AGENTQOSBENCH: EVALUATING QUALITY-OF-SERVICE AWARENESS IN AGENTS UNDER COMPLEX TOOL DEPENDENCIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Agents increasingly handle tasks with complex tool dependencies. Beyond correct completion, users have service-quality requirements or preferences. Quality of service (QoS), rooted in networking and service computing, describes latency, cost, and reliability. We evaluate agents' QoS awareness by extending QoS-aware service composition to autonomous planning with tool-call graphs, jointly assessing tool selection, dependencies, and whole-graph optimization. We introduce QoSGen (Quality-of-Service-Aware Benchmark Generation), a construction and validation pipeline that produces AgentQoSBench: 600 synthetic tasks across consumer and workplace scenarios with multiple valid plans and exact optima under explicit service models grounded in tool characteristics and composition mechanisms. For 49.7% of functionally valid tool-call plans produced by mainstream models, we verify a complete alternative with strictly better QoS. Using failure diagnosis and reasoning traces, we analyze agents' graph-search behavior, showing how local choices can overlook sharing and parallelism or create cross-branch conflicts. These findings motivate training-free search prompting and prompt-guided graph-search assistance. Prompting improves optimization within valid plans for GLM and DeepSeek, while tool assistance highlights the importance of preserving task requirements during candidate integration. Together, these approaches offer practical directions for improving QoS awareness through whole-graph comparison and agent-tool collaboration.

1 INTRODUCTION

Large language models (LLMs) increasingly tackle multi-step tasks with external tools (Shen et al., 2024; Kim et al., 2023; Zhang et al., 2026). Tasks naturally require retrieving, transforming, and combining intermediate results, so agents must choose suitable tools and correctly connect their inputs and outputs.

Correct completion is essential, but users may also prioritize faster delivery, lower cost, or greater reliability. These properties constitute *quality of service* (QoS), with trade-offs studied in QoS-aware service composition (Zeng et al., 2004). QoS research offers a foundation for agent planning and service selection, but nondeterministic outputs and unstructured information pose new challenges (Zeng et al., 2025). Building on this perspective, we adapt QoS-aware composition to agent planning. This formulation provides a common basis for adapting two decades of QoS service-selection and composition-optimization algorithms to agent tasks. Symbolic references preserve dependencies before output values are known, while explicit retrieval, parsing, and conversion calls bring information processing into QoS evaluation.

Figure 1 compares three plans for publishing the same training video and sharing it with employees. Local processing reuses compute to save cost; cloud acceleration exploits parallelism to reduce latency; in-platform processing reduces cross-system handoffs to favor reliability. The preferred plan depends on the user's service priorities.

Complex dependencies shape service quality, while QoS preferences guide tool selection and connections. We evaluate QoS awareness in this setting: can agents search through distracting alterna-

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

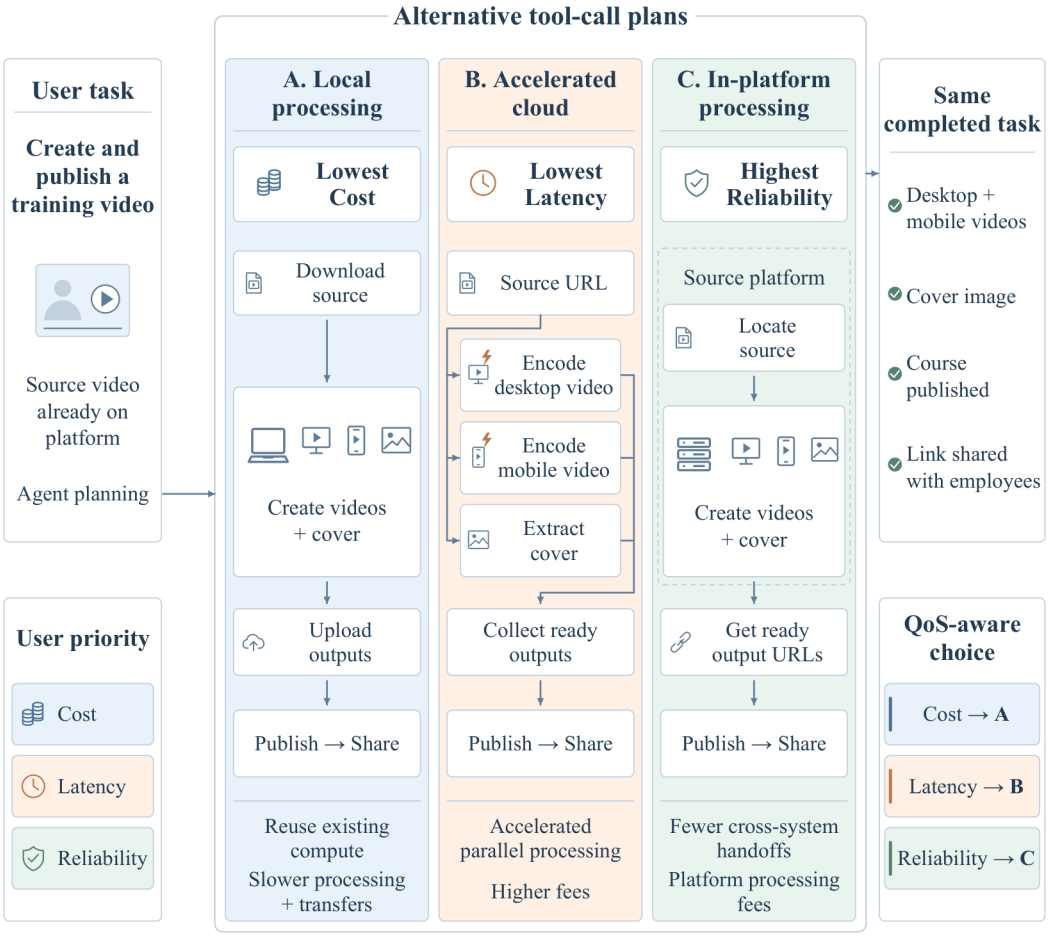

Figure 1: Three tool-call plans for one training-video task, each best on a different QoS dimension under the illustrated service assumptions.

tives to construct correct plans that optimize the stated service objectives? We use tool-call graphs to make parallelism, sharing, and multi-input dependencies explicit, beyond a linear sequence of calls.

Existing benchmarks address tool selection and dependencies (Shen et al., 2024) or quality–price and cost-based optimization (Wu et al., 2025; Liu et al., 2026a;b). Our focus is their conjunction: input provenance, object relationships, and joint task obligations evaluated alongside exact optimization of latency, cost, and reliability on the same autonomously constructed graph.

We introduce QoSGen (Quality-of-Service-Aware Benchmark Generation; Figure 2) to co-design tasks and tool worlds, verify complete alternatives and exact optima, and test instances and the evaluator. It produces AgentQoSBench: 600 synthetic tasks across consumer and workplace scenarios, balanced across one to three ordered objectives. Agents receive full tool environments and public contracts, not candidate plans. Evaluation measures functional correctness, exact optimality, and partial attainment through optimal prefixes and remaining gaps.

Across five models, 960 of 1,933 valid plans are non-optimal, although 475 already attain the first-priority optimum. Search prompting improves service selection for GLM and DeepSeek; tool assistance reveals the importance of preserving task requirements when expressing searches and integrating candidates.

Our contributions are the adaptation of QoS-aware composition to autonomous tool-graph planning (Section 3); QoSGen and the resulting AgentQoSBench benchmark (Section 4); and findings on optimization headroom and training-free improvement (Sections 5–6).

2 Related Work

QoS-aware service composition. Classical QoS research optimizes service selection and composition structure under user preferences (Zeng et al., 2004; 2008). Its retrospective connects these decisions to agent planning (Zeng et al., 2025). DeepPlanning studies constrained travel and shopping (Zhang et al., 2026); our focus is the service quality of tool-call plans.

Tool dependencies and correctness. TaskBench tests decomposition and arguments (Shen et al., 2024), while WorfBench measures graph agreement and analyzes critical-path latency (Qiao et al., 2025). FlowBench supplies procedural knowledge (Xiao et al., 2024); NESTFUL tests output-to-input dependencies (Basu et al., 2025). AppWorld validates cross-application states, and ToolSandbox evaluates trajectories against partially ordered milestones (Trivedi et al., 2024; Lu et al., 2025).

Service-aware planning evaluation. Table 1 summarizes branching or multi-input dependencies and service evaluation. OpenCATP measures plan-level execution time and price alongside output quality (Wu et al., 2025). CostBench provides static cost-optimal references and CostGap, with call-count tie-breaking, and also tests dynamic replanning (Liu et al., 2026a). MultiSoTLBench/TOPT evaluates cost–length paths within its expanded subgraph, where path cost can include tool latency (Liu et al., 2026b). TPS-Bench measures end-to-end latency and LLM inference cost (Xu et al., 2026); PeakBench separates dependency inference from capacity-constrained scheduling with verified dependencies (Chen et al., 2026). AgentQoSBench evaluates complete call graphs against input, object, and output requirements and an exact optimum under ordered latency, cost, and reliability objectives.

Table 1: Comparison of tool-planning benchmarks.

Benchmark	Plan	Complex tool dependencies	QoS			Exact opt.
			L	C	R	
TaskBench	Graph	✓	×	×	×	×
OpenCATP	DAG	✓	✓	✓	×	×
CostBench	Trajectory	◯	×	✓	×	✓
MultiSoTLBench	Chain	◯	◯	✓	×	✓
TPS-Bench	Online calls	✓	✓	✓	×	×
PeakBench	DAG / schedule	✓	✓	×	×	×
AgentQoSBench (ours)	**DAG**	✓	✓	✓	✓	✓

✓ evaluated ◯ chain/path scope × not targeted.
L: latency; C: tool/path or inference cost; R: service-success probability.
Exact opt.: exact reference within the evaluation domain.

3 QoS-Aware Tool-Call Graph Planning

3.1 QoS and Agent Planning

AgentQoSBench adapts QoS-aware service composition (Zeng et al., 2004; 2025) to agents that choose tools and construct their connections. Symbolic references identify upstream results before their values are known; retrieval, parsing, and representation conversion are explicit calls with dependencies and service costs. Public contracts support static evaluation of the resulting plans.

