COPA: COMPILING PLANS AND ANSWERS FOR EFFICIENT TOOL AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool agents often solve complex tasks by interleaving model calls with tool use, but repeatedly processing intermediate results and growing interaction histories incurs substantial token costs. This raises an intriguing question: *Can we avoid this repeated interaction by generating everything needed to complete the task in a single model call?* Simply requesting a complete plan or answer, however, is not enough: interface mismatches can prevent plan execution, while useful candidates generated along the way may be omitted from the final answer. To close these gaps, we introduce **COPA (COmpiling Plans and Answers)**, which guides models to produce tool plans or complete candidate answers for execution and validation without further model calls, with optional model feedback for task review or revision. At the core of COPA are a compiler and an answer processor: the former repairs resolvable interface mismatches, enabling the runtime to execute plans from one planning call; the latter recovers omitted answers by validating and selecting complete candidates already in model responses. On DeepPlanning's original Shopping tasks, shared-tool comparisons across Qwen and Gemma show competitive task success with 94.3%–97.2% fewer total model tokens than ReAct and CodeAct. On AppWorld public train, COPA completes 78/90 tasks with 81.6% fewer tokens than Simplified ReAct. In constructed shopping stress tests, COPA's answer processor increases Qwen's correct returns from 4/96 to 31/96 without additional model calls. Code will be made publicly available.

1 INTRODUCTION

Tool agents use external tools to complete tasks such as shopping, travel planning, and managing information across apps. A shopping agent must find suitable products, check delivery requirements, apply discounts, and return a valid basket. In a typical interactive workflow, the model chooses a tool call, reads the result, and decides what to do next (Yao et al., 2023). Each new turn sends intermediate results and an expanding history back to the model. **However, a dependency between tool calls does not necessarily require another model decision.** When the next operation is already specified, the runtime can pass a tool result directly to it, rather than ask the model to read and forward it.

Generating a complete plan in one call can reduce token consumption, but it does not guarantee successful execution or answer delivery. CodeAct composes tool calls in executable code (Wang et al., 2024); LLMCompiler plans and schedules dependent calls (Kim et al., 2024); and PIVOT uses execution feedback to refine trajectories (Zhang et al., 2026a). Execution feedback helps agents adapt, but each model-mediated revision consumes additional tokens. Some obstacles instead concern how an existing proposal is represented, executed, or returned, rather than a need for further planning or search. **When a proposal already specifies the required work, can execution and validation complete the task without further model generation?**

We therefore introduce **COPA (COmpiling Plans and Answers)**, which guides models to propose tool plans or complete candidate answers and combines a plan compiler, an execution runtime, and an answer processor to turn them into task results. To elicit executable plans, we provide tool definitions, input/output contracts, and reference rules, and ask the model to specify all operations, arguments, dependencies, and the final output in one structured response. Both paths turn model proposals into task results: plans specify operations to execute, while complete candidates specify answers to validate and return (Figures 1 and 2).

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.

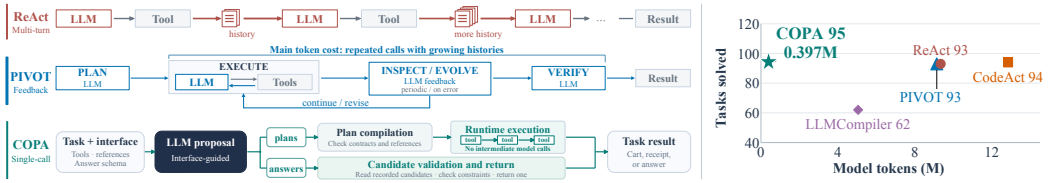

Figure 1: **Interaction pattern and shared-tool efficiency. Left:** ReAct interleaves model and tool calls; autonomous PIVOT adds feedback and verification. COPA compiles and executes plans or validates candidate answers. Single-call refers to proposal generation; optional review and repair calls are counted separately. **Right:** Qwen3.8-27B on 120 Shopping tasks with shared tools (Table 1A). LLMCompiler and PIVOT use known token lower bounds.

When a model generates a complete plan in one response, **interface mismatches can prevent execution, but correcting them need not require model feedback.** For example, a delivery filter may reference a retrieval step's product IDs even though the plan schedules both steps together. The reference already specifies that retrieval must precede filtering. COPA uses declared tool contracts and these data-flow relationships to resolve references, infer missing dependencies, and rebuild execution groups without regenerating the plan. In Shopping, this connects retrieval, delivery filtering, coupon-aware selection, and checkout: the runtime transfers intermediate results, while the tools search product combinations and compute discounts.

The answer processor addresses another problem: **an empty final answer does not necessarily mean that no usable candidate was generated.** In saved Shopping responses, the reasoning already lists a product for every requested slot, yet the final-answer field contains nothing. This also occurs in responses that end without reaching the token limit, not only in truncated generations. COPA reads these complete candidates as JSON or product-ID lists, checks public constraints, and returns the feasible candidate with the lowest public cost, without another model call.

On the original DeepPlanning Shopping tasks, COPA, ReAct, and CodeAct with Qwen3.8-27B and shared tools solve 95, 93, and 94 of 120 tasks. **COPA uses 95.8% fewer tokens than ReAct and 96.9% fewer than CodeAct** (Figure 1). A single-call complete-program CodeAct control solves 64/120 with the same tools and per-request cap (Appendix B.2.1). Gemma and Qwen3-235B also show substantial token savings (Table 1A). On 120 newly constructed Shopping instances, COPA solves 116 with Qwen3.8-27B and 115 with Qwen3-235B (Table 1B). In Travel, a fixed scheduler turns one proposal per task into itineraries, solving 83/120 tasks with Qwen3.8-27B; we also evaluate Gemma and Qwen3-235B (Table 2B). On AppWorld public train, COPA solves 78/90 tasks using 81.6% fewer tokens than Simplified ReAct, including model review and repair calls.

