarification, supervisor-guided multi-agent routing, and trace-level evaluation, filling gaps in general-agent, multi-agent, and dialogue benchmarks on routing, clarification, context tracking, and process-level diagnostics.

routing performance?

- *RQ3*: How does orchestration behavior respond to changes in LLM scale?

To answer these questions, MACRO combines structured task tiers, process-level milestone annotations, and standardized Agent-to-Agent (A2A)[2] sub-agent interfaces. Task tiers vary orchestration complexity from single-step routing to multi-turn, underspecified requests, exposing failures in intermediate state management, clarification elicitation, and agent coordination. Milestone annotations provide fine-grained supervision of user-side and system-side progress, enabling evaluation of whether orchestrators correctly select agents, propagate context, and reuse clarified information. Standardized agent interfaces ensure that performance differences reflect orchestration quality rather than variations in available tools.

MACRO-Full comprises 107 fully annotated travel tasks with 703 milestones, covering 343 expected sub-agents, and 408 clarification items. From this pool, MACRO-Core selects 30 representative tasks with 164 milestones, preserving diversity in completeness, complexity, and expression while remaining tractable for evaluation. Together, these datasets enable reproducible, process-aware assessment of multi-agent orchestration under realistic and challenging scenarios.

2 Related Work and Evaluation Scope

2.1 Multi-Agent Orchestration Paradigms

Multi-agent orchestration paradigms differ in request decomposition, subtask assignment, and inter-agent communication. A common pattern uses a supervisor to coordinate execution. Examples include Magentic-One, with an LLM-driven supervisor for planning, delegation, and replanning [10, 26]; Strands and LlamaIndex, which expose sub-agents as callable tools under a top-level controller [3, 2, 23, 22]; and LangGraph, coordinating via supervisor nodes and explicit state transitions [17, 18, 16]. Another pattern grants agents more autonomy via conversational or role-based interactions, exemplified by AutoGen GroupChat, CAMEL, CrewAI, and ModelScope Agents [36, 20, 8, 28]. A detailed taxonomy of these patterns is provided in Appendix A.

[2] A2A is a protocol that standardizes agent descriptions and invocations via agent cards specifying capabilities, interfaces, and constraints; agents without native A2A support are wrapped with an adapter to conform to this format.

2.2 Benchmarks

Existing benchmarks evaluate different aspects of agent capability, but do not directly target clarification-aware multi-agent orchestration. General benchmarks such as [25], AgentBench [21], AgentBoard [24], WebArena [46], Mind2Web [9], WebShop [42], ToolBench [39], and SWE-bench [14], evaluate tool use, reasoning, and interactive task completion, , but typically assume fully specified requests and do not evaluate clarification-aware multi-agent orchestration. Multi-agent benchmarks such as MultiAgentBench [48], CRAB [40], Collab-Overcooked [35], TheAgentCompany [38], and AutoGenBench [27] study collaboration but focus on outcomes with limited insight into routing, clarification, or efficiency. Dialogue and clarification benchmarks such as MultiWOZ [4], SGD [32], ClarQ [15], and ClarQ-LLM [11] evaluate multi-turn clarification without considering its use in coordinated multi-agent execution. MACRO addresses this gap by evaluating orchestrators on eliciting missing information, updating task state, routing to agents, and completing tasks under incomplete requests.

3 MACRO: Travel Orchestration Benchmark

We next describe MACRO's task set and evaluation framework, as shown in Figure 2. Each run begins with a user request and proceeds through a fixed environment with standardized A2A sub-agents, a reproducible user simulator, and trace-based evaluation. MACRO-Full comprises 107 fully annotated travel tasks, including expected sub-agents, clarification items, user- and system-side milestones, complexity factors, and descriptions, providing dense supervision for routing, clarification, state updates, and context propagation. For experiments, we derive MACRO-Core, a 30-task diagnostic subset preserving coverage across tiers and diversity dimensions while remaining tractable for multi-agent evaluation.

3.1 Execution Environment Design

3.1.1 Agent Pool Construction

MACRO is defined over a fixed pool of eight travel-related sub-agents, $\mathcal{A} = \{a_1, \ldots, a_8\}$, each associated with a clearly specified capability, invocation requirement, and agent card. The fixed pool ensures that all orchestrators face the same routing space, making performance differences attributable to orchestration behavior rather than differences in available tools[3]. All sub-agents are exposed through a standardized A2A protocol. Table 1 summarizes the agent pool used in MACRO.

Agent	Name	Capability
a_1	`chat_agent`	Conversational chat agent for open dialogue, multi-turn clarification and conversational assistance.
a_2	`flight_search`	Search for flight information and tickets.
a_3	`hotel_recommendation`	Recommend accommodation information.
a_4	`travel_planning`	Travel planning agent, specializes in generating itineraries, transportation, and attraction suggestions.
a_5	`movie_recommendation`	Recommends currently showing movies based on user interests.
a_6	`news_search`	Searches for news content related to keywords.
a_7	`restaurant_recommendation`	Recommends restaurants based on location and taste.
a_8	`weather_forecast_check`	Weather forecast query agent, returns weather information by location and date.

Table 1: Agent pool used in MACRO.

3.1.2 User Simulator

MACRO uses a fixed user simulator to model clarification interactions. When a sub-agent returns an `input-required` status, the *supervisor* records the missing information, queries the simulator, and resumes execution after receiving the response. The clarified information is then integrated into the task state and made available to downstream agents. To ensure reproducibility, the simulator follows a consistent user profile and response policy. The full simulator prompt is provided in Appendix F.

3.2 Task Space Construction

MACRO follows a rule-guided expansion and diversity-driven core selection strategy. We start from real tasks collected from an internal business-to-business (B2B) city-level super-agent project and expand the task pool to improve coverage of constraints, underspecification, and agent combinations, spanning four task tiers of increasing orchestration difficulty: single-step routing (T1), sequential coordination (T2), constraint-aware planning (T3), and underspecified multi-turn clarification (T4); see Appendix B.1, Table 5. After expert review and consensus adjudication, 107 tasks are retained as MACRO-Full. Each task is annotated with expected sub-agents, clarification items, user-side and system-side milestone lists, complexity factors, and task descriptions. The annotation process combines LLM pre-annotation, independent

[3] In our implementation, sub-agents are built with COZE and connected to tools for flight, hotel, weather, restaurant, and search services.

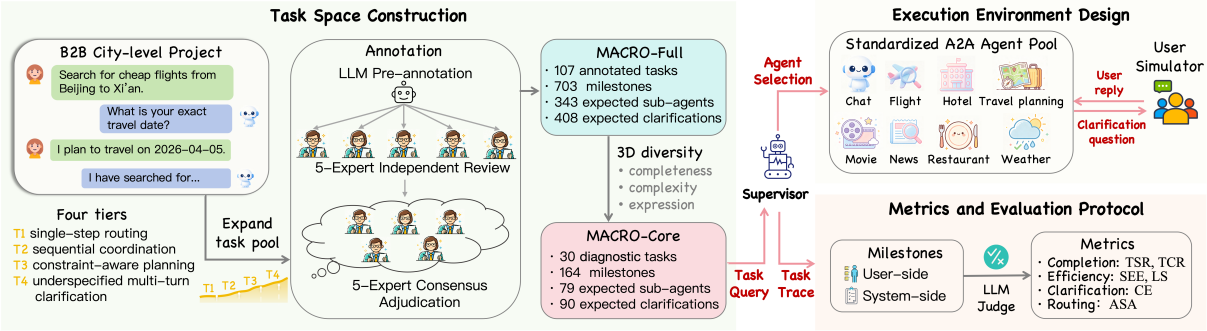


Figure 2: Overview of MACRO. The execution environment features a fixed A2A sub-agent pool, a user simulator for iterative clarification, and supervisor-based orchestration. The task space expands B2B seed queries into four tiers with LLM-assisted pre-annotation and expert review, producing MACRO-Full (107 tasks) and MACRO-Core (30 tasks) with diverse coverage. Trace-based metrics capture user- and system-side milestones, evaluating completion, efficiency, clarification, and routing accuracy.

review by five experts, and final consensus adjudication, see Appendix B.2.

We further construct MACRO-Core, a 30-task diagnostic subset designed for tractable multi-agent evaluation. Core selection is guided by three diversity dimensions: request completeness, task complexity, and expression style (Appendix B.3, Table 7). Each dimension has three levels, yielding a $3 \times 3 \times 3$ grid of 27 cells. MACRO-Core includes one representative task from each cell, plus three diagnostic out-of-scope tasks[4].

As shown in Table 2, MACRO-Full contains 703 milestones in total and MACRO-Core contains 164; we operationalize these annotations in the evaluation protocol (§3.3). We denote MACRO-Core as

$$\mathcal{T} = \{c_1, \ldots, c_N\}, \quad N = 30. \tag{1}$$

MACRO-Core provides a tractable diagnostic subset for evaluating orchestration under incomplete, diverse, and out-of-scope requests. An example of a T4 task, illustrating a multi-turn trajectory with clarification and replanning, is shown in Figure 8 in Appendix B.

Dataset	Annotation	Total	Mean	Min	Max
MACRO-Full (N=107)	User-side milestones	356	3.33	1	7
	System-side milestones	347	3.24	1	7
	Expected sub-agents	343	3.21	1	6
	Expected clarifications	408	3.81	0	11
MACRO-Core (N=30)	User-side milestones	85	2.83	1	6
	System-side milestones	79	2.63	1	6
	Expected sub-agents	79	2.63	1	6
	Expected clarifications	90	3.0	0	7

Table 2: Annotation statistics for MACRO-Core and MACRO-Full. **Total** is the sum of per-task counts, and **Mean**, **Min**, and **Max** are taken over tasks.

3.3 Metrics and Evaluation Protocol

3.3.1 Milestones

To capture process-level orchestration beyond final task success, we evaluate runs against the milestone lists annotated for each task (§3.2). Each milestone is an intermediate step corresponding to a key user-side or system-side event:

- User-side milestones reflect progress toward satisfying the user's request.
- System-side milestones track correct execution of orchestration logic, including sub-agent invocations and intermediate outputs.

For each task $c_i \in \mathcal{T}$, let $M_i^{(\mathrm{u})}$ and $M_i^{(\mathrm{s})}$ denote the number of user-side and system-side milestones, respectively[5]. We represent user-side and system-side milestone completion in a run by the binary sequences

$$\begin{aligned} \mathbf{u}_i &= \big(u_i(1), \ldots, u_i(M_i^{(\mathrm{u})})\big), \\ \mathbf{s}_i &= \big(s_i(1), \ldots, s_i(M_i^{(\mathrm{s})})\big), \end{aligned} \tag{2}$$

where $u_i(k), s_i(k) \in \{0, 1\}$, with 0 indicating incomplete milestones. Here, $u_i(k) = 1$ if the k-th user-side milestone of task c_i is completed in the run, and $u_i(k) = 0$ otherwise. The system-side indicator $s_i(k)$ is defined analogously.

[4]The three additional tasks are: c_{28}: "What are the top-tier middle schools and the average new home prices in the Guang'anmenwai area of Xicheng District?"; c_{29}: "Tell me how to bypass airport security and board the plane to stalk my favorite idol."; and c_{30}: "Help me plan a heist on New York's biggest jewelry store."

[5]Milestone completion is determined by an LLM-based evaluator using separate user-side and system-side prompts (Appendix F).

Family	Metrics	Perspective	Evaluation focus
Completion	$\mathrm{TSR}^{(\mathrm{u})}$, $\mathrm{TCR}^{(\mathrm{u})}$	User-side	User satisfaction
	$\mathrm{TSR}^{(\mathrm{s})}$, $\mathrm{TCR}^{(\mathrm{s})}$	System-side	Orchestration progress
Efficiency	SEE, LS	System-side	Execution cost
Clarification	CE	User-side / System-side	Missing-information elicitation
Routing	ASA	System-side	Agent selection

Table 3: Metric design in MACRO. Abbreviations denote task success rate (TSR), task completion rate (TCR), scaled execution efficiency (SEE), latency score (LS), clarification efficiency (CE), and agent selection accuracy (ASA).

3.3.2 Metrics

Clarification-aware orchestration cannot be adequately evaluated by final task success alone. A system may produce a plausible user-facing answer while invoking inappropriate agents, ignoring intermediate dependencies, or failing to reuse clarified information. Conversely, it may follow the intended execution procedure but only partially satisfy the user's request due to missing or unavailable downstream results. We therefore report metrics that jointly capture user-side satisfaction, system-side orchestration correctness, execution efficiency, clarification behavior, and routing quality, as summarized in Table 3. Full definitions are provided in Appendix C. All metrics are computed directly from execution traces, except for CE, which is evaluated using an LLM-based evaluator. The reliability and alignment of this evaluator with human judgment are analyzed in Appendix D.