This makes the origin and required properties of intermediate results explicit without fixing their eventual contents. An agent plans both the calls that deliver an output and the information-processing calls that make it usable downstream. Their costs and dependencies therefore participate in the same QoS comparison.

3.2 Call Graphs and Functional Correctness

Graph-planning methods select paths or subgraphs (Wu et al., 2024), with parameter links and symbolic references supporting cross-call dependencies (Lee et al., 2026; Kim et al., 2023; Wei et al., 2026). An instance x provides a task, initial facts, a full tool catalog, public contracts, and QoS objectives. These rules implicitly define a search space of possible calls and connections. The agent constructs a task-completing subgraph rather than selecting from supplied plans. Its answer is a DAG P with call instances V, input bindings, data and control dependencies, and final outputs. Each call specifies its tool and implementation.

The valid-plan set $\mathcal{F}_x$ contains all graphs satisfying the public structural, call, dependency, and task requirements. This includes appropriate input sources, object and version relationships, and complete final outputs. Alternative constructions and harmless redundancy are accepted; every submitted call is checked and charged (Appendix A).

3.3 Whole-Graph QoS and Optimization

For call v, let ℓ_v, c_v, and r_v denote modeled duration, cost, and success probability. Standard aggregation (Yang et al., 2012) gives critical-path latency, total cost, and all-call reliability:

$$\begin{aligned} L(P) &= \max_{\text{path } \pi \text{ in } P} \sum_{v \in \pi} \ell_v, \\ C(P) &= \sum_{v \in V} c_v, \\ R(P) &= \prod_{v \in V} r_v. \end{aligned} \tag{1}$$

Paths include data and control dependencies and single-node paths. We assume sufficient parallel capacity and independent call successes. All calls count; shared calls count once and repeated calls separately.

Tasks minimize latency or cost, maximize reliability, or optimize two or three objectives in a stated priority order. A later objective only breaks ties on all earlier ones: latency–cost–reliability selects the fastest valid plans, then the cheapest among them, then the most reliable. All tied optima are accepted.

The ordered optimum provides a verifiable target aligned with the user's priorities. Optimality rates measure attainment; prefixes, gaps, and ranks measure progress toward it (Section 5.3).

4 QoSGen and AgentQoSBench

QoSGen couples task construction with checks of public semantics, complete alternatives, and QoS optima. Figure 2 shows generation and validation (stages 1–4), followed by agent evaluation and refinement (stages 5–6).

4.1 Source Curation and Task–World Co-design

Related synthesis methods construct tools from queries (Ye et al., 2025), guide dialogue with graphs (Wang et al., 2025), or plan parameter sources (Khandelwal et al., 2026). We co-design tasks and tool worlds around retrieval, conversion, aggregation, and alternative complete plans. Output requirements determine necessary connections; tool guarantees determine valid alternatives. Compositions include independent bundles and related requests with shared results and cross-task dependencies.

Each instance exposes its full catalog, including background tools, nearby alternatives, integrated services, and sharing opportunities. QoS assignments follow declared service mechanisms: native versus local processing, request and write overhead, batching, and read workload. These modeled values are fixed before plan comparison. Sharing avoids duplicate charges, parallelism reduces waiting, and every call contributes to all-call reliability. Preferred plans emerge from tool characteristics, dependency structure, and user priorities.

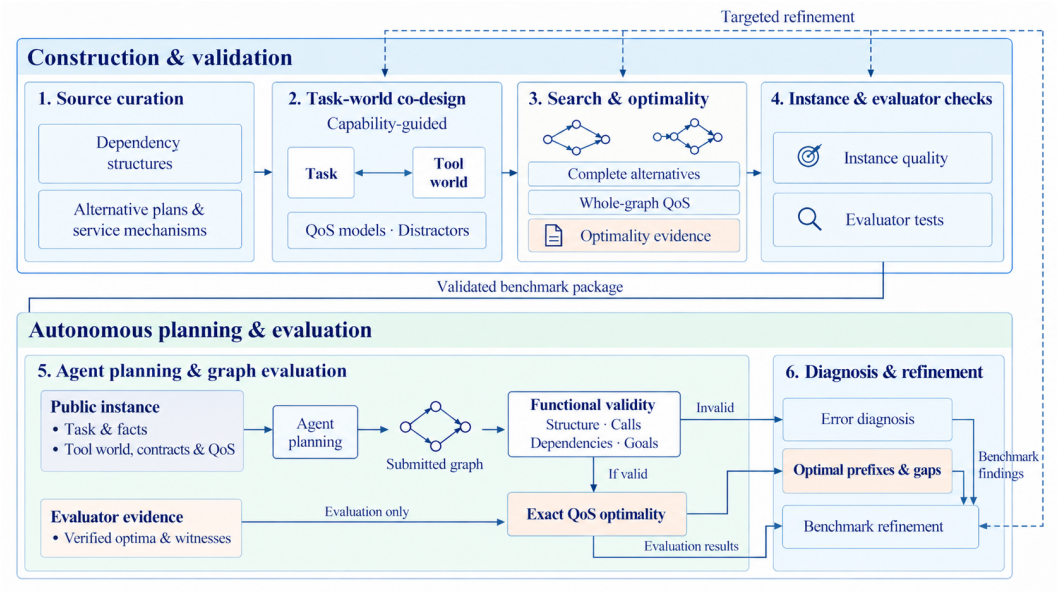

Figure 2: QoSGen and benchmark evaluation. Stages 1–4 construct and validate AgentQoSBench; stages 5–6 evaluate agent plans and guide refinement.

4.2 Complete Alternatives and QoS Optimality

Each task has at least two verified complete constructions differing in tools, decomposition, sharing, or bindings. Every specified objective distinguishes complete plans; later objectives require alternatives tied at all preceding optima.

Public rules generate a finite producer-and-binding space, retaining required identities, multiplicities, and state effects. The coverage argument maps legal plans into this space without worsening QoS. Exact search or SMT gives an optimum bound; a decoded graph must pass functional replay and attain the same service vector. Together, coverage and attainment establish the reference optimum (Appendix C).

4.3 Instance and Evaluator Checks

The evaluator checks public obligations rather than matching an author graph. Instance checks establish joint satisfiability, complete alternatives, and effective objectives. Positive controls test accepted constructions; negative graphs target source, object, dependency, and output violations. Validation includes 22,733 negative semantic controls and 685 passing test methods. Four scoring fixtures per task cover optimal, non-optimal, missing-goal, and invalid-implementation answers. All 600 public inputs reconstruct their definitions, and all reference optima are recomputed. After programmatic generation and scoring, LLMs review every task and answer against public descriptions and contracts, checking semantic agreement alongside executable behavior. Appendices B and B.1 detail the controls and runnable examples.

4.4 From Benchmark Assembly to Evaluation

QoSGen produces 600 tasks balanced across consumer and workplace scenarios and one, two, and three QoS objectives. Agents receive public instances; only the evaluator receives optima and witness graphs. Functional failures enter diagnosis directly, while valid graphs undergo QoS evaluation (Section 5). Refinement returns task or tool issues to co-design, optimum-reference issues to verification, and evaluator errors to testing, as shown in Figure 2.

5 Evaluation and Findings

We evaluate five LLMs on all 600 tasks, yielding 3,000 original-prompt answers under Thinking High. Models receive full tool environments and QoS priorities, but no reference plans or optima. Appendix A gives the protocol.

5.1 Evaluation Metrics

Functional success (F) measures valid plans; **conditional optimality** ($O \mid F$) measures optimality among valid plans; **overall optimality** (O), the primary metric, measures answers that are both valid and optimal. For N answers, n_F valid plans, and n_O valid, optimal plans,

$$F = \frac{n_F}{N}, \qquad O \mid F = \frac{n_O}{n_F}, \qquad O = \frac{n_O}{N}. \tag{2}$$

Rates are percentages. Average weights defined model rates equally; bold marks column maxima and arrows indicate preferred directions.

5.2 Task Completion Does Not Guarantee QoS Optimality

Across models, 960 of 1,933 functionally valid plans are non-optimal: each has a verified complete, valid alternative with strictly better QoS under the specified priorities. These comparisons identify concrete improvements within plans that already complete the task. DeepSeek reaches 95.33% functional success but 59.83% overall optimality (Table 2). Qwen's 9.33% functional success instead reflects difficulty constructing valid plans.

Table 2: Original-prompt results (%) by requested thinking setting. Dashes denote unavailable or undefined rates.

	Thinking High			Thinking Off		
Model	$F \uparrow$	$O \mid F \uparrow$	$\boldsymbol{O} \uparrow$	$F \uparrow$	$O \mid F \uparrow$	$\boldsymbol{O} \uparrow$
DeepSeek-V4-Pro	**95.33**	**62.76**	**59.83**	9.83	**8.47**	0.83
GLM-5.2	87.83	43.07	37.83	5.50	0.00	0.00
Gemini-3.1-Pro	67.83	54.05	36.67	—	—	—
GPT-5.6 Terra	61.83	42.86	26.50	**52.28**	5.48	**2.87**
Qwen3.5-Flash	9.33	14.29	1.33	0.00	—	0.00
Average	64.43	43.41	32.43	16.90	4.65	0.93

Single-objective tasks already reveal different optimization profiles: pooled $O \mid F$ reaches 97.55% for latency, 84.42% for reliability, and 51.69% for cost, despite similar F (60.30–61.79%). Cost optimization thus presents a substantial challenge even without competing service objectives (Appendix D).

Across objective-count groups, functional success stays within 61–67%, while conditional optimality is 77.70% for one objective and 24.81% for three. DeepSeek shows the same pattern despite 95–96% functional success across these groups.

Failure profiles locate these barriers (Table 3). QoS dominates DeepSeek and GLM failures; structure and call errors dominate Qwen's, while dependency and goal errors contribute 43.68% for Gemini. Improvement priorities therefore range from legal call construction to complete-plan comparison. Appendix D gives counts, and Appendix B.1 illustrates verdicts with runnable controls.