We also conduct controlled experiments to test the benefits of the compiler and answer processor. Each comparison uses the same saved responses, so generation does not change. Contract repair makes 40 additional plans executable across four model configurations without new model calls. On constructed Shopping stress tasks, candidate processing raises Qwen's correct returns from 4/96 to 31/96 under candidate-recording instructions at a 16k token cap. A separate 768-call study reproduces processing gains across four prompt conditions on new stress tasks.

To put these processing gains in context, we examine what additional generation can recover. Without candidate return, we admit more products by relaxing delivery limits, holding product records and the optimal basket fixed. GPT's correct returns fall from 12/12 to 5/12 at 16k, then reach 12/12 in separate 32k calls on the expanded inputs. Matched Qwen and GLM studies extend the analysis.

Together, these results show that **COPA reduces repeated model interaction by compiling and executing model-proposed plans and recovers correct answers by validating existing candidates.**

2 Related Work

Compilation and repair. CodeAct composes actions in executable Python (Wang et al., 2024); LLMCompiler plans dependencies and dispatches tool calls for execution (Kim et al., 2024). Related systems compile, optimize, and repair workflows (Wei et al., 2026; Li et al., 2026b; Zhang et al., 2026a; Harikumar, 2026; Trooskens et al., 2026; El Yadouni & Li, 2026; Chen et al., 2026b). JSONSchemaBench separates schema conformance from task quality (Geng et al., 2025). COPA

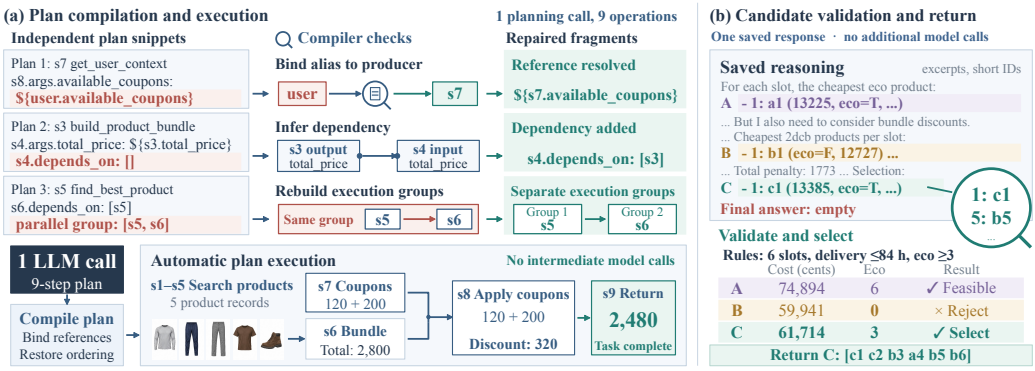

Figure 2: **COPA turns model proposals into task results through two alternative paths.** (a) Independent plan snippets illustrate alias binding, dependency inference, and execution regrouping. The nine-operation schematic shows product search, bundling, and coupon application without intermediate model calls. (b) From a saved response with an empty final answer, public checks reject B and select C, the lowest-cost feasible recorded candidate, preserving its slot–product mapping.

couples contract-aware plan execution with validation and selection of complete generated answers. Its frozen-response studies distinguish gains from these processing steps from changes in generation.

Evaluation and controlled intervention. Stage-wise studies localize failures (Chen et al., 2024; Ma et al., 2024; Zhang et al., 2025; Chen et al., 2026a; Zhu et al., 2026; Yehudai et al., 2026). Causal Agent Replay modifies stochastic continuations (Shah, 2026). Our processing tests hold responses fixed. ConstraintBench, NLCO, ComboShoppingBench, and EComAgentBench test constrained decisions (Tso et al., 2026; Jiang et al., 2026; Li et al., 2026a; Du et al., 2026). Our budget study contrasts equal cutoff changes that expand or preserve eligibility, keeping the unique optimum fixed.

Budget, reliability, and return. Test-time scaling, budget-aware prompting, and interruption studies examine generation cost and answer quality (Snell et al., 2025; Han et al., 2025; Sun et al., 2025; Wu et al., 2026; Gema et al., 2025; Srivastava et al., 2025). Reliability tests examine repeated execution and semantics-preserving transformations (Gupta, 2026; Feng et al., 2025; Honarvar et al., 2026; Mahmud et al., 2026). Our candidate-return studies process finished responses with a fixed answer processor, without further generation.

3 COPA: Compiling Plans and Answers

COPA combines a plan compiler, an execution runtime, and an answer processor to turn model proposals into task results (Figure 1, left). The model receives a task and a proposal interface. The compiler checks tool plans and repairs resolvable mismatches before runtime execution; the answer processor validates complete candidates and selects one to return. Task-specific tools and validators handle execution and checking; a controller manages model calls and coordinates these components.

3.1 Interface-Guided Planning

For tool plans, the model receives tool definitions, required arguments, declared output fields, and reference rules. We ask it to specify the operations, arguments, dependencies, and final output in one structured response. A reference such as `${s3.product_ids}` directs a later operation to use the product IDs returned by step `s3`. **The model specifies how results flow between operations, not the intermediate values themselves.** The runtime can therefore supply those values when they become available, without another planning turn.

Domain operations let the model specify the computation to perform. The four-operation Shopping interface provides product retrieval, delivery filtering, coupon-aware selection, and checkout. The model supplies item constraints, delivery limits, budget, and coupon use as arguments. In particular, `choose` searches product combinations while accounting for coupon discounts, so the model need not enumerate baskets and coupon combinations itself.