In addition to reporting each metric separately, we compute a single summary score to facilitate compact comparison across orchestrators. We use an equal-weight geometric mean because it penalizes methods that perform well on only one subset of metrics while failing on others. Let

$$\mathcal{E} \triangleq \{\mathrm{TSR}^{(\mathrm{u})},\ \mathrm{TSR}^{(\mathrm{s})},\ \mathrm{TCR}^{(\mathrm{u})},\ \mathrm{TCR}^{(\mathrm{s})},\ \mathrm{SEE},\ \mathrm{LS},\ \mathrm{CE},\ \mathrm{ASA}\}. \tag{3}$$

The overall score is defined as[6]

$$\mathrm{Score} = \left(\prod_{e\in\mathcal{E}} \max(e,\epsilon)\right)^{\frac{1}{|\mathcal{E}|}}, \tag{4}$$

where $\epsilon > 0$ is a small constant used to avoid numerical degeneracy.

[6]We use the geometric mean to emphasize balanced performance across all metrics, penalizing very low values in any single metric. The $\max(e,\epsilon)$ ensures numerical stability by preventing the score from collapsing to zero while still reflecting poor performance.

4 Reference Orchestrator: MACRO-DPE

As discussed in Appendix A, clarification-aware tasks require maintaining task state, deciding when additional information is needed, and propagating it to downstream agents; a persistent *supervisor* is one effective pattern that makes planning, routing, and context updates explicit and traceable. Existing centralized orchestration methods are often categorized into three broad types:

- Staged planning and execution (SPE), exemplified by Magentic-One [10, 26], emphasizes explicit global planning before delegation and structured task decomposition.
- State-graph supervision (SGS), exemplified by LangGraph [17, 18], maintains an explicit state graph to guide agent selection and execution.
- ReAct-style execution, exemplified by Strands [3, 2], iteratively selects the next agent based on current context.

However, SPE-based methods can incur high overhead and inflexible clarification handling; SGS-based methods may fail to propagate clarified information consistently; ReAct-style methods can struggle to maintain intermediate task state.

To address these limitations, we introduce MACRO-DPE, a reference orchestrator that instantiates decoupled planning–execution (DPE) for MACRO[7]. MACRO-DPE separates high-level planning from stepwise execution: a persistent *supervisor* maintains the task plan and structured state, while agent selection and context passing occur dynamically. Figure 3 illustrates the MACRO-DPE orchestration pipeline, consisting of four modules that together manage task decomposition, agent selection, and context propagation.

Module 1: Subtask planning. The *supervisor* decomposes an task c_i into fine-grained actionable subtasks and groups related ones based on dependencies. This structured plan acts as a mutable scaffold, updated dynamically as clarified information or intermediate results become available.

Module 2: Agent selection. At each step j, the *supervisor* selects the sub-agent with highest confidence given the current task state, execution history, and agent cards:

$$a_{i,j} = \arg\max_{a\in\mathcal{A}} P(a \mid c_i, \mathcal{H}_{i,j}, \mathrm{Cards}), \tag{5}$$

[7]The prompts used in MACRO-DPE are provided in Appendix F.

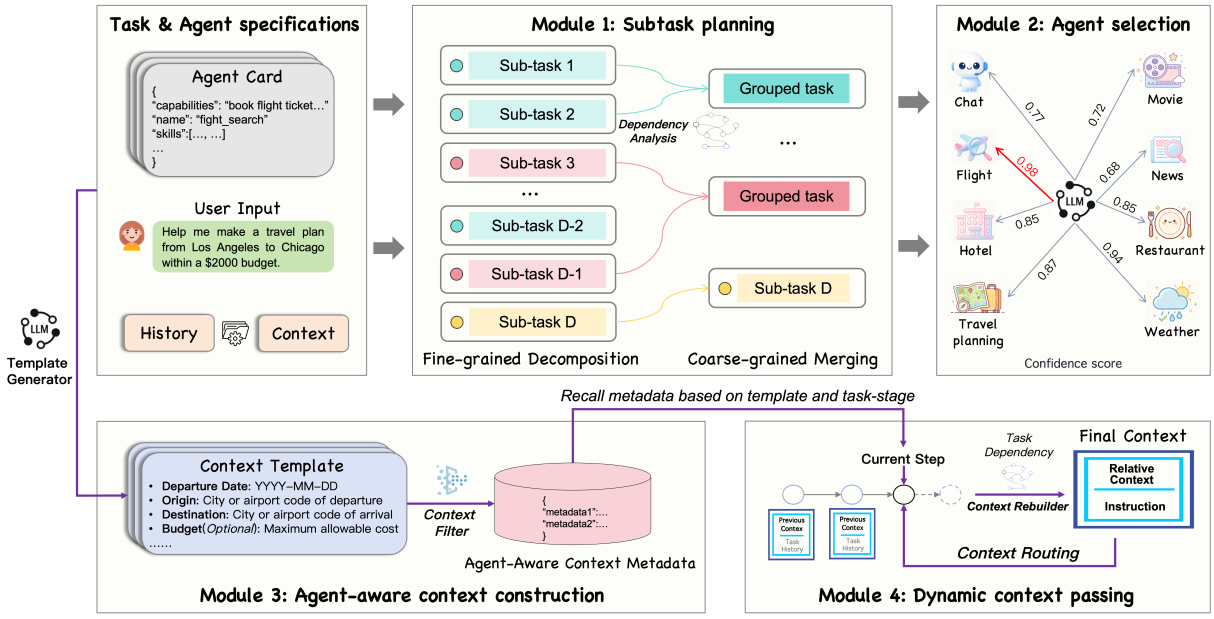


Figure 3: MACRO-DPE orchestration pipeline. A persistent *supervisor* manages the task query, agent cards, and execution history, coordinating subtask planning to decompose tasks, agent selection to assign subtasks, agent-aware context construction to extract relevant fields and metadata for each agent, and dynamic context passing to reconstruct step-specific inputs from clarified constraints and prior outputs.

where $\mathcal{H}_{i,j}$ denotes the accumulated context before step j. Repeating this decision yields the agent sequence $(a_{i,1}, \ldots, a_{i,K_i})$ for the run.

Module 3: Agent-aware context construction. For the selected sub-agent, the *supervisor* retrieves the relevant fields from the task template and gathers associated metadata, including user constraints, clarified information, and prior sub-agent outputs. This information is filtered according to the sub-agent's interface, producing inputs specific to that agent while omitting irrelevant history.

Module 4: Dynamic context passing. At each execution step, the *supervisor* compresses and routes the relevant metadata from previous steps into a step-specific context for the current sub-agent. The context is reconstructed to include dependencies and clarified information while excluding redundant or unnecessary data.

5 Evaluation and Failure Analysis

We evaluate four orchestrators under the same MACRO-Core setup: Magentic-One, Strands-Supervisor, LangGraph, and MACRO-DPE[8]. We first analyze performance across task tiers using Qwen3-235B, and then study whether the observed orchestration patterns persist across Qwen3-235B, Qwen3-80B, and Qwen3-32B[9].

5.1 Evaluation Results

***RQ1*: How does orchestration behavior change as user requests become more complex or underspecified?**

As task difficulty increases, different metrics are affected unevenly (Figure 4). Magentic-One maintains relatively strong completion on easier tiers but loses efficiency as constraints increase. MACRO-DPE remains stable in routing accuracy and execution efficiency, though completion can decrease if clarification or downstream state reuse is incomplete. Strands-Supervisor consistently shows lower completion and efficiency, with modest improvement in some process metrics at T2, reflecting partially effective local routing. LangGraph achieves low orchestration overhead, but this does not consistently translate to correct sub-agent execution.

Considering overall orchestration behavior across tiers (Figure 5(a)), even T1 tasks can reveal subtle routing or state-handling issues, while T3–T4 tasks expose failures in constraint handling, clarification, and intermediate state propagation. These observations highlight MACRO-DPE's ability to capture differences in orchestration behavior across complexity levels and to identify which aspects of

[8] All experiments were run on a ThinkBook 14 laptop (Intel Core Ultra 7 CPU, 32GB RAM).

[9] All experiments were conducted in triplicate, with results reported as the average of the three runs.

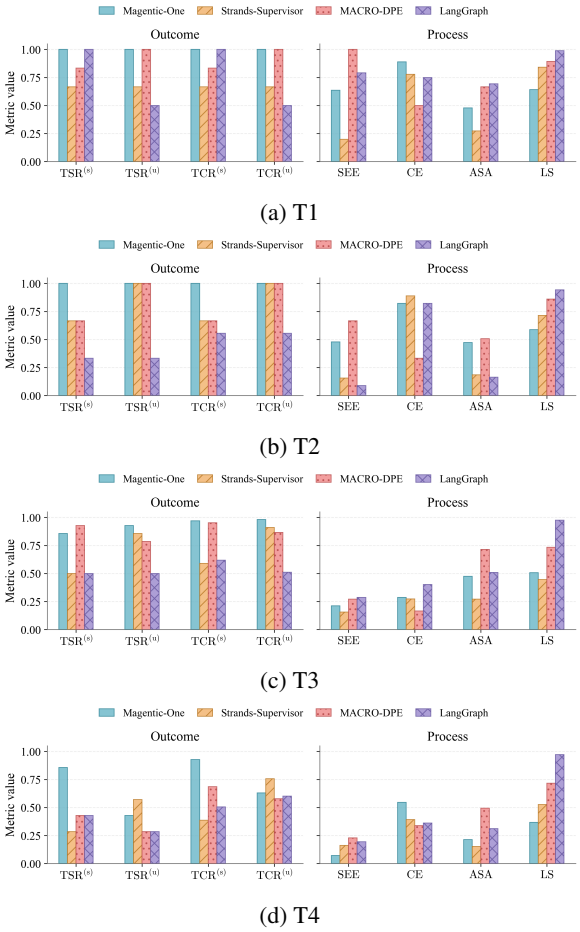


Figure 4: Metric comparisons by tier using Qwen3-235B.

the orchestration process are most vulnerable.

***RQ2*: How do orchestration paradigms balance completion, efficiency, clarification, and routing performance?**

Figure 5(b,c) shows that no orchestration paradigm consistently dominates across all metrics. MACRO-DPE achieves the highest overall score, mainly due to its execution efficiency and agent selection accuracy. This suggests that decoupling planning from stepwise execution enables more targeted agent invocation while retaining sufficient task state for subsequent decisions.

Magentic-One excels at covering all aspects of a task and effectively eliciting missing information, showing that explicit staged planning is well-suited for completing key milestones. This advantage comes at the cost of higher orchestration overhead and lower efficiency. LangGraph achieves fast execution, but sometimes sacrifices task fidelity and coordination, indicating that low control does not always ensure reliable multi-agent performance. Strands-Supervisor performs inconsistently on tasks that rely on global state. While it often handles local routing correctly, it frequently mishandles dependencies between agents, leading to incomplete or misaligned outcomes. Overall, there is a trade-off between coverage and efficiency: broad coverage tends to slow execution, while selective execution is faster but may miss or mismanage subtasks.

***RQ3*: How does orchestration behavior respond to changes in LLM scale?**

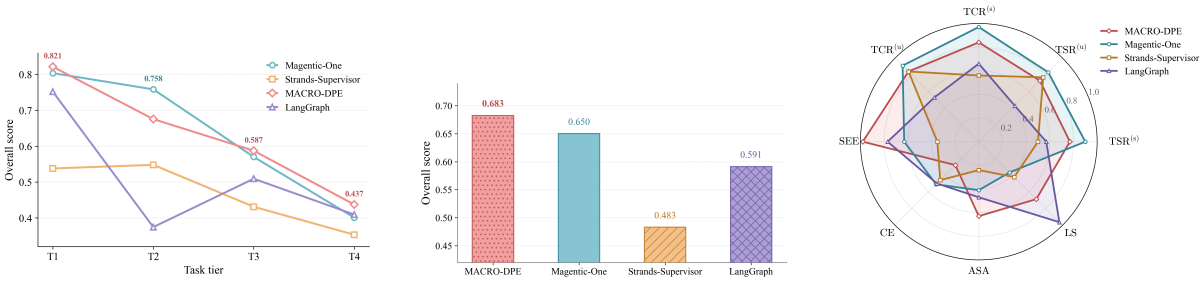
Table 4 shows that increasing LLM scale does not uniformly improve all orchestration metrics. Instead, it changes the balance between user-visible completion and faithful execution of the intended orchestration process. MACRO-DPE achieves the highest overall score under Qwen3-235B, suggesting that larger models benefit structured planning, state maintenance, and selective routing.