Under Thinking Off, the same 2,393 pairs across four models yield $F = 16.80\%$ and $O = 0.92\%$, versus 63.60% and 31.42% under high. Terra retains 52.28% functional success but reaches 2.87% overall optimality; the others each have $F < 10\%$. Both valid-subgraph discovery and optimization are affected (Appendices A and E).

Table 3: First-failure shares under Thinking High (%). Checks are cumulative: an earlier failure prevents overall success, and later checks are blocked. Each failed answer is counted once; QoS means valid but non-optimal. Average weights models equally.

Model	n	Structure	Call	Dependency	Goal	QoS
DeepSeek-V4-Pro	241	0.00	3.73	0.83	7.05	**88.38**
GLM-5.2	373	1.07	10.99	4.29	3.22	80.43
Gemini-3.1-Pro	380	4.74	2.37	**15.53**	**28.16**	49.21
GPT-5.6 Terra	441	1.81	31.07	6.12	12.93	48.07
Qwen3.5-Flash	592	**41.89**	**40.88**	6.59	2.53	8.11
Average	—	9.90	17.81	6.67	10.78	54.84

5.3 Progress toward QoS Optimality

An *optimal prefix* counts consecutive leading objectives matched to the optimum. Of 960 valid, non-optimal plans, 475 attain the first-priority optimum, including 159 that attain the first two. Their alternatives preserve these attained priorities while improving later objectives.

First-difference gaps quantify the remaining loss. Among plans first differing in cost, median excess is 4.90% for DeepSeek and 45.43% for Qwen. These gaps distinguish small residual losses from substantial departures (Appendix D).

Rank counts better QoS levels, grouping plans with equal objective vectors. Verified alternatives place at least 595 of these 960 plans (about 62%) outside the top three levels (Appendix D.3). Exact optimality supplies the endpoint; prefixes, gaps, and ranks show which priorities to preserve and how much improvement remains.

5.4 From Local Choices to Whole-Graph Search

Of the 960 verified alternatives, 65.21% use the same number of calls or more. Better QoS therefore does not require fewer calls.

Eight selected, complete reasoning texts suggest goal-directed subgraph expansion: trace outputs backward, add prerequisites, compare alternatives, and revisit choices. We ground this interpretation in the submitted graphs and diagnoses (Turpin et al., 2023); Appendix D.6 presents the cases and selection.

A DeepSeek case shows why revisiting matters. The original trace favors a 7-unit branch over an 8-unit alternative although the whole graph takes 8. The search-prompt answer uses slower, cheaper implementations for two non-critical copies, saving 10 cost units. Replacing separate record reads $(6 + 6)$ with a shared, accelerated full read and redaction $(7 + 2)$ saves another 3. Cost falls from 55 to 42 with 11 calls, latency 8, and unchanged reliability.

Other cases involve conflicting allocations, incorrect reliability comparisons, and unnecessary speed on non-critical branches. Local choices must remain consistent with the evolving graph: a replacement can change sharing, downstream readiness, and joint task satisfaction. Effective search must track these relationships while revisiting alternatives, motivating explicit candidate comparison and dependency checks (Section 6).

6 Improving QoS Awareness

We develop two training-free approaches to improving service choices: **QoS-aware search prompting** and **prompt-guided tool assistance**. Both support the *Agent planning* block of Figure 2, using the same task inputs and whole-graph evaluation.

Table 4: Search-prompt results under Thinking High (%). ΔO is the matched-task change from the original prompt (pp). Average weights models equally; bold marks column maxima.

Model	$F \uparrow$	$O \mid F \uparrow$	$\boldsymbol{O} \uparrow$	$\Delta O \uparrow$
GLM-5.2	90.48	57.01	51.59	$+\mathbf{13.86}$
GPT-5.6 Terra	49.83	35.17	17.53	-9.28
Qwen3.5-Flash	10.03	17.24	1.73	$+0.35$
Gemini-3.1-Pro	73.67	53.85	39.67	$+3.00$
DeepSeek-V4-Pro	**95.68**	**74.62**	**71.40**	$+9.53$
Average	63.94	47.58	36.38	$+3.49$

6.1 Guiding Search with a Fixed Prompt

The reasoning analysis (Section 5.4) motivates revisiting local choices as sharing, timing slack, and cross-branch constraints emerge. Our prompt keeps alternatives available for whole-graph revision, drawing on explicit backtracking in Tree of Thoughts (Yao et al., 2023) and constraint-guided verification in PlanGEN (Parmar et al., 2025).

Models are instructed to **represent** joint goals and prerequisites, **expand** alternative producers with backtracking, **check** input sources and cross-branch or downstream obligations after replacements, and **compare** complete feasible graphs, including sharing and parallelism, under the stated QoS priorities. The same 385-word instruction serves five models, two scenarios, and one to three objectives. Task inputs supply the rules; no candidates, optimum values, or auxiliary tools are provided (Appendix E).

Under Thinking High, optimal plans increase from 954 to 1,056 across 2,915 matched pairs, a 3.50-point gain, while functional successes change from 1,874 to 1,864 (Table 4). Among 1,586 both-valid pairs, 257 become optimal and 160 lose optimality, accounting for 97 of the 102 net additional optimal plans. The observed net gain lies mainly in service choice beyond task completion.

For DeepSeek, 39 of 53 net additional optimal plans come from three-objective tasks, where O rises by 22.16 points. GLM and DeepSeek improve mainly within valid plans; Gemini's gain comes from functional transitions. In the same-serving-configuration subsets, GLM and DeepSeek also show positive ΔO: 16.13 points over 124 pairs and 6.94 over 144 (Appendix E).

Under Thinking Off, functional successes rise from 402 to 438 on 2,393 pairs, but optimal plans stay at 22: 172 of 180 newly valid plans remain non-optimal. This gap motivates explicit search support (Appendix E).

6.2 Prompt-Guided Graph-Search Assistance

CodeAct couples executable actions with interpreter feedback (Wang et al., 2024). Our tool-use prompts guide agents to express goals, prerequisites, and constraints as a search specification, then check and incorporate the returned candidates. Between these stages, code performs expansion, binding, and QoS calculation.

The operation `search_dependency_graphs` accepts initial values, joint goals, action rules, and optional QoS measures and priorities. It matches attributes, instantiates compatible calls, and returns up to three DAGs with input sources and aggregate QoS, counting shared calls once. Search uses the model's specification without author plans, benchmark optima, or the evaluator. Candidates can cover a full task or a partial construction; the agent must complete remaining obligations and check the resulting graph. Added calls can change sharing, critical paths, and total QoS. New environments and priorities are supplied through inputs; search rules can evolve in external code.

R receives the *actual search response* (candidates or an error); E receives a *withheld-result marker*. Both continue the same parameter-generation and applicable feedback history to submit final graphs. We analyze 2,997 complete Thinking High pairs to study feedback use after shared modeling (Appendix E.5).

Table 5: Graph-search response exposure under Thinking High (%). E and R share the search-modeling stage; E withholds the response and R shows it. $\Delta O = O_R - O_E$ (pp). Average weights models equally; bold marks column maxima.

	$F \uparrow$		$\boldsymbol{O} \uparrow$		
Model	E	R	E	R	$\Delta O \uparrow$
GLM-5.2	80.17	78.33	42.17	38.83	-3.33
GPT-5.6 Terra	57.62	54.44	17.92	19.43	$+\mathbf{1.51}$
Qwen3.5-Flash	0.67	0.00	0.00	0.00	0.00
Gemini-3.1-Pro	73.50	71.17	31.50	31.67	$+0.17$
DeepSeek-V4-Pro	**95.83**	**92.33**	**62.50**	**58.33**	-4.17
Average	61.56	59.25	30.82	29.65	-1.17

6.3 From Search Candidates to Better Plans

The search program returns candidates for 78.00% of tasks overall and 97.38% outside Qwen, using the agents' specifications. R and E produce 889 and 924 optimal plans, and 1,776 and 1,845 valid plans, respectively (Table 5). Among 1,516 both-valid pairs, R gains optimality in 183 cases and loses it in 180; exact QoS vectors favor R in 388 and E in 386. Candidate discovery must be coupled with faithful task modeling and reliable revision of the final plan.

Strong planners have more correct plans to preserve than failures to repair. DeepSeek's R answers are valid on 19 of the 25 tasks where E fails (76%), but invalid on 40 of E's 575 successes (about 7%). A small error rate on the much larger valid subset can outweigh a high repair rate on failures. Preserving correct decisions is therefore as important as finding new candidates.

Formulation and integration must preserve task meaning. A GLM candidate exposes a shared read that, with another allocation change, reduces cost from 35 to 33 at unchanged latency and reaches the optimum. Conversely, DeepSeek omits an exact record-set requirement from its specification: the candidate solves the weaker goal, and the final graph skips a required filter. Search succeeds on the supplied model while the original task fails (Appendix E.7).

These findings motivate local, checkable revisions: replace the affected subgraph, preserve other calls and bindings, and reassess task completion and whole-graph QoS before accepting the change. A revision should retain achieved higher-priority optima (Section 5.3). An interface that exposes covered goals, changed calls, and service differences would help the agent decide whether to adopt a candidate, keeping the existing plan as a fallback. The acceptance check would use the original task contracts and compare every requested objective on the completed graph, including branches outside the search specification.

Prompting directs what to explore and preserve; code handles expansion and arithmetic. Their joint design offers a practical direction for QoS-aware planning, with search rules and checks that can be updated as tasks change without retraining the agent. Task changes can be reflected in the search specification; new constraint types or QoS aggregators can be implemented and tested in the tool.

7 Conclusion

AgentQoSBench evaluates QoS awareness through complete tool-call graphs, adapting QoS-aware service composition to autonomous planning under complex dependencies and distracting alternatives. QoSGen provides multi-solution tasks with exact optima under declared service models. Across five models, nearly half of valid plans remain non-optimal, revealing substantial room for improvement beyond task completion. Search prompting improves service selection for GLM and DeepSeek, while tool experiments highlight the need to preserve task requirements when formulating searches and integrating candidates. These findings motivate whole-graph comparison and prompt-guided tool collaboration that better reflect users' service priorities.