Travel uses the same division of work with structured scheduling inputs. One response supplies trip details and proposed transport, accommodation, and activities. Domain routines bind these choices to public records and construct an itinerary using opening hours, travel times, and departure constraints. Public-data fallbacks may supplement or revise the proposed choices (Appendix A.2).

For answer-generation tasks, the model proposes complete assignments, such as one product for every requested Shopping slot. The task prompt can already elicit these choices; an optional recording instruction asks the model to preserve them explicitly. These candidates specify proposed answers for validation, rather than operations for execution.

3.2 Plan Compilation and Execution

The compiler prepares a plan for execution by checking its tool contracts and data dependencies. It validates tool names and arguments and binds unambiguous references to producing steps and their declared output fields. Reference-implied dependencies restore missing edges and separate dependent steps in execution groups. Schema-guided binding also removes a singleton-reference wrapper only when the whole argument becomes type-compatible. On 120 saved GLM Shopping plans, this rule raises official task success from 72 to 80 without further model calls (Appendix C.5). These repairs preserve the proposed operations; Appendix A.1 details the rules.

After contract validation, the runtime executes each step once its dependencies have completed successfully. It binds references to actual returned values, stores outputs for downstream steps, and returns the plan's designated output. An AppWorld plan can thus connect a document reader to an amount extractor and then to a payment-request operation. **The runtime relays results and advances the plan without model turns between operations.**

The execution path provides two complementary benefits: the runtime reduces model turns spent relaying intermediate results, while contract repair recovers proposals blocked by interface mismatches. Shared-tool comparisons evaluate success and model cost for the complete path; fixed-response controls measure repair gains using the same proposals and tools.

3.3 Candidate Validation and Return

The answer processor receives a model response and public task data. It reads complete candidates from the final answer and available reasoning using JSON and slot/product-list readers, then normalizes and deduplicates them while preserving their assignments. In Shopping, validation checks visible product identities, slot coverage, product uniqueness, category requirements, delivery limits, and eco constraints, then computes the discounted price. An existing final answer that passes these checks remains eligible. **Reading identifies available candidates; validation checks their feasibility.**

Among feasible candidates, the processor selects the one with the lowest public task cost. Let $\mathcal{C}_{\mathrm{pub}}(r)$ be the extracted, normalized, and deduplicated candidates from response r that pass validation and have a defined public cost. When this set is nonempty, we select

$$c^* = \underset{c \in \mathcal{C}_{\mathrm{pub}}(r)}{\arg\min}\ \mathrm{cost}_{\mathrm{pub}}(c). \tag{1}$$

Cost is recomputed from public task data and measures the task objective, such as basket price, not token usage. In Figure 2b, the eco constraint rejects B; C is then selected over A because it has a lower price. **The return rule uses feasibility and objective value, not just candidate order.**

Ties are broken deterministically, and the selected candidate is returned in the required schema. If a processing stage cannot provide a qualifying replacement, it keeps the current answer: the JSON stage retains the raw final, and the list extension retains the JSON result (Appendix D.1). Neither stage combines partial candidates or constructs new assignments.

3.4 Execution and Feedback

The controller can run the initial proposal directly or retain bounded model feedback: semantic review before execution and task-local repair after validation or execution feedback. Revised plans pass through the compiler before execution. In AppWorld, repair can update task-local code while

preserving declared interfaces and the shared tool library. Model-call and execution limits bound feedback. **Contract repair and candidate return add no model calls.** Model review and revisions to plans or task-local code do require additional calls, which are included in reported controller usage. A separate evaluator scores task correctness; candidate selection does not use reference answers.

4 Experimental Design

4.1 Tasks and Metrics

We evaluate DeepPlanning Shopping and Travel (Zhang et al., 2026b) and AppWorld (Trivedi et al., 2024). The original Shopping systems and the domain-specific Travel implementation were developed on their respective evaluation task rosters; the AppWorld controller and tool library were developed on public train. Shopping compares COPA, ReAct, and CodeAct with Qwen3.8-27B, Gemma, and Qwen3-235B on the original 120 tasks and 120 constructed instances with independently verified unique optima. Each comparison shares four tools and model settings; Qwen3.8-27B comparisons also include adapted LLMCompiler and an autonomous PIVOT reproduction (Zhang et al., 2026a) (Table 1A/B). COPA uses one planning request per task; baselines allow up to 32. A separate GPT-5.4 plain comparison uses six shared operations (Appendix B.4). Travel evaluates Qwen3.8-27B, Gemma, and Qwen3-235B with identical public-input prompts and a fixed scheduler. On AppWorld, COPA and official Simplified ReAct both use GPT-5.4 with reasoning disabled; COPA's reported usage includes calls and tokens for bounded review and task-local repair.

Task success follows the official evaluators. All assigned tasks count toward success rates. Resource totals sum recorded input and completion tokens for the trajectories used to score all assigned tasks, including unsuccessful tasks. Unless otherwise specified, Qwen denotes Qwen3.8-27B in low-thinking mode and GLM denotes GLM-5.3-Flash. Task construction, full model names, controller limits, and scoring protocols are detailed in Appendices A and B.

4.2 Frozen-Response Interventions

Plan repair. We compare Raw and COPA processing on saved responses for 120 original Shopping tasks under four configurations, giving 480 assignments. Raw executes the submitted plans; COPA (contract) normalizes plan-confidence percentages and repairs reference conventions. After execution, full COPA can also refine product selections and recompute costs using public task data. COMPLETE means the plan executes and produces its designated output, not that the task is solved correctly. Processing-stage controls and paired scoring are specified in Appendix C.