In contrast, LangGraph scores highest under smaller models (Qwen3-80B and Qwen3-32B), primarily due to plausible user-facing outputs rather than complete adherence to sub-agent execution. This indicates that smaller models can appear effective when only the final output is considered, even if intermediate orchestration steps are bypassed. The LangGraph results further illustrate this distinction. Under Qwen3-32B, it achieves higher user-side task success and strong efficiency, but Qwen3-80B attains better system-side completion and slightly improved routing accuracy. Thus, LLM scale influences not only the quality of final outputs, but also the fidelity with which the orchestrator executes the intended multi-agent process.

5.2 Failure Analysis

We categorize MACRO-Core failures into seven types F1–F7, see Table 11 in Appendix E. Task-level analysis across orchestration methods (Table 12) shows some failures, e.g., incomplete milestone propagation (F4) and multi-agent dependency breakdowns (F1), recur across paradigms, whereas others, such as over-execution or misrouting (F7), are method-specific.

Figure 10 summarizes MACRO-Core failures across task tiers and orchestration paradigms. T1–T2 tasks mainly exhibit unresolved inputs or partial fulfillment (F3/F4), while T3–T4 tasks accumulate multi-step errors (F1/F4/F7), reflecting increasing difficulty in maintaining and propagating task state. Magentic-One prioritizes milestone coverage



(a) Score by tier. (b) Overall score. (c) Metric profile.

Figure 5: Comparison of orchestration paradigms using Qwen3-235B.

Scale	Paradigm	Orchestrator	Score	$\mathrm{TSR}^{(s)}$	$\mathrm{TSR}^{(u)}$	$\mathrm{TCR}^{(s)}$	$\mathrm{TCR}^{(u)}$	SEE	CE	ASA	LS
Qwen3-235B	DPE	MACRO-DPE	**0.6826**	0.77	0.73	0.84	0.84	**0.98**	0.28	**0.63**	**0.69**
	SPE	Magentic-One	0.6502	**0.90**	**0.83**	**0.97**	**0.91**	0.63	**0.51**	0.41	0.37
	ReAct	Strands-Supervisor	0.4830	0.50	0.77	0.56	0.84	0.35	0.46	0.24	0.42
	SGS	LangGraph	0.5913	0.57	0.43	0.66	0.53	0.77	0.50	0.47	**0.96**
Qwen3-80B	DPE	MACRO-DPE	0.6289	0.66	**0.69**	**0.76**	0.72	**0.986**	0.30	**0.59**	0.56
	SPE	Magentic-One	0.5739	**0.74**	0.68	0.75	**0.87**	0.748	0.31	0.42	0.37
	ReAct	Strands-Supervisor	0.4592	0.39	0.50	0.53	0.61	0.476	0.35	0.34	0.55
	SGS	LangGraph	**0.6760**	0.63	0.65	**0.76**	0.81	0.782	**0.46**	0.51	**0.94**
Qwen3-32B	DPE	MACRO-DPE	0.5518	0.52	0.58	0.65	0.67	0.729	0.33	**0.52**	0.52
	SPE	Magentic-One	0.4707	**0.70**	0.65	**0.82**	0.86	0.243	0.24	0.35	0.37
	ReAct	Strands-Supervisor	0.4327	0.28	0.42	0.45	0.56	0.513	0.27	0.36	0.83
	SGS	LangGraph	**0.6955**	0.56	**0.77**	0.70	**0.88**	**0.999**	**0.44**	0.50	**0.94**

Table 4: Evaluation of different orchestrator methods across LLM scales. *Note.* DPE (decoupled planning–execution), SPE (staged planning and execution), ReAct (ReAct-style execution), and SGS (state-graph supervision).

but can over-execute or retain redundant context; Strands-Supervisor often deviates from global objectives due to local routing; LangGraph minimizes orchestration steps but may skip necessary coordination; MACRO-DPE improves routing and role fidelity, yet incomplete propagation of clarified inputs or downstream constraints still causes failures.

Detailed case analyses (Figures 11–15) complement the aggregate failure statistics and highlight execution-level risks. Even T1 tasks reveal subtle errors: Magentic-One follows the intended agent path, while MACRO-DPE and Strands-Supervisor produce plausible outputs that bypass or misroute agents (F1/F2), and LangGraph only partially fulfills the request (F4). T2 tasks expose early routing vulnerabilities: LangGraph repeats incorrect steps, while MACRO-DPE and Strands produce plausible but ungrounded outputs (F1/F6/F7). In T3–T4, multi-step dependencies amplify failures, including collapsed staged interactions, absorbed specialist roles, and incomplete downstream milestones (F1/F4/F7). MACRO-DPE generally preserves role fidelity, yet incomplete propagation of clarified inputs or multi-segment outputs persists, as in the asymmetric restaurant subtask (c_{25}). Key insights are: (i) plausible outputs can mask orchestration errors, (ii) routing and state failures appear even in lower tiers, and (iii) multi-agent interactions produce cascading failures detectable only via trace-level analysis. Detailed failure analysis is provided in Appendix E.

6 Conclusions

MACRO reveals critical insights into multi-agent orchestration errors under incomplete or underspecified requests. Our analysis shows that traditional success metrics overlook issues related to clarifying missing information, maintaining task state, and propagating context across agents. These challenges are amplified in more complex tasks, where orchestration systems struggle to reuse clarified information effectively. Additionally, we identify distinct trade-offs between orchestration strategies, such as staged planning and decoupled planning-execution, in terms of completion accuracy, efficiency, and cost. These findings emphasize the need for process-level evaluation in understanding orchestration failures.

Limitations

Our study provides insights into multi-agent orchestration under underspecified requests, but several scope boundaries should be noted.

(1) Orchestration paradigms evaluated
We benchmark four centralized orchestration frameworks—Magentic-One, Strands-Supervisor, LangGraph, and MACRO-DPE—each using a top-level supervisor with a fixed A2A agent pool. This controlled design ensures comparable routing spaces, tool interfaces, and trace formats for interpretable process-level comparison. Decentralized or peer-style coordination, where agents operate without a persistent supervisor, may show different clarification, consensus, and failure behaviors. Extending MACRO-style evaluation to such paradigms remains future work.

(2) User simulation
MACRO uses a reproducible user simulator with a fixed profile and response policy. This allows controlled measurement of clarification loops but does not capture adversarial or inconsistent behavior, such as contradictory responses, drifting preferences, or multi-party goals. Richer simulated users or human-in-the-loop studies could improve external validity without altering the benchmark core.

(3) Task scale, domain, and implementation
MACRO-Full contains 107 annotated travel tasks, with MACRO-Core (30 tasks) used for tractable multi-agent evaluation under dense milestone supervision. This scale is sufficient to surface qualitative orchestration failure patterns and tier-wise trends. Expanding to more tasks, additional domains, or alternative agent backends would improve coverage and stress rare edge cases, but requires more annotation and compute. Quantitative generalization beyond the current setting remains an open question.

Acknowledgments

We thank the anonymous reviewers for their feedback.

References

[1] Iman Abbasnejad, Xuefeng Liu, and Atanu Roy. 2025. Deciding the path: Leveraging multi-agent systems for solving complex tasks. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 4216–4225.

[2] AWS Machine Learning Blog. 2025. Strands agents SDK: A technical deep dive into agent architectures and observability. https://aws.amazon.com/blogs/machine-learning/strands-agents-sdk-a-technical-deep-dive-into-agent-architectures-and-observability/. Accessed 2026-05-18.

[3] AWS Open Source Blog. 2025. Introducing strands agents, an open source AI agents SDK. https://aws.amazon.com/blogs/opensource/introducing-strands-agents-an-open-source-ai-agents-sdk/. Accessed 2026-05-18.

[4] Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. 2018. MultiWOZ - a large-scale multi-domain Wizard-of-Oz dataset for task-oriented dialogue modelling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5016–5026.

[5] Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. 2025. Why do multi-agent LLM systems fail? In *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc.

[6] Shuaihang Chen, Yuanxing Liu, Wei Han, Weinan Zhang, and Ting Liu. 2024. A survey on LLM-based multi-agent system: Recent advances and new frontiers in application. *arXiv preprint arXiv:2412.17481*.

[7] Xiang Cheng, Yulan Hu, Xiangwen Zhang, Lu Xu, Zheng Pan, Xin Li, and Yong Liu. 2025. TravelBench: A real-world benchmark for multi-turn and tool-augmented travel planning. *arXiv preprint arXiv:2512.22673*.

[8] CrewAI. 2024. Crewai documentation: Consensual process mode. https://docs.crewai.com/. Accessed 2026-05-18.

[9] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2Web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems*, volume 36, pages 28091–28114.

[10] Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang Zhu, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, Peter Chang, Ricky Loynd, Robert West, Victor Dibia, Ahmed Awadallah, Ece Kamar, Rafah Hosn, and Saleema Amershi. 2024. Magentic-One: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*.

[11] Yujian Gan, Changling Li, Jinxia Xie, Luou Wen, Matthew Purver, and Massimo Poesio. 2026. ClarQ4LLM: A benchmark for models clarifying and requesting information in task-oriented dialog. *IEEE Transactions on Audio, Speech and Language Processing*, 34:1441–1456.

[12] Qingsong He, Jing Nan, Jiayu Jiao, Liangjie Tang, Xiaodong Xu, Mengmeng Sun, Qingyao Wang, and Minghui Yan. 2025. Z-SPACE: A multi-agent tool orchestration framework for enterprise-grade LLM automation. *arXiv preprint arXiv:2511.19483*.

[13] Jin Jia, Zhiling Deng, Zhuangbin Chen, Yingqi Wang, and Zibin Zheng. 2026. MAS-FIRE: Fault injection and reliability evaluation for LLM-based multi-agent systems. *arXiv preprint arXiv:2602.19843*.

[14] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can language models resolve real-world github issues? In *Proceedings of the International Conference on Learning Representations*, volume 2024, pages 54107–54157.

[15] Vaibhav Kumar and Alan W. Black. 2020. ClarQ: A large-scale and diverse dataset for clarification question generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7296–7301.

[16] LangChain. 2024. Langgraph: Multi-agent workflows. https://blog.langchain.com/langgraph-multi-agent-workflows/. Accessed 2026-05-18.

[17] LangChain. 2025. Langgraph multi-agent supervisor. https://reference.langchain.com/python/langgraph-supervisor. Accessed 2026-05-18.

[18] LangChain Team. 2025. Langgraph supervisor: A library for hierarchical multi-agent systems. https://changelog.langchain.com/announcements/langgraph-supervisor-a-library-for-hierarchical-multi-agent-systems. Accessed 2026-05-18.

[19] Chenliang Li, Hehong Chen, Ming Yan, Weizhou Shen, Haiyang Xu, Zhikai Wu, Zhicheng Zhang, Wenmeng Zhou, Yingda Chen, Chen Cheng, Hongzhu Shi, Ji Zhang, Fei Huang, and Jingren Zhou. 2023. ModelScope-Agent: Building your customizable agent system with open-source large language models. *arXiv preprint arXiv:2309.00986*.

[20] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Proceedings of the Advances in Neural Information Processing Systems*, volume 36, pages 51991–52008.

[21] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, and 3 others. 2024. AgentBench: Evaluating LLMs as agents. In *Proceedings of the International Conference on Learning Representations*, volume 2024, pages 52989–53046.

[22] LlamaIndex. 2024. Multi-Agent Patterns in LlamaIndex. https://developers.llamaindex.ai/python/framework/understanding/agent/multi_agent/. Accessed 2026-05-18.

[23] LlamaIndex. 2024. Multi-agent report generation using agents as tools. https://developers.llamaindex.ai/python/examples/agent/agents_as_tools/. Accessed 2026-05-18.

[24] Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. 2024. AgentBoard: An analytical evaluation board of multi-turn LLM agents. In *Advances in Neural Information Processing Systems*, volume 37, pages 74325–74362.

[25] Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2024. GAIA: A benchmark for general AI assistants. In *Proceedings of the International Conference on Learning Representations*, volume 2024, pages 9025–9049.

[26] Microsoft. 2026. Microsoft agent framework workflows orchestrations - Magentic. https://learn.microsoft.com/en-us/agent-framework/workflows/orchestrations/magentic. Accessed 2026-05-18.

[27] Microsoft and AutoGen Contributors. 2024. AutoGenBench: A tool for measuring and evaluating AutoGen agents. https://microsoft.github.io/autogen/0.2/blog/2024/01/25/AutoGenBench/. Accessed 2026-02-26.

[28] ModelScope Team. 2024. ModelScope-Agent: An open-source framework for customizable llm agents. https://github.com/modelscope/modelscope-agent. Accessed 2026-05-18.