AI USE STATEMENT

AI assistants supported data synthesis, candidate generation, semantic review, literature verification, experimental analysis, manuscript polishing, and figure and table design. LLMs reviewed each task and its answer against public descriptions and tool contracts; reported scores were computed programmatically. Research-assistance model calls were handled separately from model responses collected for benchmark evaluation.

ETHICS STATEMENT

AgentQoSBench uses synthetic instances and source-inspired contracts to study planning without executing real consumer or workplace operations. Service-quality values are explicitly modeled. Source licensing and redistribution conditions are documented in the benchmark materials.

REPRODUCIBILITY STATEMENT

The anonymous supplement contains the complete benchmark (600 tasks and 379 tool worlds), task definitions and prompts, generator and evaluator source code, construction and experimental configurations, proof records, and author-constructed controls. It also includes versioned graph-search implementations, exact tool-use and parameter-revision prompts, search-budget configurations, and an offline demonstration of the paired real/withheld-result protocol (Appendix E.5). The package provides three runnable evaluation examples (Appendix B.1), usage instructions, dependencies, and licensing information to help readers inspect the benchmark, evaluate their own tool-call graphs, and run the graph-search tool locally.

REFERENCES

Kinjal Basu, Ibrahim Abdelaziz, Kiran Kate, Mayank Agarwal, Maxwell Crouse, Yara Rizk, Kelsey Bradford, Asim Munawar, Sadhana Kumaravel, Saurabh Goyal, Xin Wang, Luis A. Lastras, and Pavan Kapanipathi. NESTFUL: A benchmark for evaluating LLMs on nested sequences of API calls. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 33538–33547, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1702. URL https://aclanthology.org/2025.emnlp-main.1702/.

Zhi-Kai Chen, Xu-Xiang Zhong, Song-Yan Li, De-Chuan Zhan, and Han-Jia Ye. Peakbench: Benchmarking resource-aware tool invocation in llm agents. *arXiv preprint arXiv:2608.24509*, 2026.

Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340. Springer, 2008.

Dinesh Khandelwal, Gnana Prakash Punnavajhala, GPS Bhargav, Gaurav Pandey, Sachin Joshi, Hima Karanam, and Dinesh Raghu. Toolweave: Structured synthesis of complex multi-turn tool-calling dialogues. *arXiv preprint arXiv:2605.12521*, 2026.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. An llm compiler for parallel function calling. *arXiv preprint arXiv:2312.04511*, 2023.

Seungkyu Lee, Nalim Kim, and Yohan Jo. In-n-out: A parameter-level api graph dataset for tool agents. *Transactions of the Association for Computational Linguistics*, 14:1008–1027, 2026.

Jiayu Liu, Cheng Qian, Zhaochen Su, Qing Zong, Shijue Huang, Bingxiang He, and Yi R Fung. Costbench: Evaluating multi-turn cost-optimal planning and adaptation in dynamic environments for llm tool-use agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12826–12858, 2026a.

Liangliang Liu, Yanming Li, Yigang Liu, Jialong Han, Rujia Shen, Yi Guan, Yi Lin, and Jingchi Jiang. Feasible is not enough: Cost-aware optimal tool-chain planning on multi-solution tool graphs. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17387–17403, 2026b.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, et al. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1160–1183, 2025.

Mihir Parmar, Xin Liu, Palash Goyal, Yanfei Chen, Long Le, Swaroop Mishra, Hossein Mobahi, Jindong Gu, Zifeng Wang, Hootan Nakhost, Chitta Baral, Chen-Yu Lee, Tomas Pfister, and Hamid Palangi. PlanGEN: A multi-agent framework for generating planning and reasoning trajectories for complex problem solving. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20640–20666, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1042. URL https://aclanthology.org/2025.emnlp-main.1042/.

Shuofei Qiao, Runnan Fang, Zhisong Qiu, Xiaobin Wang, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Benchmarking agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pp. 69679–69703, 2025.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Beyond accuracy: Behavioral testing of nlp models with checklist. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pp. 4902–4912, 2020.

Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. Taskbench: Benchmarking large language models for task automation. *Advances in Neural Information Processing Systems*, 37:4540–4574, 2024.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems*, 36:74952–74965, 2023.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. *arXiv preprint arXiv:2402.01030*, 2024.

Zezhong Wang, Xingshan Zeng, Weiwen Liu, Liangyou Li, Yasheng Wang, Lifeng Shang, Xin Jiang, Qun Liu, and Kam-Fai Wong. Toolflow: Boosting llm tool-calling through natural and coherent dialogue synthesis. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 4246–4263, 2025.

Xiaolong Wei, Yuehu Dong, Xingliang Wang, Xingyu Zhang, Zhejun Zhao, Dongdong Shen, Long Xia, and Dawei Yin. Beyond react: A planner-centric framework for complex tool-augmented llm reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 33845–33853, 2026.

Duo Wu, Jinghe Wang, Yuan Meng, Yanning Zhang, Le Sun, and Zhi Wang. Catp-llm: Empowering large language models for cost-aware tool planning. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 8699–8709. IEEE, 2025.

Xixi Wu, Yifei Shen, Caihua Shan, Kaitao Song, Siwei Wang, Bohang Zhang, Jiarui Feng, Hong Cheng, Wei Chen, Yun Xiong, et al. Can graph learning improve planning in llm-based agents? *Advances in Neural Information Processing Systems*, 37:5338–5383, 2024.

Ruixuan Xiao, Wentao Ma, Ke Wang, Yuchuan Wu, Junbo Zhao, Haobo Wang, Fei Huang, and Yongbin Li. Flowbench: Revisiting and benchmarking workflow-guided planning for llm-based agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 10883–10900, 2024.

Hanwen Xu, Xuyao Huang, Yuzhe Liu, and Zhijie Deng. Tps-bench: Evaluating ai agents' tool planning & scheduling abilities in compounding tasks. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 34949–34961, 2026.

Yong Yang, Marlon Dumas, Luciano García-Bañuelos, Artem Polyvyanyy, and Liang Zhang. Generalized aggregate quality of service computation for composite services. *Journal of Systems and Software*, 85(8):1818–1830, 2012.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Junjie Ye, Zhengyin Du, Xuesong Yao, Weijian Lin, Yufei Xu, Zehui Chen, Zaiyuan Wang, Sining Zhu, Zhiheng Xi, Siyu Yuan, et al. Toolhop: A query-driven benchmark for evaluating large language models in multi-hop tool use. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2995–3021, 2025.

Liangzhao Zeng, Boualem Benatallah, Anne HH Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. Qos-aware middleware for web services composition. *IEEE Transactions on software engineering*, 30(5):311–327, 2004.

Liangzhao Zeng, Anne HH Ngu, Boualem Benatallah, Rodion Podorozhny, and Hui Lei. Dynamic composition and optimization of web services. *Distributed and Parallel Databases*, 24(1):45–72, 2008.

Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Anne HH Ngu. Qos-aware service composition: A retrospective. *IEEE Transactions on Software Engineering*, 51(3):836–841, 2025.

Yinger Zhang, Shutong Jiang, Renhao Li, Jianhong Tu, Yang Su, Lianghao Deng, Xudong Guo, Chenxu Lv, and Junyang Lin. Deepplanning: Benchmarking long-horizon agentic planning with verifiable constraints. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7377–7407, 2026.

A Evaluation Protocol and Supplementary Materials

Inputs and submissions. All results use the final 600-task AgentQoSBench package. Each independent request supplies the English task, full applicable world (454–534 tools; 249,205–323,043 characters), source guarantees, and ordered objectives. Author plans, optima, and proof records are hidden. Source descriptions specify the guarantees of retrieval calls; source artifacts become available through those calls.

The answer is one JSON object containing `dag`, with `nodes`, `public_bindings`, `literal_bindings`, `data_edges`, `control_edges`, and `final_outputs`; empty relation arrays may be omitted. Nodes have unique case-sensitive IDs without dots, registered tools, and optional declared implementations. Each input binds exactly once to an available public input, a whole upstream `result`, or a permitted integer literal. Typed adapters handle conversions; implicit field projections are unavailable. Omitted implementations use the public default; null, empty, and undeclared values are invalid. Implementation tables give absolute QoS values. Admission accepts a unique unchanged JSON graph and rejects ambiguous alternatives.

Experimental conditions. Original prompting uses the public user message; search prompting prepends the instruction in Appendix E. Requests use streaming Chat Completions, temperature 1, independent contexts, and no application-level output-token cap. Final answers are scored; returned reasoning is retained separately. Thinking High requests `reasoning_effort=high`; Thinking Off uses model-specific disabling controls documented in the supplement's `EXPERIMENT_CONFIG.md`. Gemini is evaluated with Thinking High.

Coverage and recovery. Each model–task–condition cell contributes one retained complete answer. Recovery targets missing answers independently of scores. Table 6 accounts for the 10,705 included answers and 95 missing active cells. Rates use included answers; paired comparisons match model and task IDs with determinate outcomes on both sides. The latency objective is the planned graph's makespan.

Table 6: Coverage of the adopted four-condition results. Missing active cells are excluded from rate denominators; Gemini off is not applicable.

Condition	Models	Planned	Included	Missing
Original, Thinking High	5	3,000	3,000	0
Search, Thinking High	5	3,000	2,915	85
Original, Thinking Off	4	2,400	2,393	7
Search, Thinking Off	4	2,400	2,397	3
Total		10,800	10,705	95

Scoring. The sequential checks are **Struct** (representation and acyclicity), **Call** (tools and bindings), **Dep** (semantic prerequisites), **Goal** (required results and states), and **QoS** (specified optimum). For answer i,

$$f_i = \mathrm{Struct}_i \wedge \mathrm{Call}_i \wedge \mathrm{Dep}_i \wedge \mathrm{Goal}_i, \tag{3}$$

$$o_i = f_i \wedge \mathrm{QoS}_i. \tag{4}$$

Thus $n_F = \sum_i f_i$ and $n_O = \sum_i o_i$ in Equation 2; stored fields `F` and `S` encode these outcomes. When $n_F = 0$, $O \mid F$ is undefined. First-failure analysis records the earliest failed check; later checks are blocked. Prefix and gap analyses use valid, non-optimal plans. The four conditions share public inputs and scoring. Conditional rates use each condition's functional subset; paired rates use the shared determinate subset.