Candidate return. Two candidate-return studies evaluate Qwen and GLM on separate sets of 48 newly constructed six-slot Shopping stress tasks (blocks), distinct from the 120 constructed instances used for system evaluation. Each task requires one product per slot under delivery and eco constraints; EXACT denotes the unique minimum-cost basket. Candidate readers are fixed before collecting responses. The first study crosses original and candidate-recording prompts with 16k and 32k budgets, comparing two minimum-cost processors on the same saved responses: JSON reads JSON candidates; JSON+list also reads product-ID lists. The second independently varies candidate recording and a clarification that the JSON requirement applies only to the final answer, forming four prompt conditions at 32k. Each block is evaluated twice per condition, yielding 96 responses per model/prompt/budget cell and 768 calls per study. Only eligibility-expanded inputs are evaluated. Raw scores the unprocessed final answer. Prompt definitions, selection controls, and statistical analyses are given in Appendices D.1 and D.2.

4.3 Eligibility and Budget Interventions

Separate direct-generation tests relax delivery cutoffs to admit more products while keeping product evidence and the optimal basket fixed. GPT-5.4-high evaluates 12 blocks with two, four, or six eligible products per slot at 16k, and the same inputs with six per slot at 32k. We evaluate Qwen and GLM on each of two 24-block task sets at delivery cutoffs of 36, 60, and 84 hours and 16k/32k budgets, with three requested-seed repeats per condition. The two 24-hour cutoff changes have different roles: one admits additional products, while the other leaves the eligible set unchanged and serves as the matched control. Budgets use separate requests, without candidate-return processing.

Table 1: **Shopping systems under shared tools.** 120 tasks per set. Calls: recorded model requests; tokens: scored-trajectory input plus completion tokens (millions). $\geq$: known token lower bound.

		A. Original Shopping				B. Constructed Shopping			
Model	System	Solved	Calls	Tokens	Tool calls	Solved	Calls	Tokens	Tool calls
Qwen3.8-27B	COPA	95	**120**	**0.397**	1,009	116	**120**	**0.374**	2,590
	ReAct	93	1,116	9.337	964	119	1,053	7.549	899
	CodeAct	94	1,232	12.840	1,003	116	1,237	12.291	889
	LLMCompiler	62	491	$\geq$5.047	935	98	238	$\geq$1.504	1,074
	PIVOT	93	910	$\geq$9.136	1,079	119	858	6.490	935
Gemma-4-31B-it	COPA	90	**120**	**0.250**	1,008	77	**120**	**0.263**	1,688
	ReAct	89	1,079	9.023	948	109	1,009	7.278	883
	CodeAct	84	499	4.362	968	83	537	3.881	1,137
Qwen3-235B	COPA	81	**120**	**0.870**	872	115	**120**	**0.787**	1,978
	ReAct	74	1,240	11.138	1,022	117	1,371	10.273	1,076
	CodeAct	80	416	4.273	2,028	120	268	2.175	1,371

A further study shifts all times and cutoffs by four hours while preserving the decision problem. Appendix E describes task construction and call schedules, defines the matched budget contrasts, and specifies the requested-seed and time-shift checks.

5 Experimental Results

5.1 Cross-Task Capability and Efficiency

COPA solves tasks with far fewer model turns under matched tools. With Qwen3.8-27B and four shared Shopping operations, COPA, ReAct, and CodeAct solve 95, 93, and 94 tasks (Table 1A). One planning request per task reduces total model tokens by 95.8% and 96.9% relative to ReAct and CodeAct. Against autonomous PIVOT, COPA solves 95 versus 93 tasks with at least 95.7% fewer tokens. Mean task time falls by factors of 1.53 and 1.77 relative to ReAct and CodeAct, respectively (Appendix B.1). Tool-call totals remain close; runtime execution avoids repeatedly sending intermediate results and growing histories to the model. A single-call complete-program CodeAct control solves 64/120. All 120 programs parse; among the 37 tasks solved only by COPA, 21 control programs misuse argument or return types (Appendix B.2.1).

The efficiency pattern extends to other backbones and a different tool library. With Gemma on the original roster, COPA solves 90 tasks versus ReAct's 89 and CodeAct's 84, with 97.2% and 94.3% fewer tokens, respectively. A separate Qwen3.8-27B run shows similar results (Appendix B.3). With Qwen3-235B, COPA, ReAct, and CodeAct solve 81, 74, and 80 tasks, respectively (Table 1A). COPA uses 0.870M tokens versus 11.138M for ReAct and 4.273M for CodeAct, reducing token use by 92.2% and 79.6%. Across these three backbones, COPA also completes more original Shopping tasks than either baseline. The six-operation GPT-5.4 plain comparison extends this result to a different tool library: COPA uses 120 calls and 0.279M tokens, versus ReAct's 989 calls and 3.755M tokens. Official rescoring gives 56/120 versus 55/120 successes, with 92.6% fewer tokens (Appendix B.4).

Token savings extend to constructed instances, with varying task success. On the constructed roster, COPA solves 116/120 tasks with Qwen3.8-27B versus ReAct's 119 and CodeAct's 116 (Table 1B), using 95.0% and 97.0% fewer tokens. PIVOT also solves 119 tasks; COPA uses 94.2% fewer tokens while solving three fewer tasks. COPA makes more tool calls (2,590 versus 899 and 889), but the runtime executes them without a new model turn for each call. With Qwen3-235B, COPA solves 115/120 tasks versus 117 for ReAct and 120 for CodeAct, using 92.3% and 63.8% fewer tokens. Gemma's original prompt yields 77/120. With clarified public schemas and execution guidance, four fixed-seed repeats each reach 120/120 on the same tasks, averaging 0.535M tokens per run with one model call per task (Appendix B.3).