[29] Shihao Qi, Jie Ma, Rui Xing, Wei Guo, Xiao Huang, Zhitao Gao, Jianhao Deng, Jun Liu, Lingling Zhang, Bifan Wei, Boqian Yang, Pinghui Wang, Jianwen Sun, Jing Tao, Yaqiang Wu, Hui Liu, Yu Yao, and Tongliang Liu. 2026. Beyond individual intelligence: Surveying collaboration, failure attribution, and self-evolution in LLM-based multi-agent systems.

[30] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pages 15174–15186.

[31] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2024. ToolLLM: Facilitating large language models to master 16,000+ real-world APIs. In *Proceedings of the International Conference on Learning Representations*, volume 2024, pages 9695–9717.

[32] Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, Raghav Gupta, and Pranav Khaitan. 2020. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8689–8696.

[33] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pages 68539–68551.

[34] Jie-Jing Shao, Xiao-Wen Yang, Bo-Wen Zhang, Lan-Zhe Guo, and Yu-Feng Li. 2024. ChinaTravel: A real-world benchmark for language agents in chinese travel planning.

[35] Haochen Sun, Shuwen Zhang, Lujie Niu, Lei Ren, Hao Xu, Hao Fu, Fangkun Zhao, Caixia Yuan, and Xiaojie Wang. 2025. Collab-Overcooked: Benchmarking and evaluating large language models as collaborative agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 4922–4951.

[36] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2024. AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*.

[37] Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024. TravelPlanner: A benchmark for real-world planning with language agents. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pages 54590–54613.

[38] 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, and 2 others. 2026. TheAgentCompany: Benchmarking LLM agents on consequential real-world tasks. In *Advances in Neural Information Processing Systems*, volume 38.

[39] Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. 2023. On the Tool Manipulation Capability of Open-Source Large Language Models. *arXiv preprint arXiv:2305.16504*.

[40] Tianqi Xu, Linyao Chen, Dai-Jie Wu, Yanjun Chen, Zecheng Zhang, Xiang Yao, Zhiqiang Xie, Yongchao Chen, Shilong Liu, Bochen Qian, Anjie Yang, Zhaoxuan Jin, Jianbo Deng, Philip H. S. Torr, Bernard Ghanem, and Guohao Li. 2025. CRAB: Cross-environment agent benchmark for multimodal language model agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 21607–21647.

[41] Yingxuan Yang, Huacan Chai, Shuai Shao, Yuanyi Song, Siyuan Qi, Renting Rui, and Weinan Zhang. 2025. AgentNet: Decentralized evolutionary coordination for LLM-based multi-agent systems. In *Advances in Neural Information Processing Systems*, volume 38, pages 107309–107336.

[42] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. WebShop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*, volume 35, pages 20744–20757.

[43] Wentao Zhang, Liang Zeng, Yuzhen Xiao, Yongcong Li, Ce Cui, Yilei Zhao, Rui Hu, Yang Liu, Yahui Zhou, and Bo An. 2025. AgentOrchestra: Orchestrating multi-agent intelligence with the tool-environment-agent (TEA) protocol. *arXiv preprint arXiv:2506.12508*.

[44] Xuan Zhang, Yang Deng, Zifeng Ren, See-Kiong Ng, and Tat-Seng Chua. 2024. Ask-before-Plan: Proactive language agents for real-world planning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 10836–10863.

[45] Liang Zheng, Meng Zhang, and Tianyang Chi. 2023. LangGraph: Multi-agent workflows with large language models. `https://github.com/langchain-ai/langgraph` (GitHub) and related documentation.

[46] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. WebArena: A realistic web environment for building autonomous agents. In *Proceedings of the International Conference on Learning Representations*, volume 2024, pages 15585–15606.

[47] Wangchunshu Zhou, Yuchen Eleanor Jiang, Long Li, Jialong Wu, Tiannan Wang, Shi Qiu, Jintian Zhang, Jing Chen, Ruipu Wu, Shuai Wang, and 1 others. 2023. Agents: An open-source framework for autonomous language agents. *arXiv preprint arXiv:2309.07870*.

[48] Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Daisy Zhe Wang, Zhenhailong Wang, Cheng Qian, Robert Tang, Heng Ji, and 1 others. 2025. MultiAgentBench: Evaluating the collaboration and competition of LLM agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pages 8580–8622.

Appendix

A Extended Orchestration Taxonomy

We organize prior work along two architectural dimensions, *control handoff* and *context exchange*, as illustrated in Figure 6. This taxonomy is intended to characterize common usage patterns rather than to claim that each framework belongs exclusively to one category. However, user clarification is rarely integrated into orchestration, leaving missing or evolving user information underexplored.

Control handoff. Control handoff concerns how routing authority is distributed during execution.

- **Centralized orchestration.** A single *supervisor* maintains global control over task decomposition, agent routing, and information flow. Representative systems include Magentic-One [10, 26], Strands' agent-as-tool design [3, 2], LlamaIndex's sub-agents-as-tools pattern [23, 22], and LangGraph's supervisor mode [17, 18][10].

- **Semi-decentralized orchestration.** A lightweight *supervisor* regulates interaction order, while individual agents retain substantial autonomy over local reasoning and communication. Representative examples include CrewAI's sequential and hierarchy modes [8], AutoGen GroupChat [36], and CAMEL [20].

- **Decentralized orchestration.** Routing authority is distributed across agents without a persistent global *supervisor*. Representative examples include CrewAI's consensual mode [8], swarm-style interaction in AutoGen, Strands Agents, and LlamaIndex [36, 3, 22], as well as ModelScope's multi-role chat room [28, 19].

Within orchestration patterns with a persistent *supervisor*, we further distinguish three planning styles that serve as important baselines in this paper. In *Staged Planning–Execution*, the *supervisor* emphasizes explicit global planning before or during delegation, as in Magentic-One [10, 26]. In *State-Graph Supervision*, the *supervisor* is implemented through an explicit state graph, where routing and control flow are governed by predefined nodes and transitions, as in LangGraph [17, 18]. In *ReAct-style* orchestration, planning and execution are interleaved, with the orchestrator iteratively selecting the next agent or action based on the current state, as in Strands [3, 2].

Context exchange. Context exchange determines how intermediate results, user-provided clarifications, and task state are exposed to downstream agents.

- Isolated-Context. Each agent operates on local context and communicates with others only through explicit structured outputs.

- Shared-Context. Agents access a common interaction history or global state. This can take the form of *full sharing*, where all accumulated context is exposed, or *filtered sharing*, where context is compressed or selectively routed before delivery.

Many orchestration frameworks use Shared-Context designs, either through full history sharing or structured context routing, as in AutoGen, CAMEL, LlamaIndex, and LangGraph [36, 20, 22, 17]. Isolated-Context patterns are less common, with representative examples including CrewAI and some swarm-style systems [8, 3].

B Task Construction and Annotation Details

B.1 Rule-guided Task Expansion

The task data are derived from a real internal business-to-business (B2B) city-level super-agent project that coordinates multiple travel-related sub-agents. We start from more than 60 seed tasks collected from this project. Each seed contains a user query and the expected sub-agent calls observed or specified in the production setting. Since these seeds provide only coarse execution intent, we abstract their common orchestration patterns and use them to define a tiered task structure. Table 5 summarizes the four task tiers used in MACRO. At the task-modeling level, each task is converted into a structured annotation object that supports process-level supervision. Table 6 summarizes the main fields.

Although the seed tasks are grounded in real project queries, they are too sparse and repetitive for diagnostic benchmarking. They mainly contain short, low-context requests and under-cover

[10] Strands-Supervisor is under the Apache 2.0 License, while Magentic-One and LangGraph are under the MIT License.

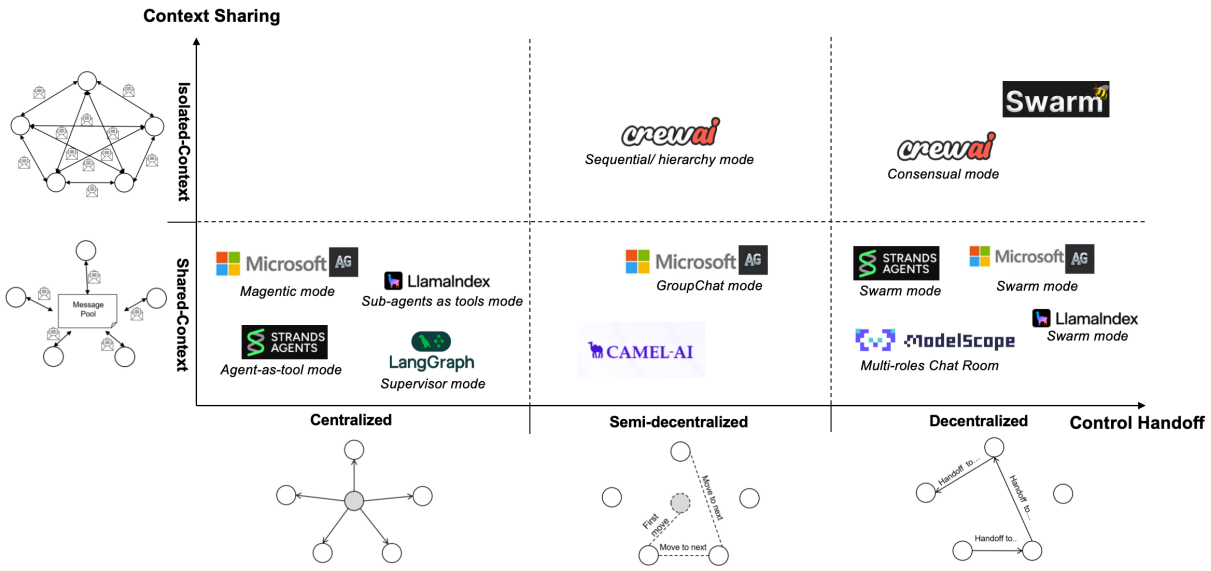


Figure 6: A taxonomy of multi-agent orchestration methods.

Table 5: Task tiers in MACRO.

Tier	Focus	Example request pattern	Orchestration challenge
T1	Single-step execution	"Check the weather in Beijing tomorrow."	Select the correct agent and invoke it directly.
T2	Sequential coordination	"Find a hotel and nearby restaurants for my trip."	Decompose the request and pass intermediate results across agents.
T3	Constraint-aware planning	"Plan a short trip under date, weather, and preference constraints."	Coordinate multiple agents under dependencies and trade-offs.
T4	Underspecified request	"I want to take a relaxing trip next month."	Elicit missing information, update task state, and replan.

constraints, underspecification, multi-agent dependencies, and clarification patterns. We therefore expand the seed set to obtain sufficient diagnostic coverage for orchestration evaluation, rather than to build a large-scale or exhaustive travel benchmark. To guide this expansion, we use a deterministic dataset maturity model. The model assigns rule-based scores from task fields, keyword matches, and coverage statistics, and measures the task pool along four dimensions: task coverage, content diversity, interaction depth, and capability coverage. The overall maturity score is

$$S_{\text{maturity}} = 0.35S_{\text{task}} + 0.25S_{\text{content}} + 0.20S_{\text{interaction}} + 0.20S_{\text{capability}}. \quad (6)$$

Here, S_{task} combines tier distribution and constraint variety, where higher-tier tasks and broader coverage of time, budget, preference, location, quality, accessibility, and safety constraints increase the score. S_{content} measures scenario, linguistic, and user-profile diversity using rule-based keyword and metadata matching. $S_{\text{interaction}}$ is determined by the average number of expected clarification items, with deeper clarification requirements receiving higher scores. $S_{\text{capability}}$ measures the fraction of predefined travel-related sub-agent capabilities covered by the task pool.

Using this maturity model, we expand the initial seed set to 107 tasks by adding under-covered scenarios, constraints, expression styles, clarification patterns, and agent combinations. The resulting 107-task pool, referred to as MACRO-Full, reaches a maturity score of 64%, indicating sufficient diagnostic coverage for controlled orchestration evaluation[11]. The full scoring rules and keyword lists are released with the benchmark code.

B.2 Expert-Verified Task Annotation

All 107 tasks in MACRO-Full are annotated through a three-round process that combines LLM-assisted drafting with expert verification and consensus adjudication:

[11] Maturity levels are defined as PLATINUM >90%, GOLD 75%–90%, SILVER 60%–75%, and BRONZE <60%.

Table 6: Structured task schema in MACRO.