Review package. The anonymous supplement includes 600 public inputs and 379 full tool worlds; the manifest and evaluation definitions; saved proofs and author controls; exact prompts; construction and experimental configurations; and construction and scoring code. The README, data card, experiment configuration, evaluator and generator guides, license notices, and check summaries describe how to inspect the benchmark and score reader-supplied answers. Appendix B.1 provides three runnable controls with expected outcomes.

B QoSGen Construction Controls

Task distribution. Table 7 reports the scenario and objective-count distribution. Single-, double-, and triple-objective tasks each number 200, covering all 15 nonempty ordered selections of latency, cost, and reliability. Each instance provides 454–534 tools; mechanism and composition reuse are described below.

Table 7: AgentQoSBench tasks by scenario and number of ordered QoS objectives.

Scenario	Tasks	One	Two	Three
Consumer	298	100	100	98
Workplace	302	100	100	102
Total	600	200	200	200

Table 8: Construction-side strategy checks under the information conditions described below. Not found means that the bounded search did not locate an optimum.

Diagnostic	Optimal	Not found	Not tried	Unresolved
Public-closure-assisted greedy	49	551	0	0
Fixed public graph, implementation tuning	59	215	285	41
Author short graph with implementation tuning	153	332	0	115

Composition and reuse. The 600 configurations use 32 cards and 32 recipes, 215 recipe-plus-additional-card sets, and 379 full worlds. There are 503 mixed and 97 independent compositions, with 599 distinct goal definitions after removing titles and IDs; one pair differs in objective order. Each task requests 4–26 cards, while its world retains all 32 card carriers. Saved attaining graphs contain 8–102 calls (mean 40.805).

Alternatives include card substitutions and mixed combinations. Functional signatures distinguish producing rules and bindings, ignoring node names and implementations. Public contracts separate documented behavior from synthetic relations, adapters, and modeled service properties. Independent components need no artificial edges; related components retain actual state and cross-output obligations.

Validation controls. Following capability-oriented behavioral testing (Ribeiro et al., 2020), controls target explicit obligations. Validation covers all 600 tasks, with 21,756 positive replay records and 22,733 negative semantic controls (6,894 module-task and 15,839 complete-task cases). Positive counts refer to replay records, including saved alternatives and attaining graphs. Negative controls are judged against their associated tasks; complete-task controls cover Dep or Goal failures for all 32 cards.

The 1,200 priority witnesses tie all earlier objectives while worsening the tested priority. They confirm that each specified objective distinguishes complete functional plans. The validation suite records 685 passing test methods and 2,400 scoring fixtures: an optimum, a non-optimum, an omitted-goal graph, and an explicit-null implementation case for every task. All fixtures match their expected outcomes; 1,800 independent rational-arithmetic checks agree with the QoS aggregates. All 600 public presentations reconstruct their underlying definitions. After programmatic generation and scoring, item-by-item LLM review assessed every task and answer against the public descriptions and contracts.

Static strategy diagnostics. Table 8 summarizes three construction-side searches. Closure-assisted greedy receives legal bindings; fixed-graph tuning starts from a discovered public graph; author-short-graph tuning receives a simplified author plan. Each diagnostic tests simplification under its own supplied information and task subset.

Legal broad reads, integrated operations, sharing, and native transformations are accepted. These diagnostics test construction-side simplifications; model interventions are evaluated in Section 6.

B.1 Three Executable Reader Examples

The supplement provides three author controls for the cost-minimizing task `Q600V6-1-001`. It combines a recipient-specific dispatch, an approved meeting brief with participant actions, and a service-case handoff. Inputs and definitions are in `benchmark/`; answers and expected outcomes are in `evaluator/examples/`.

From the supplement root, use Python 3.11 or newer with its documented environment and a new output directory:

```
python -B -X utf8 evaluator/examples/run_examples.py --out demo
```

The command scores all three answers, writes verdicts and `summary.json`, and prints `status: PASS` when they match `expected.json`, including the two intentional failures in Table 9.

Table 9: Expected outcomes for three author controls on one task. P: pass; F: fail; —: blocked after an earlier failure. These examples do not enter model averages.

Case / answer filename	Struct	Call	Dep	Goal	QoS
`optimal.json`	P	P	P	P	P
`nonoptimal.json`	P	P	P	P	F
`goal_failure.json`	P	P	P	F	—

Valid and optimal. `optimal.json` is the saved attaining graph. Its 14 calls deliver the three required outputs at cost 23 and makespan 13. All checks pass; cost matches the certified optimum.

Valid but non-optimal. `nonoptimal.json` changes six implementations while retaining the 14 calls, bindings, dependencies, and outputs. Cost becomes 28 and makespan 16; only cost is optimized. QoS fails, and `counterexample.plan` returns the complete `optimal.json` graph, saving exactly 5. This pair isolates service selection.

A violated public goal. `goal_failure.json` retains all 14 calls but omits `final_outputs`, violating the designated-recipient dispatch obligation `roles:dispatch`. The evaluator reports `RESULT_OBLIGATION` at Goal and blocks QoS. The task requires its computed results to appear among the designated final outputs. The supplement's `CHECKS.md` records that all three controls match these expectations.

C Exact Optimality: Coverage and Attainment

The v6 certificates establish optimality over the supported public semantic domain through QoS-preserving graph reduction, finite closure, exact optimization, and replay of an attaining graph.

C.1 Coverage Property and Observable Obligations

Let $\mathcal{N}_x \subseteq \mathcal{F}_x$ denote the materializable normal-form plans for an admitted instance. The required property is

$$\forall P \in \mathcal{F}_x \; \exists P' \in \mathcal{N}_x : \quad L(P') \le L(P), \quad C(P') \le C(P), \quad R(P') \ge R(P). \tag{5}$$

Componentwise non-worsening preserves the optimum under every supported priority order. The property concerns optimality; suboptimal levels and complete top-k rankings require separate analysis.

With nonnegative costs and durations and reliabilities at most one, deleting irrelevant calls cannot worsen QoS. Relevance includes output ancestry, preconditions, observed identity and multiplicity,

and terminal state or resource effects. Pure-call dominance replaces a call only with one producing identical values from a subset of its inputs and resources, at no worse cost, latency, or reliability. Required distinctness and necessary state transitions are retained; the scorer checks and charges every submitted call.

C.2 Finite Values and Fresh Occurrences

Closure propagates values from public inputs and declared retrievals through relevant producers and compatible bindings in the full tool world, to a fixed point. Values are equal only when their complete canonical records agree, including identities, provenance, versions, and cross-value links. Table canonicalization removes row-order differences but preserves duplicate rows and nonempty provenance.

Occurrence budgets are computed separately by output type. A fresh result contributes one occurrence; demands sum across a producer's input positions, take a componentwise maximum across alternative producers, then sum across goals weighted by their required counts. Grounding uses these per-type budgets for fresh slots; physical allocations are also bounded by capacity. Fresh names use the result body and slot; physical resource IDs and copied references remain part of the value.

Checked finite identity inequalities and required copies are supported: 265 tasks require multiplicity greater than one with ID distinctness. Fifty-four have top-level joint relations, all equality. Identity-producing cycles lacking an independent finite demand bound and literal generated-ID requirements remain outside the grammar.

C.3 Restricted State and Resource Effects

Supported effects on existing objects comprise a single final patch value, declared revision-sensitive approval/publication, retirement/replacement, and fixed positive consumption of monotonically decreasing quotas. Actual object and resource identities stay linked; 228 tasks have explicit terminal-state goals.

Multiple different patches on one object, generated-object writes, incompatible write–read return types, multiple historical-state observations, exact terminal-credit equalities, and mutable-revision inequalities require further arguments. Arbitrary state machines, scalar graph parameters, and runtime-parameterized QoS remain uncertified.

Replay checks validity under every order permitted by data/control dependencies. It explores conflicting state-operation orders and executes commuting operations without enumerating their permutations. A violating order is returned as evidence; exhausting the state budget returns unresolved.

C.4 Ordered-Document Extension

Fifteen tasks use the checked ordered-document extension. Public reads yield finite positive intervals of a fixed package edition; initial segments are permitted, while v6 starts from retrieval calls. Concatenation requires adjacent nonoverlapping segments with matching output type, package, version, and length, plus pairwise distinct volume object–version pairs across both inputs. It copies their ordered volume records, preserving file identities, versions, chunks, and positions.

Each concatenation strictly increases interval length within the public package bound. Finite seeds and copy-only operations bound interval values and payload combinations. All declared producers of relevant segment types must satisfy the grammar; otherwise coverage is unresolved. This semantic closure includes whole reads and other legal complete constructions.

C.5 Exact Solving and Attainment

After closure completes, exact shared-action search or SMT optimizes the public objective order over selected implementations, shared producers, required resources, and dependency times; SMT uses Z3 (De Moura & Bjørner, 2008). Implementation pruning follows justified service dominance. Search ranges over public producers, observable identities, and compatible bindings; computation-budget truncation leaves coverage unresolved.

For stateful tasks, the search domain relaxes execution order rather than enumerating control-edge patterns. Given the coverage property, its exact optimum is a lower bound in the ordered minimization representation. Certification requires a compiled DAG that passes functional replay under every permitted order and exactly matches the solver's cost, latency, and reliability. An unattained bound remains unresolved.

Stored verification records cover all 600 tasks: 87 use exact search and 513 use SMT. Their normal-form records identify finite values, observed fresh occurrences, and single final patches, with the ordered-document extension where applicable. The attaining graph is replayed and its three QoS aggregates are checked exactly.

A single complete, functionally valid graph with a strictly better QoS vector under the task's priority order disproves a submitted graph's optimality. That counterexample need not itself be globally optimal. In the main evaluation, all 960 counterexamples also attain the reference objectives, enabling gap-to-optimum analysis. Certifying those reference values is the separate role of coverage, exact solving, and attainment.