The savings persist when the controller retains model feedback. On AppWorld, COPA uses 201 calls and 1.622M tokens, including review and repair: 81.6% fewer tokens than Simplified ReAct. It solves 78/90 tasks versus 85/90; both complete 25/30 scenarios (Table 2A). Reviews check task coverage; execution errors can trigger repairs to task-local tool code. Feedback thus checks and repairs proposed work, while the runtime coordinates tool calls (Appendix B).

Table 2: **Capability across task interfaces.** A: AppWorld public train; tokens in millions and total elapsed time in seconds across 90 tasks. Scenario success (official SGC) requires all three task variants to succeed. B: domain-tool capability; 8k/32k denote completion-token caps.

A. AppWorld: controller with feedback

System	Solved /90	Scenarios /30	Model calls	Tokens	Time (s)
COPA	78	25	201	1.622	986.62
Simplified ReAct	85	25	1,144	8.820	6,523.39

B. Domain-tool capability

Task	System	Model / configuration	Solved
Shopping	COPA	GPT-5.4 plain; four operations	101/120
Shopping	ReAct	GPT-5.4 high; native tools	82/120
Travel	COPA	Qwen3.8-27B; 8k	83/120
Travel	COPA	Gemma-4-31B-it; 8k	72/120
Travel	COPA	Qwen3-235B; 32k	72/120

Domain operations let one proposal specify substantially more work. With GPT-5.4 and four Shopping operations, COPA solves 101/120 tasks (84.2%), all without extra products (Table 2B). The model specifies item constraints, delivery limits, budget, and dependencies; the tools carry out filtering, coupon-aware selection, and checkout. In particular, `choose` jointly searches product and coupon combinations; the six-operation library instead ranks item candidates before optimizing coupons. Thus the 101-versus-56 difference involves selection procedures, not just tool count. Appendix A.2 gives the domain implementations and the historical Shopping ReAct reference.

The same Travel scheduler supports multiple model backbones. COPA solves 83/120 tasks with Qwen3.8-27B and 72/120 each with Gemma and Qwen3-235B under the official all-constraint evaluator (Table 2B). Each model submits trip choices in one call to the same scheduler. For Qwen3-235B, the larger-budget run solves 72 rather than 30 tasks, with no capped responses versus 70 initially and only 2.41% more actual tokens (Appendix B.6). A larger allowance need not entail proportional growth in token use.

5.2 Plan Compilation and Repair

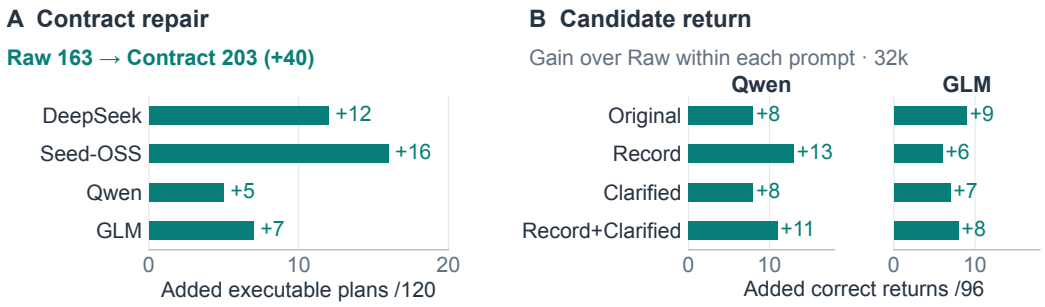

Figure 3: **Same-response processing gains.** A: contract repair. B: JSON+list over Raw. Record requests candidate recording; Clarified limits the JSON requirement to the final answer, not intermediate reasoning. Neither path adds model calls. Counts and prompts: Tables C16 and D22.

We now hold responses fixed to test plan processing independently of generation. **Some plans need a representation repair, not another planning call.** Contract repair normalizes plan-confidence percentages and resolves intermediate or final-output references. Across four configurations on 120 tasks using A1's six-operation interface, it raises executable plans from 163 to 203 and strict-reference matches from 34 to 43, without new model calls (Figure 3A). JSON extraction and schema validation alone add no executable plans. For example, binding a final-output reference from the summary tool's name to its sole calling step restores execution without changing any operations.

Direct-reference resolution is the most widely needed rule family in A1 (Appendix C). The four-operation interface provides a second test: on 120 saved GLM responses, schema-guided reference binding makes 11 additional plans executable, eight of which yield new official task successes without

Table 3: **Frozen-response answer recovery.** 48 blocks, 768 calls; Exact /96. A: list gains over JSON with recording; four-comparison-adjusted 98.75% block-bootstrap intervals. B: Qwen selection with recording; C: Qwen without recording. * denotes COPA variants; First/Last keep JSON when it is feasible and cheaper than the selected list.

A. Additional coverage from native lists

Config.	Budget	Raw	JSON*	JSON+list*	List gain (pp)	Interval (pp)
Qwen	16k	4	6	**31**	**+26.0**	[15.6, 37.5]
Qwen	32k	65	65	**80**	**+15.6**	[6.3, 27.1]
GLM	16k	0	0	0	0.0	[0.0, 0.0]
GLM	32k	5	6	7	+1.0	[0.0, 4.2]

B. Selection with the same reader

Budget	JSON*	First*	Last*	Min-cost*
16k	6	24	25	**31**
32k	65	77	78	**80**

C. Reading without recording

Budget	JSON*	JSON+list*
16k	4	**30**
32k	57	**72**

new model calls (Appendix C.5). This connects representation repair to completed Shopping tasks, beyond contract acceptance.