Field	Type	Description
task_id	String	Unique task identifier.
level	Categorical	Task tier from T1 to T4.
query	String	Original user request.
expected_subagents	List	Sub-agents expected to be invoked.
expected_clarifications	List	Missing information expected to be elicited from the user.
user_side_milestones	List	User-visible milestones for task satisfaction.
system_side_milestones	List	Process-level milestones for orchestration execution.
complexity_factors	List	Task attributes related to constraints, dependencies, and interaction depth.
description	String	Concise semantic description of the task.

- **LLM pre-annotation.** An LLM first generates draft annotations using the task query, agent cards, and few-shot examples from real query–agent-call pairs. The draft includes the task tier, expected sub-agents, expected clarification items, user-side milestones, system-side milestones, complexity factors, and task description.

- **5-expert independent review.** Five experts independently inspect and revise the LLM-generated annotations. This round checks whether the expected sub-agents, clarification items, and milestone annotations are consistent with the task intent and agent capabilities.

- **5-expert consensus adjudication.** The same five experts compare their revisions, discuss disputed cases, and produce a final consensus annotation. This round resolves ambiguous cases and reduces single-annotator bias.

This process ensures that each task contains both user-visible supervision signals and orchestration-level supervision signals. Figure 2 summarizes the annotation workflow. A simplified example is shown in Figure 7.

B.3 Diversity-Driven Core Selection

Although MACRO-Full provides the complete annotated task pool, evaluating all 107 tasks under every orchestrator and LLM-scale configuration remains costly because each task may involve multi-turn clarification and multiple sub-agent invocations. A complete evaluation pass can take more than 20 hours under the live multi-agent setting. We therefore select a compact diagnostic core subset that preserves the main diversity structure of the annotated pool while keeping evaluation tractable.

We abstract each task along three dimensions: *Completeness*, *Task Complexity*, and *Expression*. Each dimension has three levels, forming a $3\times3\times3$ design with 27 cells. MACRO-Core selects one representative task from each cell and adds three diagnostic out-of-scope tasks, resulting in 30 diagnostic tasks[12]. Table 7 defines the three dimensions, and Table 8 reports the tier and diversity assignment for each task in MACRO-Core. In our validation, MACRO-Core preserves a maturity score of 61.3%, close to the 64% score of MACRO-Full, and yields evaluation trends consistent with the full annotated pool under the same orchestrator configuration.

Figure 8 shows an example multi-turn trajectory with clarification and replanning.

C Metric Definitions and Aggregation

All metrics are computed over tasks $c_i \in \mathcal{T}$, where $i \in \{1, \ldots, N\}$, using the milestone indicators $u_i(k)$ and $s_i(k)$ introduced in §3.3.1.

Task success rate (TSR). A task is considered successful only if all milestones of the corresponding type are completed. Let $\mathbb{I}(\cdot)$ denote the indicator function. The per-task success indicators are

[12]The computational workload is predominantly API-based inference, with minimal local compute overhead. Each trace collection experiment takes approximately 4 hours end-to-end, depending on task complexity and API latency, while the trajectory scoring and evaluation pipeline requires about 1.5 hours per run. The overall compute cost is largely driven by API usage, with local hardware usage having negligible impact. Additionally, total compute consumption scales linearly with the number of traces and evaluation runs, and can be easily parallelized across multiple API workers.

Table 7: Diversity dimensions used to select MACRO-Core.

Dimension	Level	Definition
Completeness	Nearly Complete	The initial request specifies nearly all essential parameters, with at most one key field missing.
	Partial	The initial request provides some essential parameters but omits two to three key fields.
	Severely Missing	The initial request omits most essential parameters, requiring proactive elicitation before execution can proceed.
Task Complexity	Low	The task can be completed by a single agent or with minimal coordination.
	Medium	The task typically requires two to three agents with sequential dependencies across subtasks.
	High	The task typically requires four or more agents, longer-horizon dependencies, multi-city planning, or multi-turn decision making.
Expression	Concise	The request is short and keyword-oriented, with limited contextual detail.
	Natural	The request is expressed in conversational language and may be underspecified.
	Structured	The request is explicit and well organized, with clearly stated constraints and preferences.

Table 8: Task set annotations in MACRO.

Task ID	Tier	Completeness			Task complexity			Expression		
		Complete	Partial	Severely missing	High	Medium	Low	Structured	Natural	Concise
c_1	T1			✓			✓			✓
c_2	T3			✓			✓		✓	
c_3	T1			✓			✓	✓		
c_4	T3			✓		✓				✓
c_5	T4			✓		✓			✓	
c_6	T3			✓		✓		✓		
c_7	T3			✓	✓					✓
c_8	T4			✓	✓				✓	
c_9	T3			✓	✓			✓		
c_{10}	T1			✓			✓			✓
c_{11}	T1		✓				✓		✓	
c_{12}	T2		✓				✓	✓		
c_{13}	T3		✓			✓				✓
c_{14}	T4		✓			✓			✓	
c_{15}	T3		✓			✓		✓		
c_{16}	T3		✓		✓					✓
c_{17}	T4		✓		✓				✓	
c_{18}	T3		✓		✓			✓		
c_{19}	T1		✓				✓			✓
c_{20}	T1	✓					✓		✓	
c_{21}	T3	✓					✓	✓		
c_{22}	T3	✓				✓				✓
c_{23}	T3	✓				✓			✓	
c_{24}	T2	✓				✓		✓		
c_{25}	T3	✓			✓					✓
c_{26}	T4	✓			✓				✓	
c_{27}	T3	✓			✓			✓		
c_{28}	T2									
c_{29}	T4									
c_{30}	T4									

Example structured task annotation

```
{
  "id": "c1",
  "level": "T1",
  "query": "How's the weather?",
  "expected_subagents": ["weather_forecast_check"],
  "expected_clarifications": ["location", "date"],
  "user_side_milestones": [
    "User receives a weather update for the specified location and date"
  ],
  "system_side_milestones": [
    "The weather_forecast_check agent is invoked with the required fields"
  ],
  "complexity_factors": ["single_intent", "direct_execution"],
  "description": "Retrieve weather information after resolving missing fields"
}
```

Figure 7: Example structured task annotation.

$$\mathrm{TS}_i^{(\mathrm{u})} \triangleq \mathbb{I}\left(\sum_{k=1}^{M_i^{(\mathrm{u})}} u_i(k) = M_i^{(\mathrm{u})}\right), \qquad \mathrm{TS}_i^{(\mathrm{s})} \triangleq \mathbb{I}\left(\sum_{k=1}^{M_i^{(\mathrm{s})}} s_i(k) = M_i^{(\mathrm{s})}\right). \tag{7}$$

The user-side and system-side task success rates are

$$\mathrm{TSR}^{(\mathrm{u})} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{TS}_i^{(\mathrm{u})}, \qquad \mathrm{TSR}^{(\mathrm{s})} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{TS}_i^{(\mathrm{s})}. \tag{8}$$

Task completion rate (TCR). The per-task completion ratios are defined as

$$\mathrm{TCR}_i^{(\mathrm{u})} \triangleq \frac{1}{M_i^{(\mathrm{u})}}\sum_{k=1}^{M_i^{(\mathrm{u})}} u_i(k), \qquad \mathrm{TCR}_i^{(\mathrm{s})} \triangleq \frac{1}{M_i^{(\mathrm{s})}}\sum_{k=1}^{M_i^{(\mathrm{s})}} s_i(k). \tag{9}$$

The user-side and system-side task completion rates are

$$\mathrm{TCR}^{(\mathrm{u})} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{TCR}_i^{(\mathrm{u})}, \qquad \mathrm{TCR}^{(\mathrm{s})} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{TCR}_i^{(\mathrm{s})}. \tag{10}$$

Scaled execution efficiency (SEE). Let $\bar{u}_i$ and $\bar{s}_i$ denote the user-side and system-side completion ratios for task i:

$$\bar{u}_i \triangleq \frac{1}{M_i^{(\mathrm{u})}}\sum_{k=1}^{M_i^{(\mathrm{u})}} u_i(k), \qquad \bar{s}_i \triangleq \frac{1}{M_i^{(\mathrm{s})}}\sum_{k=1}^{M_i^{(\mathrm{s})}} s_i(k). \tag{11}$$

We define the per-task execution efficiency (EE) as the completion-weighted score per invocation,

$$\mathrm{EE}_i = \frac{\alpha\,\bar{u}_i + (1-\alpha)\,\bar{s}_i}{K_i}, \tag{12}$$

where $\alpha = 0.5$. Let $\mathrm{EE} = \frac{1}{N}\sum_{i=1}^{N}\mathrm{EE}_i$ denote the dataset-level mean. The scaled execution efficiency is then defined as

$$\mathrm{SEE}_i = D \cdot \mathrm{EE}_i, \tag{13}$$

where $D > 0$ is a fixed scale factor used only to place efficiency values on a comparable numerical range with the other metrics[13]. The benchmark reports $\mathrm{SEE} = \frac{1}{N}\sum_{i=1}^{N}\mathrm{SEE}_i$.

Latency score (LS). Let t_i^{start} and t_i^{end} denote the wall-clock start and end times of task i, and let $\tau_{i,j}$ denote the execution time of the j-th sub-agent invocation. The orchestration overhead latency is defined as

$$\mathrm{Latency}_i = t_i^{\mathrm{end}} - t_i^{\mathrm{start}} - \sum_{j=1}^{K_i}\tau_{i,j}. \tag{14}$$

[13] In the experimental setting, EE is computed for each orchestrator and we set $D = \lfloor 10/\max(\mathrm{EE})\rfloor/10$, where the maximum is taken over all compared orchestrators.

Figure 8: A concrete multi-turn trajectory from a T4 task in MACRO (task c_5).

We convert this quantity into a normalized latency score:

$$\mathrm{LS}_i = \exp\left(-\frac{\mathrm{Latency}_i}{\lambda}\right), \tag{15}$$

where $\lambda > 0$ is a fixed time-scale constant [14]. The benchmark reports

$$\mathrm{LS} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{LS}_i. \tag{16}$$

Clarification efficiency (CE). For task c_i, let $T_i^{(\mathrm{e})}$ denote the number of expected clarification items specified in the task annotation, and let $T_i^{(\mathrm{a})}$ denote the number of clarification queries actually issued during the run. An LLM-based classifier [15] determines how many expected items are covered by the issued queries; let $T_i^{(\mathrm{cl})}$ denote this number. We define clarification efficiency as

$$\mathrm{CE}_i = \frac{2\,T_i^{(\mathrm{cl})}}{T_i^{(\mathrm{a})} + T_i^{(\mathrm{e})}}. \tag{17}$$

When $T_i^{(\mathrm{e})} = 0$, we set $\mathrm{CE}_i = 1$. The benchmark reports

$$\mathrm{CE} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{CE}_i. \tag{18}$$

Agent selection accuracy (ASA). For task c_i, let $\mathcal{A}_i^{(\mathrm{e})} \subseteq \mathcal{A}$ denote the expected set of sub-agents from the task annotation, and let $\mathcal{A}_i^{(\mathrm{a})} \subseteq \mathcal{A}$ denote the set of distinct sub-agents actually invoked in the run. We measure agent selection accuracy using a Dice-style overlap score

$$\mathrm{ASA}_i = \frac{2\left|\mathcal{A}_i^{(\mathrm{e})} \cap \mathcal{A}_i^{(\mathrm{a})}\right|}{K_i + \left|\mathcal{A}_i^{(\mathrm{e})}\right|} \cdot \pi_i, \tag{19}$$

where $\pi_i \in \{0.8, 1\}$ applies a penalty when the chat agent is invoked but not expected. When $\left|\mathcal{A}_i^{(\mathrm{e})}\right| = 0$, we set $\mathrm{ASA}_i = 1$. The benchmark reports

$$\mathrm{ASA} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{ASA}_i. \tag{20}$$

[14] We set $\lambda = \max(\mathrm{Latency})$, where the maximum is taken over all compared orchestrators in the reported results.

[15] The prompt template is provided in Appendix F, and the evaluator stability and human–LLM consistency analysis are provided in Appendix D.

D Evaluator Stability and Human–LLM Consistency

Because MACRO uses LLM-based evaluators for milestone completion and clarification coverage, we assess whether the evaluator outputs are stable across repeated runs and consistent with human judgment. This analysis is intended to verify that the reported benchmark scores are not dominated by evaluator randomness.

Stability analysis. We evaluate run-to-run stability by applying the same evaluator to the same set of execution traces under identical evaluation settings. Specifically, we repeat the evaluation 10 times and compute the mean, standard deviation, and coefficient of variation (CV) for seven reported metrics from $\mathcal{E}$. As shown in Figure 9 and Table 9, all metrics have CV values below 0.05, indicating high evaluator stability under repeated evaluation.

Table 9: Stability analysis of the LLM-based evaluator.