D Extended Main-Evaluation Results

These results use 3,000 original-prompt, Thinking High answers: 600 complete, scorable answers per model. The slices cover the benchmark's scenarios and objective-count groups. Within comparable groups, bold marks column maxima, Average weights models equally, and arrows indicate preferred directions.

D.1 Scenarios and Objective Counts

Table 10: Results by scenario. N counts answers; n_F and n_O count valid and optimal plans. Maxima and averages are computed within each scenario.

Model	Scenario	N	n_F	n_O	$O \mid F \uparrow$ (%)
GLM-5.2	Consumer	298	264	116	43.94
GLM-5.2	Workplace	302	263	111	42.21
GPT-5.6 Terra	Consumer	298	173	75	43.35
GPT-5.6 Terra	Workplace	302	198	84	42.42
Qwen3.5-Flash	Consumer	298	20	1	5.00
Qwen3.5-Flash	Workplace	302	36	7	19.44
Gemini-3.1-Pro	Consumer	298	194	113	58.25
Gemini-3.1-Pro	Workplace	302	213	107	50.23
DeepSeek-V4-Pro	Consumer	298	**285**	**183**	**64.21**
DeepSeek-V4-Pro	Workplace	302	**287**	**176**	**61.32**
Average	Consumer	—	187.20	97.60	42.95
Average	Workplace	—	199.40	97.00	43.13

Conditional optimality falls across objective-count groups despite stable functional success (Table 11). Pooled overall optimality is similar for consumer and workplace tasks (32.75% and 32.12%).

Single-objective latency, reliability, and cost groups contain 335, 330, and 335 answers across five models. Their pooled $O \mid F$ is $199/204 = 97.55\%$, $168/199 = 84.42\%$, and $107/207 = 51.69\%$, respectively. These denominators count valid plans within each task group.

D.2 Optimal Prefixes and Gap Magnitudes

All 960 valid, non-optimal answers, spanning 448 tasks, have a complete better graph matching the stored optimum on every specified objective. Independent exact-rational checks reproduce costs, reliability products, and dependency-based finish times for all 1,933 valid original graphs and 960 alternatives. Table 12 locates each answer's first differing priority j; its optimal-prefix length is $j - 1$. Thus 475 answers attain the first objective, including 159 that attain the first two.

Table 11: Results by objective count. Maxima and averages are computed within each group; the balanced groups contain different tasks.

Model	Objectives	N	n_F	n_O	$O \mid F \uparrow$ (%)
GLM-5.2	1	200	172	128	74.42
GLM-5.2	2	200	178	66	37.08
GLM-5.2	3	200	177	33	18.64
GPT-5.6 Terra	1	200	99	75	75.76
GPT-5.6 Terra	2	200	145	63	43.45
GPT-5.6 Terra	3	200	127	21	16.54
Qwen3.5-Flash	1	200	20	6	30.00
Qwen3.5-Flash	2	200	21	2	9.52
Qwen3.5-Flash	3	200	15	0	0.00
Gemini-3.1-Pro	1	200	129	95	73.64
Gemini-3.1-Pro	2	200	134	80	59.70
Gemini-3.1-Pro	3	200	144	45	31.25
DeepSeek-V4-Pro	1	200	**190**	**170**	**89.47**
DeepSeek-V4-Pro	2	200	**192**	**126**	**65.63**
DeepSeek-V4-Pro	3	200	**190**	**63**	**33.16**
Average	1	—	122.00	94.80	68.66
Average	2	—	134.00	67.40	43.08
Average	3	—	130.60	32.40	19.92

Table 12: First differing priority relative to the optimum. Every earlier priority is exactly tied. Counts are model–task cells; Average is a mean count.

Model	First	Second	Third	Total
GLM-5.2	**172**	**102**	26	**300**
GPT-5.6 Terra	105	81	26	212
Qwen3.5-Flash	40	8	0	48
Gemini-3.1-Pro	100	52	35	187
DeepSeek-V4-Pro	68	73	**72**	213
Total	485	316	159	960
Average	97.00	63.20	31.80	192.00

For the first differing objective d, cost and latency gaps are $100(q_d(P)-q_d^*)/q_d^*$ percent; all reference denominators here are positive. Reliability shortfall is $100(R^* - R(P))$ percentage points. Table 13 keeps these units separate and uses linearly interpolated quantiles at index $(n-1)p$. Gaps summarize the first differing priority; exact equality remains the success criterion.

D.3 Ranking-Based Optimization Headroom

Complete valid plans are ranked by the task's ordered objective vector, with equal vectors forming one QoS level. Evidence pools same-task valid plans from all four prompt–thinking conditions, validated author plans, and optimality witnesses. At least three distinct, strictly better levels certify exclusion from Top-3. Figure 3 reports this share among valid, non-optimal original-prompt Thinking High plans: at least 595/960 (61.98%) overall. The exclusion shares are lower bounds based on witnessed alternatives, not exhaustive ranking: optimum-preserving reductions need not preserve suboptimal levels. Rank counts better levels; the first-difference gap measures distance from the optimum.

D.4 First Failures and Complementary Successes

Table 14 assigns each unsuccessful answer to its first failed check. Later checks are blocked, so counts locate the first detected violation rather than all defects in an answer.

Table 13: First-difference gaps among valid, non-optimal plans: cost and latency in percent, reliability in percentage points. Maxima are within metrics; Average averages model statistics, not pooled answers.

Model	Metric	n	Min.↓	Q25↓	Median↓	Q75↓	Max.↓
GLM-5.2	Cost	171	0.70	3.01	7.02	12.92	63.64
	Latency	16	7.14	8.90	12.50	18.18	33.33
	Reliability	113	0.29	0.79	1.58	2.72	30.22
GPT-5.6 Terra	Cost	139	0.71	4.17	8.00	16.70	**97.33**
	Latency	24	4.76	8.90	16.23	21.25	33.33
	Reliability	49	0.43	1.49	2.38	4.50	30.94
Qwen3.5-Flash	Cost	22	**9.76**	**34.56**	**45.43**	**55.76**	85.29
	Latency	8	**11.11**	17.21	20.89	25.00	30.00
	Reliability	18	**1.30**	**2.51**	**18.12**	**29.93**	**36.25**
Gemini-3.1-Pro	Cost	93	0.68	3.13	6.67	23.42	63.33
	Latency	20	8.33	**19.06**	**21.11**	**72.73**	**100.00**
	Reliability	74	0.27	0.68	1.35	3.22	35.18
DeepSeek-V4-Pro	Cost	143	0.45	2.59	4.90	10.19	50.49
	Latency	18	6.25	18.18	19.35	19.35	37.50
	Reliability	52	0.40	0.59	1.96	3.34	15.80
Average	Cost	—	2.46	9.49	14.40	23.80	72.02
Average	Latency	—	7.52	14.45	18.02	31.30	46.83
Average	Reliability	—	0.54	1.21	5.08	8.74	29.68

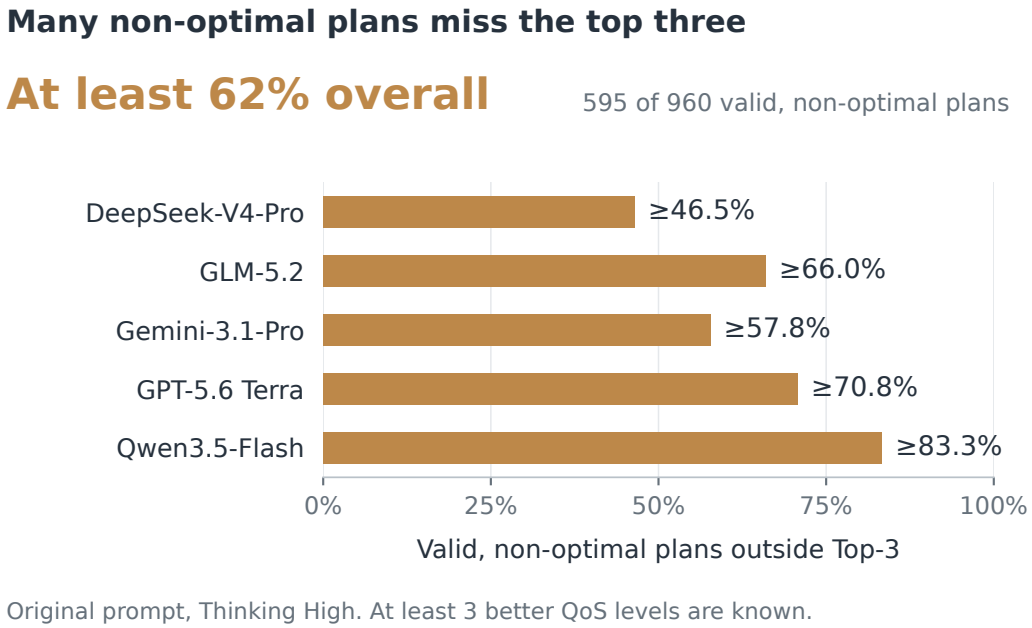

Figure 3: Many valid, non-optimal plans lie outside Top-3 under the original prompt and Thinking High. Bars show each model's exclusion share; ≥ marks a lower bound witnessed by at least three distinct, strictly better QoS levels.

At least one model attains the optimum on 436 tasks, all five on seven, and none on 164. DeepSeek, GLM, Gemini, Terra, and Qwen uniquely solve 81, 22, 17, 13, and zero tasks. This union summarizes complementary successes across the models.

Table 14: First-failure and success counts among 600 answers per model. Each failed answer is counted once; later checks are blocked. Average is a mean count.

Model	Struct	Call	Dep	Goal	QoS	Success↑
DeepSeek-V4-Pro	0	9	2	17	213	**359**
GLM-5.2	4	41	16	12	**300**	227
Gemini-3.1-Pro	18	9	**59**	**107**	187	220
GPT-5.6 Terra	8	137	27	57	212	159
Qwen3.5-Flash	**248**	**242**	39	15	48	8
Total	278	438	143	208	960	973
Average	55.60	87.60	28.60	41.60	192.00	194.60

On the 41 tasks where all five models produce valid plans, GLM, Terra, Qwen, Gemini, and DeepSeek optimize 25, 25, 8, 33, and 34. On the 339 tasks where both GLM and Terra produce valid plans, they optimize 145 and 151, respectively.