5.3 Candidate Validation and Return

Candidate reading and selection improve correct returns without changing generation. Under recording instructions, Qwen's correct returns at 16k rise from 4/96 for Raw to 6/96 with JSON and 31/96 with JSON+list (Table 3). Adding lists also raises 32k returns from 65/96 to 80/96; both adjusted list-over-JSON intervals exclude zero. Recovered candidates were already present in capped 16k responses or in 32k responses that ended below the cap with empty finals. The processor turns these existing choices into correct returns rather than generating new solutions.

Coverage and selection both matter. With the same list reader, minimum-cost selection improves on the first-list control from 24 to 31 correct returns at 16k and from 77 to 80 at 32k (Table 3B). Reading exposes additional candidates; selection determines which feasible candidate is returned. GLM mainly benefits from JSON, which raises 32k returns from 23 to 27 under the original prompt. Median Qwen processing takes under 3 ms at both budgets (Appendix D.1).

Recovery across prompts. The gains do not require candidate-recording instructions: under the original prompt, list reading raises Qwen's correct returns from 4 to 30 at 16k and from 57 to 72 at 32k (Table 3C). On separate new stress tasks, within-prompt JSON+list processing adds 8–13 correct Qwen answers and 6–9 GLM answers per 96 responses under all four prompt conditions, with no observed losses (Figure 3B). All 40 Qwen recoveries require list reading: 37 stop below the cap with empty finals and three hit the cap. Those 37 finals are already null in the saved HTTP payloads; all 384 Qwen responses preserve the returned content and reasoning fields through client parsing. For GLM, 25 of 30 gains come from JSON and five from lists (Appendix D.2). **Recovery extends beyond truncated responses**, including prompts with explicit final-output requirements.

5.4 Eligibility and Generation Budget

Candidate return can use only what the model generates. We therefore examine generation separately, varying eligibility and budget without candidate return. **Expanding the eligible set can reduce correct delivery even when the optimal basket stays unchanged.** At 16k, GPT is Exact on all 12 blocks with two or four eligible products per slot, but only five with six. Seven calls hit the cap; separate 32k calls on the same expanded inputs answer all seven correctly, reaching 12/12 (Figure E2). Qwen shows partial recovery in a separate twelve-block replication: all 36 requests hit the 16k cap, while separate 32k requests yield 26/36 Exact answers, with recovery under every pre-frozen requested seed (Table E30A). To compare recovery on common inputs, we run Qwen and GLM on each of two 24-block task sets (Figure 4). At 16k, both models are nearly perfect before expansion, but neither exceeds 4/72 correct returns on either task set after expansion. At 32k on the expanded inputs, Qwen exceeds two-thirds on both sets, versus under one-quarter for GLM.

The recovery gap persists under matched controls and equivalent time encodings. Eligibility-preserving cutoff changes control for the cutoff literal; requested-seed checks and a four-hour shift retain the qualitative pattern, although effect sizes vary (Appendices E.3.1 and E.5). These studies motivate COPA's division of work: additional alternatives can hinder answer delivery even when the required basket is unchanged. The plan path delegates selection to domain operations, whereas the answer path returns a complete selection already generated. Both paths can complete work without another model turn; increasing the generation budget instead allows more model computation, with model-dependent gains.

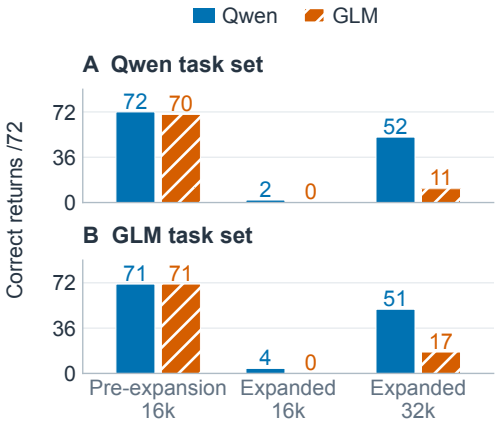

Figure 4: **Eligibility expansion and budget recovery.** 24 blocks per task set, three repeats per model/condition. Product evidence and optimum stay fixed. Budgets use separate requests without candidate-return processing.

More well-formed output does not imply the same gain in correct answers. Beyond the 3,504 assigned requests in the GPT, crossed-roster, and encoding studies, a separate sweep over 37 configurations finds a net increase of 84 schema-valid returns but only 15 additional Exact answers with more budget (Appendices E.6 and E.1). Larger budgets should therefore be evaluated by correct task completion, not just successful parsing.

6 Discussion

The interface determines how much work one model turn can delegate. Shared-tool comparisons show that lower token use need not require fewer tool calls: COPA reduces the model turns used to coordinate them (Table 1). Domain operations let one proposal request computations such as joint product–coupon search or itinerary scheduling (Table 2B). Gemma's clarified-interface repeats show how explicit schemas and execution guidance support reliable one-call delegation on these tasks. AppWorld retains bounded review and task-local repair around deterministic execution while reducing token use (Table 2A). Model feedback can thus serve review and revision rather than routine relay of intermediate outputs. Reported online costs exclude domain-tool development and maintenance.

A missing final answer need not mean a missing solution. Final-answer scores cannot by themselves distinguish failure to generate a solution from failure to return one already present. The same-response comparisons locate usable candidates in both capped responses and responses with empty finals (Table 3; Figure 3B). They also show why reading and selection matter: an available candidate must be recognized, checked, and chosen before it can become the returned answer. This suggests distinguishing a need for new candidates from a failure to deliver existing ones. The separate budget studies show that more generation can improve correct delivery, but does not resolve every failure (Figure 4). Future work could use candidate completeness and validation results to guide the choice between returning an existing answer and requesting further generation.