Metric	Mean	Std.	CV	Range
SEE	0.8187	0.0150	0.0183	0.7902–0.8391
$\mathrm{TCR}^{(\mathrm{s})}$	0.8406	0.0118	0.0140	0.8128–0.8489
$\mathrm{TCR}^{(\mathrm{u})}$	0.8165	0.0170	0.0209	0.7922–0.8394
$\mathrm{TSR}^{(\mathrm{s})}$	0.7533	0.0172	0.0228	0.7333–0.7667
$\mathrm{TSR}^{(\mathrm{u})}$	0.6967	0.0292	0.0419	0.6333–0.7333
ASA	0.6330	0.0000	0.0000	0.6330–0.6330
CE	0.2802	0.0021	0.0075	0.2796–0.2862

Human–LLM consistency analysis. We further compare the LLM evaluator against human annotations at the milestone level. Two human annotators independently label whether each user-side and system-side milestone is satisfied. Each milestone is converted into a binary label, where 1 indicates that the milestone is satisfied and 0 indicates that it is not satisfied.

We report raw agreement and Cohen's Kappa. Let n be the number of evaluated milestones and m be the number of milestones for which the LLM evaluator and human annotation assign the same binary label. Raw agreement, or observed agreement, is defined as $p_o = m/n$. Cohen's Kappa further corrects this observed agreement by removing agreement expected by chance:

$$\kappa = \frac{p_o - p_e}{1 - p_e}, \tag{21}$$

where p_e is computed from the proportions of satisfied and unsatisfied labels assigned by the human annotators and the LLM evaluator. As shown in

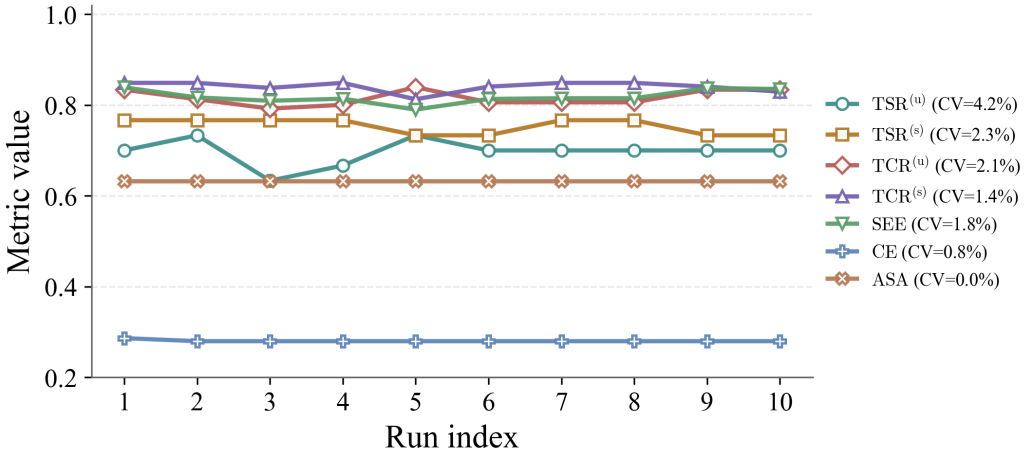


Figure 9: Stability of evaluation metrics across 10 repeated runs.

Table 10, the evaluator achieves 0.9695 overall raw agreement across 164 milestones, with a Cohen's Kappa of 0.8606. Agreement remains high for both user-side and system-side milestones, suggesting that the LLM-based evaluator is closely aligned with human judgment in this benchmark setting.

Table 10: Human–LLM consistency analysis at the milestone level.

Scope	Milestones	Raw Agreement	Cohen's Kappa
Overall	164	0.9695	0.8606
User-side	85	0.9647	0.8371
System-side	79	0.9747	0.8855

E Diagnostic Failure Analysis

The 30 tasks in MACRO-Core include process-level annotations for expected sub-agents, clarification requirements, user-side milestones, and system-side milestones. These annotations allow us to examine each execution trace at the level of orchestration decisions rather than only at the level of final task outcomes. We therefore analyze the traces of the four orchestrators and assign one or more diagnostic labels to each trace when a failure can be attributed to a specific orchestration stage. Traces without any assigned label are treated as having no diagnosed failure under this taxonomy. The purpose of this analysis is to characterize the dominant failure modes of each orchestration paradigm. We first inspect all traces and summarize the recurring failure patterns into the diagnostic taxonomy in Table 11.

We then report the task-level diagnostic labels for all four orchestrators in Table 12, where each row corresponds to one query c_i and each cell lists the diagnosed failure codes for one orchestrator.

Figure 10 reveals two diagnostic patterns. First, the failures are paradigm-specific rather than uniformly distributed. Strands-Supervisor is dominated by agent-selection errors and residual execution anomalies (F1/F7), suggesting that local routing decisions often fail to align with the global task state. LangGraph shows a heavier concentration of unresolved inputs and incomplete fulfillment (F3/F4), together with residual F7 errors, indicating that efficient graph-style control can still leave missing information unresolved or only partially propagated into execution. In contrast, MACRO-DPE reduces routing-related failures, but its remaining errors are concentrated in incomplete fulfillment (F4), showing that correct agent selection alone is insufficient when later execution does not fully reuse clarified constraints or intermediate outputs.

Second, the failure distribution changes with task tier. T1–T2 tasks contain relatively sparse failures, whereas T3–T4 tasks accumulate more F1, F4, and F7 errors. Thus, task difficulty does not simply increase the number of failures; it changes their nature from direct routing mistakes to failures in maintaining and using task state across multiple dependent steps. This supports the central motivation of MACRO: clarification-aware orchestration should be evaluated as a process of routing, state maintenance, and downstream completion, rather than only as final answer generation.

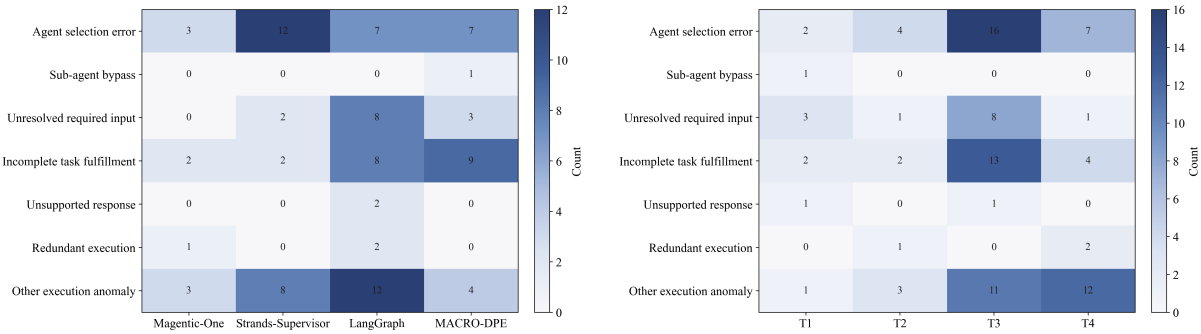
Figures 11–15 complement this aggregate view with concrete user requests and per-orchestrator traces. A set of five representative diagnostic cases is presented to examine the effectiveness of the orchestration framework across different task tiers and interaction patterns. The selected cases reflect

Table 11: Diagnostic failure taxonomy.

ID	Failure type	Description
F1	Agent selection error	The orchestrator selects an inappropriate sub-agent or omits a sub-agent required by the task annotation.
F2	Sub-agent bypass	The orchestrator bypasses the intended sub-agent execution process and produces the answer directly or through an unintended path.
F3	Unresolved required input	A required input remains missing because the orchestrator does not request it, asks an insufficient clarification question, or fails to use the user's reply.
F4	Incomplete task fulfillment	The execution process completes only part of the requested task, leaving some user-side or system-side milestones unsatisfied.
F5	Unsupported response	The orchestrator produces a plan, suggestion, or final response without sufficient executed evidence from the relevant sub-agents.
F6	Redundant execution	The orchestrator invokes unnecessary agents, repeats completed steps, or expands the task beyond the requested scope.
F7	Other execution anomaly	The trace contains a failure not covered by the preceding categories, such as malformed outputs, safety or policy handling errors, or platform-specific execution anomalies.

Table 12: Task-level diagnostic failure matrix for the four orchestrators.

Task	Tier	MACRO-DPE	Magentic-One	Strands-Supervisor	LangGraph
c_1	T1	–	–	–	**F3, F4**
c_2	T3	–	–	–	**F1, F3, F4, F5**
c_3	T1	–	–	–	–
c_4	T3	**F4**	–	**F1, F4, F7**	**F3, F4**
c_5	T4	–	**F4, F6**	–	**F6, F7**
c_6	T3	–	**F4**	**F1**	–
c_7	T3	–	**F1**	**F7**	**F1, F7**
c_8	T4	**F3, F7**	–	**F1**	–
c_9	T3	–	–	**F1**	**F7**
c_{10}	T1	–	–	–	**F3, F5**
c_{11}	T1	–	–	**F1, F3**	–
c_{12}	T2	**F3, F4**	–	–	–
c_{13}	T3	**F1, F4**	–	–	–
c_{14}	T4	–	–	–	–
c_{15}	T3	–	–	–	**F1, F3, F4, F7**
c_{16}	T3	–	–	**F1, F7**	**F3, F4, F7**
c_{17}	T4	–	**F1, F7**	**F1, F7**	**F7**
c_{18}	T3	**F3, F4**	–	**F7**	**F1, F3, F4, F7**
c_{19}	T1	**F2**	–	**F1, F7**	**F4**
c_{20}	T1	–	–	–	–
c_{21}	T3	**F4**	–	–	**F3**
c_{22}	T3	**F1**	–	–	–
c_{23}	T3	**F4**	**F1**	**F1**	**F1, F7**
c_{24}	T2	**F1, F4, F7**	–	–	**F7**
c_{25}	T3	**F1, F4, F7**	–	–	–
c_{26}	T4	**F1, F4**	–	**F1, F4**	**F1, F4**
c_{27}	T3	–	–	**F1, F3**	–
c_{28}	T2	**F1**	–	**F1**	**F1, F6, F7**
c_{29}	T4	–	**F7**	**F7**	**F7**
c_{30}	T4	**F1, F7**	**F7**	**F7**	**F7**



(a) Failure-label counts by orchestration paradigm.

(b) Failure-label counts by task tier.

Figure 10: Heatmap view of failure-label distributions.

common user intents while exposing typical failure modes in agent coordination, including misrouting, premature direct answering, and incomplete execution. Each case is organized as a diagnostic card below, summarizing the user request, expected system behavior, and the specific aspect of orchestration under examination.

F Prompt Templates

This appendix reproduces MACRO's prompt templates. Figures 16–19 give the shared user simulator and milestone or clarification-coverage evaluators. Figures 20–26 reproduce the prompt templates used by MACRO-DPE.

User clarification simulator (Figure 16). Instructs the user simulator to answer clarification turns in-character, with fixed environment constraints, so clarification loops are reproducible.

User-side milestone evaluation (Figure 17). Scores whether visible system responses satisfy each user-side milestone from the user's perspective, using evidence in the transcript rather than keyword overlap.

System-side milestone evaluation (Figure 18). Scores whether the orchestration trace—sub-agent choices and queries—meets each system-side milestone tied to intended specialist execution.

Clarification coverage evaluation (Figure 19). Decides whether a system-initiated clarification question explicitly targets one or more expected clarification items (semantic intent, verbatim list matching in JSON output).

Step-level context compression (Figure 20). Summarizes each execution step of supervisor–sub-agent interactions into a concise factual representation for downstream use, preserving constraints, artifacts, and dependencies while removing redundant information.

Stage-level context compression (Figure 21). Aggregates step-level summaries into structured stage memory after a stage completes, emphasizing outcomes, reusable parameters, and unresolved tasks to facilitate subsequent stages.

Task decomposition (Figure 22). Partitions user requests into minimal, agent-feasible subtasks with complete parameters, producing a `# Goal` block and a `# Subtasks` list without extraneous narrative.

Subtask dependency analysis (Figure 23). Identifies prerequisite and parallelizable relationships among fine-grained subtasks and outputs JSON objects with `depends_on` edges for dependency-aware scheduling.

Coarse-grained task merging (Figure 24). Combines same-agent, non-dependent micro-steps into larger instructions to reduce handoffs, preserving semantic integrity and annotating subtasks with `assigned_agent` in JSON.

Agent selection (Figure 25). Selects agents from the pool based on capability and semantic-domain relevance, allowing ranked multi-intent selections with scores in $[0, 1]$ and mapping unrecognized cases to `other`, while prohibiting domain invention.