D.5 Public-Contract Checks for Semantic Cases

Three cases pass Struct and Call but fail Dep. GLM's `Q600V6-1-190` maps column names and imports quantities still measured in cases into a destination requiring pieces (`IMPORT_UNIT`). Terra's `Q600V6-1-072` connects allocation `d1` to `s1`, although the public relation permits only `s2` (`ALLOCATION_CONTINUITY`). Qwen's original-high `Q600V6-1-013-MV6` preserves people and shares an equipment check, but requests room quantities 2 and 1 against capacity 2 (`JOINT_CAPACITY`). These cases distinguish interface legality from unit conversion, allocation continuity, and joint feasibility; Goal and QoS remain unassessed.

D.6 Graph-Level Improvements and Public Reasoning Cases

Among 960 saved better graphs, 334 use fewer calls, 501 the same number, and 125 more: 626/960 (65.21%) improve without reducing calls. Across four conditions, these counts are 1,272, 1,026, and 268 over 2,566 comparisons. Each comparison uses a saved pair of complete valid same-task graphs and counts every submitted node.

We selected eight complete publicly returned traces to examine contrasting mechanisms: GLM `2-198` and DeepSeek `3-024` under both high-prompt conditions; Qwen `1-013-MV6` and `3-003`; search-high Terra `2-166`; and original-high DeepSeek `1-006` (item suffixes). The case analysis combines these texts with the submitted graphs and diagnoses to interpret candidate expansion, sharing, comparison, and revision. "Complete" means all text in the selected returned field; Terra's is summary-shaped. Of 5,915 high-condition answers, 4,476 return nonempty reasoning text, none from Gemini.

Table 15: Same-task DeepSeek plans under reliability → latency → cost. Both are valid; quantities use the task's modeled units. Arrows indicate preferred QoS directions.

Submitted plan	Calls	Reliability↑	Latency↓	Cost↓
Original prompt	11	0.99^{11}	8	55
Search prompt	11	0.99^{11}	8	42

Whole-plan context changes a candidate's value. In DeepSeek's `Q600V6-3-024` (Table 15), the original trace considers a full-record-then-redact route but favors another branch taking 7 rather than 8, overlooking the complete plan's existing latency of 8. The search answer exploits this slack: it keeps the delivery-critical copy fast but uses slower, cheaper implementations for two independent final copies, saving 10 cost units. Replacing separate record reads $(6 + 6)$ with an accelerated full read and redaction $(7 + 2)$ saves another 3. These savings, computed from the submitted calls, preserve call count and higher-priority objectives.

A discovered alternative can be compared incorrectly. GLM's original-high `Q600V6-2-198` compares a one-call native index with a two-call alternative but writes $0.9801 > 0.99$. Under reliability before latency, its valid 13-call graph is non-optimal ($R = 0.99^{13}$, $L = 12$, $C = 63$). The search-prompt graph passes with 12 calls ($R = 0.99^{12}$, $L = 12$, $C = 61$). Here the candidate was discovered; its comparison was wrong.

Local speed need not improve the critical path. Terra's search-high `Q600V6-2-166` describes choosing the fastest implementation at each node. Four final allocation calls could instead trade latency 2 for 3 and reliability 0.970299 for 0.99 without exceeding the graph's latency of 9. Its valid answer has 75 calls, reliability about 0.262713, and cost 275; the saved better graph has 74 calls, the same latency, reliability about 0.429889, and cost 317. Cost is unranked under this task's latency-then-reliability objective. The saved graph changes both these allocations and other calls.

Revision must preserve final-output identity. DeepSeek's search-high `Q600V6-1-063` reads current ledgers but returns `previous_product_export`: its five rows match current values, yet version `previous` violates the required `v1`, causing Goal failure. The original-high answer passes. This graph-based diagnosis supplements the eight trace readings.

Sharing must respect distinct-object obligations. DeepSeek's successful original-high `Q600V6-1-006` reads the master once, creates all five required editable copies, and reuses one for delivery. Sharing the source preserves the distinct-output obligation.

E Training-Free Planning: Prompts, Tools, and Protocols

E.1 Configuration and Counts

Search conditions prepend the same 2,795-byte, 385-word English system instruction to unchanged public task inputs under high and off. It covers joint prerequisites, alternative-graph search, dependency checks, and whole-graph QoS comparison. All four conditions use streaming Chat Completions, temperature 1, no application-level output-token cap, and no auxiliary tools. Appendix E.5 describes the separate tool study. Bold marks model maxima; Average weights models equally.

Table 16: Full-condition counts. Each triplet is (N, n_F, n_O): included answers, functional successes, and joint successes. A dash indicates no included condition, not zero performance.

Model	Original Thinking High	Search Thinking High	Original Thinking Off	Search Thinking Off
GLM-5.2	600, 527, 227	599, 542, 309	600, 33, 0	600, 55, 4
GPT-5.6 Terra	600, 371, 159	582, 290, 102	593, 310, 17	597, 317, 17
Qwen3.5-Flash	600, 56, 8	578, 58, 10	600, 0, 0	600, 0, 0
Gemini-3.1-Pro	600, 407, 220	600, 442, 238	—	—
DeepSeek-V4-Pro	600, 572, 359	556, 532, 397	600, 59, 5	600, 68, 1

Table 16 retains 3,000/2,915/2,393/2,397 complete answers across the four conditions. Search–high lacks 18 Terra, one GLM, 22 Qwen, and 44 DeepSeek answers. Admission and scoring are shared; earlier benchmark revisions are not pooled. All 2,566 valid, non-optimal answers have complete better graphs, with saved independent arithmetic checks covering functional graphs and alternatives.

E.2 Thinking Off Results and High–Off Comparison

Table 17 reports off results. Qwen's conditional optimality is undefined because it produces no valid plan. Gemini has no included off condition; seven original-off and three search-off Terra answers are unavailable and excluded from denominators.

On 2,393 matched original-high/off pairs, functional successes are 1,522 versus 402 and optimal plans 752 versus 22; among 315 both-valid pairs, optimal counts are 172 versus 15. High and off denote the requested reasoning settings, and comparisons pair independently sampled answers to the same task under the recorded serving configurations.

Table 17: Thinking Off results. N counts included answers; F and O are percentages. Average is the unweighted mean of model rates.

	Original prompt			Search prompt		
Model	N	$F\uparrow$	$O\uparrow$	N	$F\uparrow$	$O\uparrow$
DeepSeek-V4-Pro	600	9.83	0.83	600	11.33	0.17
GLM-5.2	600	5.50	0.00	600	9.17	0.67
GPT-5.6 Terra	593	**52.28**	**2.87**	597	**53.10**	**2.85**
Qwen3.5-Flash	600	0.00	0.00	600	0.00	0.00
Average	—	16.90	0.93	—	18.40	0.92

E.3 Same-Task Transitions

Tables 18 and 19 retain matched model–task IDs. Digits denote original/search verdicts: 00 fails both, 01 improves, 10 regresses, and 11 succeeds both. The paired change is $\Delta = 100(n_{01}-n_{10})/M$ percentage points; its denominator includes only tasks with both answers.

Table 18: Thinking High paired transitions. f_i and o_i are functional and optimal outcomes; M counts matched tasks. Average averages model counts and paired changes.

Model	Outcome	M	00	01	10	11	$\Delta\uparrow$ (pp)
GLM-5.2	f_i	599	10	63	47	479	+2.67
	o_i	599	241	**132**	49	177	**+13.86**
GPT-5.6 Terra	f_i	582	155	68	**137**	222	−11.86
	o_i	582	382	44	**98**	58	−9.28
Qwen3.5-Flash	f_i	578	**494**	32	26	26	+1.04
	o_i	578	**563**	7	5	3	+0.35
Gemini-3.1-Pro	f_i	600	104	**89**	54	353	**+5.83**
	o_i	600	294	86	68	152	+3.00
DeepSeek-V4-Pro	f_i	556	0	26	24	**506**	+0.36
	o_i	556	110	102	49	**295**	+9.53
Average	f_i	—	152.60	55.60	57.60	317.20	−0.39
	o_i	—	318.00	74.20	53.80	137.00	+3.49

Table 19: Thinking Off paired transitions, using the same column definitions and averaging as Table 18.

Model	Outcome	M	00	01	10	11	$\Delta\uparrow$ (pp)
GLM-5.2	f_i	600	523	44	22	11	**+3.67**
	o_i	600	596	4	0	0	**+0.67**
GPT-5.6 Terra	f_i	593	196	**87**	**82**	**228**	+0.84
	o_i	593	571	**5**	**5**	**12**	0.00
Qwen3.5-Flash	f_i	600	**600**	0	0	0	0.00
	o_i	600	**600**	0	0	0	0.00
DeepSeek-V4-Pro	f_i	600	492	49	40	19	+1.50
	o_i	600	594	1	**5**	0	−0.67
Average	f_i	—	452.75	45.00	36.00	64.50	+1.50
	o_i	—	590.25	2.50	2.50	3.00	0.00

Under high, functional success gains 278 pairs and loses 288; optimality gains 371 and loses 269. Under off, matched functional counts are 402 versus 438, not the full search-off count of 440. Both prompts have 22 optimal plans: 12 shared and ten unique to each.

Table 20: Thinking High joint-success decomposition. Both-valid pairs are valid under both prompts; remaining net change involves a functional transition. Average is the mean model change.

	Both-valid	Net joint-success change		
Model	pairs	Both-valid ↑	Remaining ↑	Total ↑
DeepSeek-V4-Pro	506	+51	+2	+53
GLM-5.2	479	**+75**	+8	**+83**
Gemini-3.1-Pro	353	−3	**+21**	+18
GPT-5.6 Terra	222	−25	−29	−54
Qwen3.5-Flash	26	−1	+3	+2
Total	1,586	+97	+5	+102
Average	—	+19.40	+1.00	+20.40

Table 20 decomposes the observed outcome changes: 257 both-valid pairs gain optimality and 160 lose it. Under off, 172 of 180 newly valid plans remain non-optimal. Better alternatives first improve priorities one/two/three in 366/14/0 original-off and 397/20/1 search-off cases; 763 of 798 improve the leading priority.