7 Conclusion

We introduced COPA, which guides models to propose plans and complete candidate answers for execution or validation. Across Shopping, Travel, and AppWorld, models specify tool calls and dependencies or structured task choices, while the runtime and tools handle execution and domain computation. Shared-tool comparisons with Qwen and Gemma show competitive task success with substantially fewer model tokens. Qwen-based COPA also achieves high success rates and substantial token savings on additional constructed Shopping tasks. Frozen-response experiments show that contract repair makes more plans executable and candidate processing recovers correct answers, both without additional model calls. Separate generation studies show that more eligible products can hinder correct answer delivery even when the optimal basket stays unchanged. **COPA reduces repeated model interaction by delegating linked operations to the runtime and improves correct answer delivery by validating and selecting complete candidates from model responses.**

AI USE STATEMENT

Generative AI tools assisted with literature retrieval and discovery, including identifying related work, and with manuscript writing and polishing. Human authors retain responsibility for the content, claims, code, and research artifacts.

ETHICS STATEMENT

The controlled studies use synthetic decision instances and existing benchmark artifacts; they involve no human subjects or personal data. We preserve provider and other operational failures as separate outcomes to avoid attributing unobserved behavior to a model. The main foreseeable risk is overgeneralizing benchmark- and configuration-specific engineered stress results, which we address through explicit scope limits and complete assigned-call accounting.

REPRODUCIBILITY STATEMENT

Appendices A–E describe the interfaces and implementations, model and controller configurations, scoring criteria, statistical analyses, and supplementary results.

REFERENCES

Jing Chen, Yang Sun, Li Zhang, Lin Xu, and Jie Shi. Long-horizon agent trajectory attribution: A unified benchmark and fine-grained annotation framework. *arXiv preprint arXiv:2608.06909*, 2026a. URL https://arxiv.org/abs/2608.06909.

Mengzhuo Chen, Junjie Wang, Zhe Liu, Yawen Wang, Haiming Zheng, and Qing Wang. From failed trajectories to reliable LLM agents: Diagnosing and repairing harness flaws. *arXiv preprint arXiv:2606.06324*, 2026b. URL https://arxiv.org/abs/2606.06324.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and Feng Zhao. T-eval: Evaluating the tool utilization capability of large language models step by step. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 9510–9529, 2024. doi: 10.18653/v1/2024.acl-long.515. URL https://aclanthology.org/2024.acl-long.515/.

Zeyao Du, Tong Li, and Haibo Zhang. EComAgentBench: Benchmarking shopping agents on long-horizon tasks with distributed hidden intent. *arXiv preprint arXiv:2606.17698*, 2026. URL https://arxiv.org/abs/2606.17698v3.

Salma El Yadouni and Guanyi Li. TraceCompiler: Skill-guided mining and compilation of LLM agent traces into mostly deterministic workflows. *arXiv preprint arXiv:2608.02680*, 2026. URL https://arxiv.org/abs/2608.02680.

Shiwei Feng, Xiangzhe Xu, Xuan Chen, Kaiyuan Zhang, Syed Ahmed, Zian Su, Mingwei Zheng, and Xiangyu Zhang. TAI3: Testing agent integrity in interpreting user intent. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-5503. URL https://doi.org/10.52202/085713-5503.

Aryo Pradipta Gema, Alexander Hägele, Runjin Chen, Andy Arditi, Jacob Goldman-Wetzler, Kit Fraser-Taliente, Henry Sleight, Linda Petrini, Julian Michael, Beatrice Alex, Pasquale Minervini, Yanda Chen, Joe Benton, and Ethan Perez. Inverse scaling in test-time compute. *arXiv preprint arXiv:2507.14417*, 2025. URL https://arxiv.org/abs/2507.14417.

Saibo Geng, Hudson Cooper, Michał Moskal, Samuel Jenkins, Julian Berman, Nathan Ranchin, Robert West, Eric Horvitz, and Harsha Nori. JSONSchemaBench: A rigorous benchmark of structured outputs for language models. *arXiv preprint arXiv:2501.10868*, 2025. URL https://arxiv.org/abs/2501.10868v3.

Aayush Gupta. ReliabilityBench: Evaluating LLM agent reliability under production-like stress conditions. *arXiv preprint arXiv:2601.06112*, 2026. URL https://arxiv.org/abs/2601.06112.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware LLM reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 24842–24855. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-acl.1274. URL https://aclanthology.org/2025.findings-acl.1274/.

Pranav Harikumar. PlanCompiler: A deterministic compilation architecture for structured multi-step LLM pipelines. *arXiv preprint arXiv:2604.13092*, 2026. URL https://arxiv.org/abs/2604.13092.

Shahin Honarvar, Amber Gorzynski, James Lee-Jones, Harry Coppock, Marek Rei, Joseph Ryan, and Alastair F. Donaldson. Capture the flags: Family-based evaluation of agentic LLMs via semantics-preserving transformations. *arXiv preprint arXiv:2602.05523*, 2026. URL https://arxiv.org/abs/2602.05523.

Xia Jiang, Jing Chen, Cong Zhang, Jie Gao, Chengpeng Hu, Chenhao Zhang, Yaoxin Wu, and Yingqian Zhang. Reasoning in a combinatorial and constrained world: Benchmarking LLMs on natural-language combinatorial optimization. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 30592–30648, 2026. doi: 10.18653/v1/2026.findings-acl.1529. URL https://aclanthology.org/2026.findings-acl.1529/.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. An LLM compiler for parallel function calling. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 24370–24391, 2024. URL https://proceedings.mlr.press/v235/kim24y.html.