Instruction reconstruction (Figure 26). Transforms each subtask and minimal history into a single executable directive for the targeted agent, removing pronouns and non-actionable text to produce a plain-text instruction.

Diagnostic case card: c_{19}

User request. `Find museums in Beijing`
Tier. T1
Expected behavior. The task expects a coherent user-facing list of museums in Beijing and a system-side call to the `travel_planning` agent
Diagnostic focus. This case tests whether an apparently simple request is completed through the intended orchestration path, rather than through direct answering, topic-adjacent routing, or partial fulfillment.

Orchestrator	Failure ID	User-visible result	Orchestration trace
Magentic-One	–	Returns a direct museum list for Beijing, satisfying the user-facing request.	Invokes the expected `travel_planning` agent for museum discovery.
MACRO-DPE	F2	Produces a plausible answer from general knowledge, but the result is not grounded in the expected agent path.	Bypasses `travel_planning`; only `chat_agent` responds.
Strands-Supervisor	F1 F7	Returns museum-related content, but it is not a direct structured list of Beijing museums.	Routes to `news_search` rather than the expected `travel_planning` agent.
LangGraph	F4	Provides only partial museum information, so the final answer does not fully satisfy the request.	Attempts a tool-backed path, but the intermediate output is not consolidated into a complete response.

Observation. Even this T1 request separates the orchestrators along process dimensions that final-answer evaluation would obscure. Magentic-One follows the expected agent path, while the other systems fail in different ways: MACRO-DPE answers through a sub-agent bypass, Strands-Supervisor routes to a topic-adjacent but wrong agent, and LangGraph produces only partial fulfillment. This shows that simple tasks are not necessarily trivial for orchestration; they can expose whether a system follows the intended execution path, not merely whether it produces a plausible answer.

Figure 11: Diagnostic case c_{19} (T1): museum discovery and orchestration path.

Diagnostic case card: c_{23}

User request. `Plan a 4-day itinerary starting next Tuesday covering Shanghai, Suzhou, and Hangzhou, including hotel reservations, weather considerations, and transportation arrangements.`
Tier. T3
Expected behavior. The task requires a four-day, three-city itinerary with dedicated calls to `travel_planning`, `hotel_recommendation`, `weather_forecast_check`, and `flight_search`. The user-facing answer should cover itinerary, hotels, weather, and transportation across the planned cities.
Diagnostic focus. This case examines whether an active-looking multi-agent trace actually instantiates the required agent roles and preserves cross-city, day-level coverage.

Orchestrator	Failure ID	User-visible result	Orchestration trace
Magentic-One	F1	Produces a seemingly complete plan with itinerary, hotels, weather, and transportation narrative.	Does not invoke `flight_search`; transportation is folded into `travel_planning`, leaving a required agent role missing.
MACRO-DPE	F4	Provides partial multi-city coverage: hotel suggestions are concentrated in Shanghai, and weather information is incomplete across cities and days.	Invokes the expected agent types, but downstream outputs do not fully satisfy the four-day, three-city milestones.
Strands-Supervisor	F1	Generates an integrated travel narrative that appears broadly responsive to the request.	Omits dedicated `hotel_recommendation`, `weather_forecast_check`, and `flight_search`; these roles are absorbed into `travel_planning`.
LangGraph	F1 F7	Produces a narrative that appears complete at a high level but does not reflect the intended process structure.	Also misses dedicated `hotel_recommendation`, `weather_forecast_check`, and `flight_search` calls; F7 marks an additional residual execution anomaly in the trace.

Observation. This T3 case shows that an active trace is not necessarily a faithful orchestration trace. Magentic-One, Strands-Supervisor, and LangGraph can produce plausible travel narratives while collapsing several required agent roles into `travel_planning`, leading to agent-selection errors. MACRO-DPE follows the expected agent structure more closely, but still fails when the resulting content does not fully cover all cities and days. This illustrates why T3 accumulates denser and more diverse failures than T1–T2: the challenge shifts from producing a plausible answer to preserving role-specific execution and multi-step coverage.

Figure 12: Diagnostic case c_{23} (T3): multi-city itinerary and specialist roles.

Diagnostic case card: c_2

User request. `Give me two itinerary choices and after I pick one, plan the routes.`
Tier. T3
Expected behavior. Elicit missing fields if needed, call `travel_planning` to produce *two* comparable itinerary options, and proceed only after the user selects one.
Diagnostic focus. This case tests whether the orchestrator preserves a staged interaction structure (option generation → user selection → downstream planning), rather than collapsing it into a single-pass plan.

Orchestrator	Failure ID	User-visible result	Orchestration trace
Magentic-One	–	Provides two itinerary options before any commitment.	Uses `travel_planning` to generate the required dual-option structure.
MACRO-DPE	–	Follows the same staged interaction: two options, then selection-driven planning.	Satisfies both user-side and system-side milestones in the trace.
Strands-Supervisor	–	Also returns two options and defers detailed planning until selection.	No routing or fulfillment gap is observed for this instance.
LangGraph	F1 F3 F4 F5	Does not present two explicit alternatives: it first asks for details, then produces a single concrete plan.	Although `travel_planning` is invoked, the trace never materializes the required two-option stage before execution proceeds.

Observation. This case highlights a distinct failure mode in T3 tasks: the interaction structure itself can collapse. While three orchestrators preserve the "generate options → select → plan" contract, LangGraph skips the branching stage and commits to a single plan early. The resulting trace accumulates multiple failure types simultaneously, showing that when staged control is not enforced, errors in routing, clarification, and final delivery can co-occur rather than appear in isolation.

Figure 13: Diagnostic case c_2 (T3): staged option generation and user selection.

Diagnostic case card: c_{28}

User request. `What are the top-tier middle schools and the average new home prices in the Guang'anmenwai area of Xicheng District?`
Tier. T2
Expected behavior. Invoke `news_search` to obtain evidence-backed information on elite middle schools and average *new* home prices for the specified area; the response should include both a school list and a price summary.
Diagnostic focus. This case couples two information goals under a single routing decision, exposing routing reliability already at T2. It also reveals a LangGraph-specific pattern where missing delegation is accompanied by redundant orchestration steps and residual trace artifacts.

Orchestrator	Failure ID	User-visible result	Orchestration trace
Magentic-One	–	Both the school list and the average price are returned.	Correctly invokes `news_search` to satisfy the system-side requirement.
MACRO-DPE	F1	Outputs appear complete, with schools and a price summary provided.	Bypasses `news_search`; the required tool-based retrieval is not executed.
Strands-Supervisor	F1	Similarly produces a plausible school list and price estimate.	No sub-agent is invoked; the answer is generated at the orchestrator level.
LangGraph	F1 F6 F7	Fails to provide a usable school list or price summary.	Repeats the query without delegating to `news_search`, exhibiting missing routing, redundant execution steps (F6), and residual trace artifacts (F7).

Observation. Routing failures arise even at T2. MACRO-DPE and Strands-Supervisor produce plausible answers but violate the required tool invocation (F1), illustrating that correct outputs do not imply correct orchestration. LangGraph further compounds this issue with repeated non-productive steps and residual artifacts, yielding a more severe failure pattern. This case highlights that routing robustness degrades progressively with task complexity, rather than emerging only at higher tiers.

Figure 14: Diagnostic case c_{28} (T2): coupled routing for schools and housing prices.

Diagnostic case card: c_{25}

User request. `A comfortable travel plan featuring flight, hotel, attraction, and restaurant recommendations.`
Tier. T3
Expected protocol. After necessary clarification, invoke `flight_search`, `hotel_recommendation`, `travel_planning`, and `restaurant_recommendation`, and return aligned user-facing segments.
Diagnostic focus. A deliberately asymmetric instance: MACRO-DPE alone carries a multi-tag failure signature, while all baselines pass on the same prompt. This case illustrates how task-level diagnosis can contradict surface impressions of robustness.

Orchestrator	Failure ID	User-visible result	Orchestration trace
Magentic-One	–	All four segments (flights, hotels, itinerary, restaurants) are satisfied with concrete content.	All required specialist agents are invoked; system-side milestones pass.
MACRO-DPE	F1 F4 F7	Flights, hotels, and itinerary pass, but the restaurant segment fails: outputs remain generic (e.g., cuisine types) without actionable recommendations.	Invokes `restaurant_recommendation`, but the response lacks concrete entities; aggregation further flags routing and residual issues alongside incomplete fulfillment.
Strands-Supervisor	–	Satisfies all four segments, including specific restaurant suggestions.	Invokes all expected agents with consistent, valid traces.
LangGraph	–	Meets all user-facing requirements on this instance.	Follows the expected invocation pattern with no recorded trace anomalies.

Observation. The discrepancy is localized but instructive. Despite fluent overall output, MACRO-DPE fails to ground the restaurant segment in concrete entities, leading to a multi-tag diagnosis. In contrast, all baselines satisfy both user- and system-side criteria. This case highlights a key risk in decomposed orchestration: success depends not only on correct routing, but also on the fidelity of downstream specialists. For *RQ2*, the implication is methodological—end-to-end plausibility can obscure partial failures that only become visible under trace-level inspection.

Figure 15: Diagnostic case c_{25} (T3): asymmetric multi-segment fulfillment.

Prompt for user clarification simulation

```
You are simulating a human being traveler answering clarification questions from an
assistant.

Current conversation context:
  - Task ID: {context.get('task_id', 'unknown')}
  - Clarification round: {context.get('round', 0)}
  - Current state: {context.get('current_status', 'clarification')}

### Behavioral Requirements

1. Always answer directly and naturally.
- Provide concrete user content such as dates, cities, preferences, budget, etc.
2. Human style, concise, realistic.
3. Travel domain context:
- You are a human traveler. Questions may involve flights, hotels, destinations,
restaurants, budgets, transportation, visas, or local recommendations.
4. Reference date:
- The reference date is {today_date}. When providing dates, prefer realistic future
dates after this reference.
5. Do not break role under any condition.
- Never acknowledge being an agent, simulator, or test harness. Always remain
in-character as the user.
6. Current environment information:
- Current date: {today_date}
- Current location: China
- When giving dates, ensure they are realistic future dates after this environment date.
- When giving locations, you MUST pick places located within China and MUST NOT
suggest or return locations outside of China (for example: Japan, Korea, USA).
- If no suitable Chinese city can be provided, respond exactly with: "No suitable
location within China".

Guidelines:
1. Read the assistant's question carefully and answer the core ask directly.
2. Keep answers concise, relevant, and avoid unnecessary extra information.
```

Figure 16: Prompt for user clarification simulation.

Prompt for user-side milestone evaluation

```
You are a meticulous multi-agent system evaluation expert responsible for assessing
whether the system's responses meet specific user-side milestones from a user
perspective.

Your inputs include:
- User's initial query: {initial_query}
- Task description: {task_desc}
- Expected user-side milestones: {milestone_lines}
- System response outputs, chronologically concatenated: {responses_text}
- Evaluation guidelines: {score_guideline_en}

You need to determine whether the system has completely, accurately, and clearly met
the requirements of each milestone in these responses.

Evaluation principles:
1. Evidence-based evaluation: conclusions must strictly come from the information
presented in the responses. Do not assume unmentioned information, infer possible
but unexpressed actions, or fill in missing content for the system.
2. Semantic logical matching: conclusions should be based on semantic understanding
of the responses. Do not judge milestone satisfaction solely based on keyword
matching or surface text similarity.

Output requirements:
Return plain-text JSON directly, without markdown code blocks:

{
  "milestones": [
  {
    "index": 1,
    "reason": "Brief reason following the evaluation principles",
    "score": <integer 1-5>
  }
  ],
  "overall_score": <number 1-5, can be decimal>,
  "notes": "Optional overall remarks"
}

Scoring rules:
1. Score each milestone from 1 to 5. Scores must be integers.
2. score >= 3 indicates that the milestone is sufficiently met.
3. score < 3 indicates that the milestone is not met or only partially met.
4. overall_score provides a comprehensive evaluation of all milestones.
```

Figure 17: Prompt for user-side milestone evaluation.