Within the same-serving-configuration subsets under Thinking High, GLM gains 16.13 points in O over 124 pairs and DeepSeek gains 6.94 over 144 pairs. Terra's 205 pairs give $\Delta O = -1.95$ points versus -9.28 overall; 212 off DeepSeek pairs yield six fewer functional successes versus nine more overall.

A 20,000-replicate shared-world bootstrap gives 95% intervals of $[+1.94, +5.07]$ points for high ΔO and $[-1.98, +1.26]$ for ΔF; off intervals are $[-0.37, +0.41]$ and $[-0.04, +3.05]$. The high pooled change is $+3.50$ points versus a model-equal $+3.49$. The prompt was developed and evaluated on this benchmark.

E.4 Exact System Instruction

> Approach this as a graph-planning problem. Use the operations, initial conditions, constraints, and objectives provided in the task. Plan without executing operations or calling auxiliary tools. Apply the following methods internally and return the requested answer.
>
> 1. Represent the search problem. Describe each operation by its prerequisites, outputs, and effects. Connect operations through actual input-output support, not name similarity. Distinguish prerequisites that must hold together (AND) from alternative producers (OR), following the task's rules. Track all goals: a path to one goal is not yet a solution to the full problem.
>
> 2. Search systematically. Maintain a partial graph and its unresolved requirements. Select an unresolved requirement, consider alternative producers, and expand the prerequisites of a chosen producer. Check these choices against what is reachable from the initial conditions. Keep competing choices available. When a branch cannot satisfy its dependencies or constraints, backtrack to the responsible choice and try another branch. Reuse subgraphs where valid. Explore different operation combinations and decompositions, not merely different orderings of the same graph. Keep a complete feasible candidate while exploring coordinated replacements; do not discard a route just because one step looks unattractive.
>
> 3. Check dependencies. For each selected operation, determine what supplies each required input, whether the source guarantees the needed properties, and whether it is available when used. Check related prerequisites together. Verify that no required dependency is unresolved and that the graph satisfies the task's execution-order rules. Check whether combining individually valid branches introduces conflicts. After a replacement, recheck affected downstream operations and all goals. Do not add ordering dependencies between independent branches without a reason.

4. Make the search QoS-aware. When quality of service (QoS) matters, distinguish hard constraints from optimization preferences. Use the task's metrics, aggregation rules, and comparison precision. Track how graph structure, shared work, and parallelism affect end-to-end quality. Let QoS guide which alternatives to explore, but do not treat an unfinished candidate's local score as its complete-plan score. Compare complete feasible graphs under the stated preference rule; fewer calls or individually better operations need not yield a better graph. Challenge the first feasible solution with alternative subgraphs, recompute the full effect of changes, and retain the best valid candidate found. A limited search does not prove global optimality.

Before submitting, check the selected graph against the goals, constraints, and required answer format.

E.5 Graph-Search Tool and Paired Protocol

Stage 1 uses a text-JSON interface under Thinking High: `initial`, `goals`, and `actions` define a finite search problem. Immutable attributes distinguish objects, versions, guarantees, and exact record sets; action rules specify named ports, matching conditions, and output properties. Measures support sum, product, and critical-path aggregation with ordered minimization/maximization. The program searches DAGs over a fixed, finite set of action instances with immutable objects, counting shared calls once.

Candidates report calls, input/goal sources, and measures within the submitted model. Final generation must restore obligations omitted from the model and recompute whole-graph QoS. No-solution refers to the submitted model, and incomplete search reports its budget status separately. The tool uses the submitted model alone, without access to benchmark answers or the original evaluator.

Limits are three candidates, 200,000 search steps, 50,000 grounded actions, and 5,000 milliseconds; 12 feedback cases retain smaller recorded limits. Deterministic compatibility handling normalizes syntax without adding obligations. Initial/compatibility processing yields candidates for 1,990 inputs; the remaining 1,010 selectively receive one error-feedback and parameter-revision turn, restoring candidates for 350 (Table 21).

Table 21: Tool-input coverage over 600 tasks per model. Feedback counts inputs receiving one parameter-revision turn. Candidate rate measures search availability, not final-task correctness. Average weights models equally.

Model	Feedback inputs	Inputs with candidates	Candidate rate ↑ (%)
GLM-5.2	91	579	96.50
GPT-5.6 Terra	132	565	94.17
Qwen3.5-Flash	599	3	0.50
Gemini-3.1-Pro	150	**597**	**99.50**
DeepSeek-V4-Pro	38	596	99.33
Total	1,010	2,340	78.00
Average	202.00	468.00	78.00

Stage-2 continuations share the task, Stage-1 response, and applicable feedback history. R receives the actual search response; E receives `result_withheld`, without candidates or status. Both independently generate final graphs at temperature 1, the same reasoning setting, and no application-level output-token cap. E is a continuation baseline with the shared modeling history, distinct from the single-turn prompt-study setup.

Of 6,000 planned final-answer cells, 5,997 are complete and non-empty: 2,998 R and 2,999 E, yielding 2,997 pairs. Selection by completeness retains 5,912 primary and 85 recovery responses. Three Terra cells remain missing. The analysis uses recovered complete answers; Stage-1 and feedback outputs supply modeling context, while Stage-2 final graphs are scored. This paired study is separate from the 10,705-answer prompt study.

Stage-2 ledgers record 6,104 HTTP-entered attempts, including 13 rate-limited attempts, excluding Stage 1, feedback, and local search. R adds a mean 8,569.7 serialized message bytes (median 6,963), measuring additional input size. Initial dispatch is R-first/E-first/undetermined for 1,977/1,011/9 pairs; all 1,009 complete feedback pairs are R-first. The shared-history comparison follows this recorded dispatch and feedback protocol.

E.6 Tool-Response Outcomes and Diagnostics

Table 22 separates protocol strata. Pooled E/R rates are $F = 61.56/59.26\%$, $O = 30.83/29.66\%$, and $O \mid F = 50.08/50.06\%$; the main table uses model-equal means. Across 20,000 shared-world bootstrap replicates, 95% intervals are $[-3.85, -0.74]$ points for ΔF and $[-2.79, +0.48]$ for ΔO, which includes zero.

Table 22: Tool-response outcomes by protocol. Counts use complete pairs; each arrow runs from E to R. Average averages the two observed strata, not the five models.

Protocol	M	n_F E→R	$\Delta F \uparrow$ (pp)	n_O E→R	$\Delta O \uparrow$ (pp)
Initial / compatibility	1,988	$1,521 \to 1,467$	-2.72	$776 \to 741$	-1.76
One feedback turn	1,009	$324 \to 309$	$\mathbf{-1.49}$	$148 \to 148$	$\mathbf{0.00}$
Total	2,997	$1,845 \to 1,776$	-2.30	$924 \to 889$	-1.17
Average	—	$922.5 \to 888.0$	-2.10	$462.0 \to 444.5$	-0.88

Functional success improves/regresses in 260/329 pairs; optimality in 272/307. Optimality gains comprise 89 functional-failure-to-optimal and 183 valid-non-optimal-to-optimal transitions; losses comprise 127 optimal-to-functional-failure and 180 optimal-to-valid-non-optimal transitions. Functional transitions therefore contribute -38, and both-valid transitions $+3$. These transitions compare independently generated final graphs from the shared modeling history.

Exact priority-ordered QoS comparison on 1,516 both-valid pairs favors R/E/ties in 388/386/742 cases; among 536 both-non-optimal pairs, counts are 205/206/125. These counts summarize the direction of priority-ordered differences. Among the 2,337 complete candidate-available pairs, n_F changes $1,792 \to 1,726$ and n_O $904 \to 868$.

In the sequential Struct/Call/Dep/Goal/QoS checks, first failures number 498/362/105/187/921 for E and 626/276/105/214/887 for R. Qwen obtains candidates on three tasks; 400/600 R answers still contain search parameters, versus 204 for E. Excluding Qwen leaves 2,397 pairs with $n_F = 1,841 \to 1,776$ and $n_O = 924 \to 889$: the decline also occurs beyond Qwen's output-role confusion.

E.7 Tool-Assisted Planning Cases

The following cases link submitted graphs to their search records. For GLM's `Q600V6-2-133-WR6`, both plans are valid. The candidate exposes a raw read shared across archive, public, and internal outputs, replacing E's separate public export. Alongside another allocation change, R reduces cost $35 \to 33$ at latency 14 and reaches the cost-then-latency optimum.

For DeepSeek's `Q600V6-1-041-SV6-02`, the internal output must contain exactly p0/p1/p2, but direct export includes p3. The search goal encodes kind, scope, and representation without this exact set. A candidate satisfies the weaker goal; R omits the required filter and fails Goal, whereas E filters and is optimal. Formalization loses an obligation that integration fails to restore.

For DeepSeek's `Q600V6-1-043`, R uses upload port `input` instead of `file` and fails Call; E uses the correct port. Candidates contain no upload node, locating the error in final integration. For Qwen's `Q600V6-1-024`, a no-solution result for the submitted model is followed by another search request in R (Struct failure), while E constructs a valid graph. Candidate scope and interaction stage both matter.

F Additional Discussion and Limitations

AgentQoSBench studies static planning under explicit contracts and modeled service properties. Latency assumes sufficient parallel capacity, and reliability assumes independent call successes. Resource contention, retries, and observation-dependent replanning are natural extensions toward online execution. Exact and lexicographic objectives provide a clear measure of optimization ability; budget-based or satisfactory-service objectives would connect this measure to a wider range of user preferences.

The benchmark combines reusable mechanisms in full tool environments, jointly testing candidate discovery, dependency planning, and service optimization. Further studies can vary mechanism families and catalog layout, given known sensitivity to information position (Liu et al., 2024), and track robustness across model configurations and repeated runs. Extending prompt-guided search to unfamiliar tool environments and selective candidate integration can further develop its practical value.