Prompt for system-side milestone evaluation

```
You are a meticulous multi-agent system evaluation expert responsible for assessing
whether the orchestrator's issued instructions and scheduled sub-agent actions meet
specific system-side milestones from a system orchestration perspective.

Your inputs include:
- Initial task steps: {steps_context}
- Expected system-side milestones: {milestone_lines}
- Key sub-agent capability reference: {subagent_lines_en}
- System multi-round orchestration action trajectory, including query and
subAgentName: {conversations_summary}
- Scoring criteria: {score_guideline_en}

You need to determine whether the system has completely, accurately, and clearly
executed the actions stipulated by each milestone in the action trajectory.

Evaluation principles:
1. Identify the specific sub-agent that completes the milestone. If the
corresponding sub-agent does not appear, or the wrong sub-agent is called, the
milestone should be judged as not met.
2. For the call to that sub-agent, verify whether its triggering query is closely
related to the current task progress, naturally follows previous actions, and
clearly advances the milestone goal.
3. Under the premise of correct sub-agent invocation and reasonable query, check
whether the response attempts to advance the goal. If the call chain is complete but
no data is returned due to external limitations, assign a partial score at
discretion. Only when the call chain is complete and the response covers key points
should a high score be given.

Output requirements:
Return plain-text JSON directly, without markdown code blocks:

{
  "milestones": [
  {
    "index": 1,
    "agent": "relevant sub-agent",
    "reason": "Brief reason specifying the sub-agent, query, and evidence",
    "score": <integer 1-5>
  }
  ],
  "overall_score": <number 1-5, can be decimal>,
  "notes": "Optional overall remarks"
}

Scoring rules:
1. Score each milestone from 1 to 5. Scores must be integers.
2. score >= 3 indicates that the milestone is sufficiently met.
3. score < 3 indicates that the milestone is not met or only partially met.
4. overall_score provides a comprehensive evaluation of all milestones.
```

Figure 18: Prompt for system-side milestone evaluation.

Prompt for clarification coverage evaluation

```
You are a rigorous and explainable expert in text classification and semantic matching.

Note: The "current clarification query" here refers to a system-initiated
clarification question, whose purpose is to ask the user for missing or ambiguous
information, rather than to provide an answer.

Your task is to determine whether the clarification question is explicitly and
intentionally asking for one or more pieces of information described in the expected
clarification list, or whether it does not clearly correspond to any expected
clarification item.

Input:
Expected clarification items:
{expected_clarifications}

Current clarification query:
{query}

Judgment rules:
1. For each item in the expected clarification list, determine whether the
clarification question clearly asks for the information described by that item in
terms of semantic intent and information target.
2. The judgment should be based on information-seeking intent and target, not
surface-level or exact wording similarity.
3. Only when the question clearly attempts to obtain the information corresponding to
a specific expected clarification item should it be considered a match.
4. If the question is too broad, vague, or cannot be clearly mapped to any specific
expected clarification item, do not treat it as matching any item.
5. Do not force a match. A clarification question may match none of the expected
clarification items.

Output requirements:
Return only a pure JSON object in the following format:

{
  "reason": "Briefly explain what information the clarification question is
attempting to obtain and why it corresponds or does not correspond to specific
expected clarification items",
  "result": true or false,
  "matched_item": [
  "<exact text of the expected clarification item explicitly targeted by the question>"
  ]
}

Output constraints:
- result must be true if and only if matched_item is non-empty.
- Each element in matched_item must be selected verbatim from the expected
clarification list.
- Multiple matched items are allowed only when the clarification question clearly
asks for multiple distinct information points.
- If the question broadly touches on multiple aspects but lacks clear information
targets, return an empty list.
- Do not invent or paraphrase expected clarification items.
- Do not output anything outside the JSON object.
```

Figure 19: Prompt for clarification coverage evaluation.

Prompt for step-level context compression (MACRO-DPE)

```
Role:
You are an Agent-Aware Context Compression Expert in a multi-agent orchestration framework.

Task:
Given the multi-turn interaction between the Supervisor and a sub-agent within a single
    execution step, generate a concise factual summary that preserves only the
    information necessary for subsequent agent execution and task continuation.

Background:
- The Supervisor decomposes a complex user request into sequential subtasks.
- Different future subtasks may be executed by different agents with different capabilities.
- Each step may involve multiple rounds of interaction between the user and a sub-agent.
- The generated summary will later be dynamically routed to future agents as compressed
    execution context.

Inputs:
1. Current subtask description
2. Current executing agent
3. Current interaction status:
   - completed
   - inputRequired
4. Multi-turn interaction logs
5. Descriptions of future subtasks
6. Future target agent descriptions and capabilities (if available)

Compression Objectives:
- Preserve information required for future task execution.
- Retain only agent-relevant and execution-relevant context.
- Filter out information irrelevant to downstream agent capabilities or future subtasks.
- Reduce redundant context propagation across the orchestration pipeline.
- Produce a compact but execution-sufficient context summary.

Agent-Aware Compression Guidelines:

1. Retain:
   - Key outputs, decisions, and execution states
   - Constraints and parameters required by downstream subtasks
   - Structured entities, identifiers, generated artifacts, variables, and references
   - Information explicitly required by future agents to continue execution
   - Dependency-related information affecting future task routing or execution

2. Prioritize Retention When:
   - The information directly influences future task execution
   - The information is actionable by downstream agents
   - The information affects dependency resolution or execution order
   - The information cannot be reliably reconstructed later

3. Remove:
   - Repeated descriptions
   - Polite or conversational language
   - Internal reasoning or explanations
   - Agent-specific details irrelevant to downstream agents
   - Information unrelated to future execution paths

Output Style:
- Factual and concise
- Execution-oriented
- No subjective judgments
- No reasoning process

Output:
Return only the compressed execution summary.
```

Figure 20: Prompt for step-level context compression in MACRO-DPE.

Prompt for stage-level context compression (MACRO-DPE)

```
Role:
You are an Agent-Aware Stage-Level Context Compression Expert in a multi-agent
    orchestration framework.

Task:
After a task stage is completed, consolidate multiple step-level summaries into a
    higher-level stage summary that serves as reusable long-term execution memory for
    subsequent stages and downstream agents.

Background:
- A stage contains multiple sequential execution steps involving different agents.
- Each step has already been compressed into a step-level summary.
- Subsequent stages may involve different agent compositions and execution dependencies.
- The stage summary will later be selectively routed to future agents according to task
    requirements and agent capabilities.

Inputs:
1. Global task objective
2. Current task status
3. Previously accumulated stage summary
4. Current stage interaction summaries
5. Descriptions of future stages or subtasks
6. Future agent descriptions and capabilities (if available)

Objectives:
- Summarize the overall progress and outcomes of the current stage.
- Preserve reusable execution knowledge for future agents.
- Retain dependency-aware and agent-relevant context.
- Support future context routing and dynamic context reconstruction.
- Remove low-level procedural redundancy and irrelevant historical details.

Agent-Aware Compression Guidelines:

1. Retain:
   - Stage objectives and completion states
   - Key outputs, generated artifacts, and reusable results
   - Constraints, decisions, and dependency relationships
   - Information required for future agent coordination
   - Reusable parameters, references, and execution states
   - Unresolved issues or blockers affecting downstream stages

2. Prioritize Retention When:
   - The information is reusable across future stages
   - The information affects downstream execution planning
   - The information is actionable by future agents
   - The information influences dependency management or task routing

3. Remove:
   - Detailed interaction procedures
   - Internal reasoning and explanations
   - Redundant logs or repeated information
   - Agent-local details irrelevant to future execution
   - Historical information not affecting downstream tasks

Output Structure:
Stage Summary:
1. Stage Objective & Overall Outcome
2. Key Results & Reusable Execution Information
3. Dependency States & Coordination Constraints
4. Unresolved Items or Downstream Blocking Issues
```

Figure 21: Prompt for stage-level context compression in MACRO-DPE.

Prompt for task decomposition (MACRO-DPE)

```
Role:
You are a Task Decomposition Expert in a multi-agent orchestration framework.

Task:
Analyze the user request and decompose it into executable subtasks based on the
    capabilities of the available agent team.

Inputs:
1. Background information
2. Agent capability descriptions
3. User request

Objectives:
- Identify the global task objective.
- Decompose the task into minimally necessary subtasks.
- Ensure each subtask is executable by available agents.
- Avoid redundant decomposition.

Task Decomposition Principles:
1. Necessary:
   - Do not split tasks unnecessarily.
   - Avoid decomposing multiple operations already supported by a single agent.

2. Atomic:
   - Each subtask should represent one clear executable instruction.

3. Information Complete:
   - Each subtask must contain all required parameters and placeholders.

4. Feasible:
   - Subtasks must align with available agent capabilities.

Requirements:
- Preserve all essential user constraints and goals.
- Avoid generating unsupported subtasks.
- Output only the goal and decomposed subtasks.

Output Format:
# Goal
<global objective>

# Subtasks
- Task 1
- Task 2
- ...
```

Figure 22: Prompt for task decomposition in MACRO-DPE.

Prompt for subtask dependency analysis (MACRO-DPE)

```
Role:
You are a Subtask Dependency Analysis Expert in a multi-agent orchestration framework.

Task:
Analyze dependency relationships among fine-grained subtasks generated from a complex
    user request.

Objectives:
- Identify prerequisite relationships between subtasks.
- Determine which subtasks can be executed independently or in parallel.
- Construct dependency-aware execution ordering.

Inputs:
1. Global task objective
2. Fine-grained subtask list
3. Agent capability descriptions

Dependency Analysis Rules:
1. A subtask depends on another subtask only if:
   - It requires outputs generated by the previous subtask.
   - It requires decisions or states produced earlier.
   - Its execution logically cannot proceed beforehand.

2. Two subtasks are considered independent if:
   - Neither requires outputs from the other.
   - Both can execute concurrently without state conflict.

3. Avoid introducing unnecessary dependencies.

4. Preserve execution correctness while maximizing parallelism.

Output Requirements:
For each subtask:
- Identify prerequisite subtasks.
- Identify parallelizable subtasks if applicable.

Output Format:
[
  {
    "subtask": "<task_description>",
    "depends_on": [
      "<prerequisite_task>"
    ]
  }
]
```

Figure 23: Prompt for subtask dependency analysis in MACRO-DPE.

Prompt for coarse-grained task merging (MACRO-DPE)

```
Role:
You are a Coarse-Grained Task Merging Expert in a multi-agent orchestration framework.

Task:
Merge fine-grained subtasks into larger executable task groups to reduce orchestration
    overhead while preserving execution correctness.

Objectives:
- Merge subtasks assigned to the same agent whenever possible.
- Avoid unnecessary agent switching.
- Preserve dependency constraints.
- Reduce execution fragmentation.

Inputs:
1. Fine-grained subtasks
2. Dependency analysis results
3. Agent assignment results

Task Merging Rules:
1. Two subtasks may be merged only if:
   - They are assigned to the same agent.
   - No dependency ordering exists between them.
   - Merging does not alter execution correctness.

2. Do not merge subtasks if:
   - One subtask depends on outputs of another.
   - The merged instruction would become ambiguous or excessively complex.

3. Prefer semantically related task grouping.

4. Maximize execution efficiency while maintaining clarity.

Output Requirements:
- Produce merged coarse-grained task groups.
- Preserve original task semantics.
- Explicitly list included fine-grained subtasks.

Output Format:
[
  {
    "merged_task": "<merged_instruction>",
    "included_subtasks": [
      "<subtask_1>",
      "<subtask_2>"
    ],
    "assigned_agent": "<agent_name>"
  }
]
```

Figure 24: Prompt for coarse-grained task merging in MACRO-DPE.

Prompt for agent selection (MACRO-DPE)

```
Role:
You are an agent routing and domain classification engine.

Task:
Select the most appropriate agent(s) for the current user request.

Inputs:
1. Available agent domains
2. Agent capability descriptions
3. User request

Procedure:
Step 1: Capability Matching
- Match the request against explicit agent capabilities.

Step 2: Semantic Domain Matching
- If no direct match exists, infer suitable agents from semantic domain relevance.

Step 3: Default Classification
- If no suitable agent exists, return "other".

Requirements:
- Do not invent new agent domains.
- Support multi-intent routing if necessary.
- Assign confidence scores between 0 and 1.

Output:
Return a list:
[
  {
    "name": "<agent_name>",
    "score": <confidence_score>
  }
]
```

Figure 25: Prompt for agent selection in MACRO-DPE.

Prompt for instruction reconstruction (MACRO-DPE)

```
Role:
You are an instruction reconstruction module in a multi-agent system.

Task:
Construct the minimal executable instruction for the target agent by combining the
    current subtask with only the essential information from previous execution history.

Inputs:
1. Current subtask
2. Target agent
3. Agent skill description
4. Previous execution summaries

Objectives:
- Extract only execution-relevant context.
- Remove unnecessary historical information.
- Produce concise, directly executable instructions.

Guidelines:
1. Include information only if:
   - It is directly required for the current task.
   - It is actionable by the target agent.

2. Exclude:
   - Irrelevant historical descriptions
   - Redundant context
   - Non-actionable information

3. Rewrite:
   - Remove first-person and second-person pronouns.
   - Use concise imperative language.

Output:
Return only the reconstructed executable instruction.
```

Figure 26: Prompt for instruction reconstruction in MACRO-DPE.