Adrian Li, Kelong Mao, Yudong Guo, Heming Xia, Xinwei Yang, Lirui Luo, Jace Wong, Pu Yao, Sulong Xu, and Simiu Gu. ComboShoppingBench: Evaluating LLM agents for budget-constrained basket shopping with coupons. *arXiv preprint arXiv:2608.09282*, 2026a. URL https://arxiv.org/abs/2608.09282.

Ao Li, Shangpeng Yang, Fahao Chen, Tianheng Xu, Peng Li, and Zhou Su. GraphFlow: A graph-based workflow management for efficient LLM-agent serving. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026b. URL https://arxiv.org/abs/2605.22566.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. AgentBoard: An analytical evaluation board of multi-turn LLM agents. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://arxiv.org/abs/2401.13178.

Hasan Najib Mahmud, Shreya Gupta, Isha Chaudhary, Nathaniel Enis, Ravi Mangal, Gagandeep Singh, and Corina Pasareanu. A jagged frontier: Evaluating robustness of code agents to semantics-preserving transformations. *arXiv preprint arXiv:2608.18389*, 2026. URL https://arxiv.org/abs/2608.18389.

Robert G. Newcombe. Improved confidence intervals for the difference between binomial proportions based on paired data. *Statistics in Medicine*, 17(22):2635–2650, 1998. doi: 10.1002/(SICI)1097-0258(19981130)17:22<2635::AID-SIM954>3.0.CO;2-C.

Jaineet Shah. Causal agent replay: Counterfactual attribution for LLM-agent failures. *arXiv preprint arXiv:2606.08275*, 2026. doi: 10.48550/arXiv.2606.08275. URL https://arxiv.org/abs/2606.08275v1.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

Saurabh Srivastava, Janit Bidhan, Hao Yan, Abhishek Dey, Tanu Kansal, Paras Kath, Sina Mansouri, Mohit Marvania, Vamsi Shankar Simhadri, and Gaurav Singh. Batch prompting suppresses overthinking reasoning under constraint: How batch prompting suppresses overthinking in reasoning models. *arXiv preprint arXiv:2511.04108*, 2025. URL https://arxiv.org/abs/2511.04108v4.

Yi Sun, Han Wang, Jiaqiang Li, Jiacheng Liu, Xiangyu Li, Hao Wen, Yizhen Yuan, Huiwen Zheng, Yan Liang, Yuanchun Li, and Yunxin Liu. An empirical study of LLM reasoning ability under strict output length constraint. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 7652–7671. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.389. URL https://aclanthology.org/2025.emnlp-main.389/.

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*, pp. 16022–16076, 2024. doi: 10.18653/v1/2024.acl-long.850. URL https://aclanthology.org/2024.acl-long.850/.

Geert Trooskens, Aaron Karlsberg, Anmol Sharma, Lamara De Brouwer, Max Van Puyvelde, Matthew Young, John Thickstun, Gil Alterovitz, and Walter A. De Brouwer. Compiled AI: Deterministic code generation for LLM-based workflow automation. *arXiv preprint arXiv:2604.05150*, 2026. URL https://arxiv.org/abs/2604.05150.

Joseph Tso, Preston Schmittou, Quan Huynh, and Jibran Hutchins. ConstraintBench: Benchmarking LLM constraint reasoning on direct optimization. *arXiv preprint arXiv:2602.22465*, 2026. URL https://arxiv.org/abs/2602.22465.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 50208–50232, 2024. URL https://proceedings.mlr.press/v235/wang24h.html.

Lei Wei, Qi Liu, Ruiyang Huang, Xiao Peng, TT, Lanbo Lin, Chenhao Jiang, Yuanwu Xu, Tianyuan Yang, Jiayao Liu, Li Cai, Yuying Li, and Bin Wang. EvoC2F: Compiling tool orchestration for efficient and evolvable LLM agents. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=ZSGB91kMOG.

Tsung-Han Wu, Mihran Miroyan, David M. Chan, Trevor Darrell, Narges Norouzi, and Joseph E. Gonzalez. Are large reasoning models interruptible? In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2510.11713v4.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://react-lm.github.io/.

Asaf Yehudai, Lilach Eden, and Michal Shmueli-Scheuer. Agentic CLEAR: Automating multi-level evaluation of LLM agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 750–763. Association for Computational Linguistics, July 2026. doi: 10.18653/v1/2026.acl-demo.74. URL https://aclanthology.org/2026.acl-demo.74/.

Zhaoyang Yu, Jiayi Zhang, Huixue Su, Yufan Zhao, Yifan Wu, Mingyi Deng, Jinyu Xiang, Yizhang Lin, Lingxiao Tang, Yuyu Luo, Bang Liu, and Chenglin Wu. ReCode: Unify plan and action for universal granularity control. *arXiv preprint arXiv:2510.23564*, 2025. URL https://arxiv.org/abs/2510.23564.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? on automated failure attribution of LLM multi-agent systems. In *Proceedings of the 42nd International Conference on Machine Learning*, pp. 76583–76599, 2025. URL https://proceedings.mlr.press/v267/zhang25cq.html.

Tuo Zhang, Alin-Ionut Popa, Yan Xu, Rui Song, and Dimitrios Dimitriadis. PIVOT: Bridging planning and execution in LLM agents via trajectory refinement. *arXiv preprint arXiv:2605.11225*, 2026a. URL https://arxiv.org/abs/2605.11225.

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, 2026b. doi: 10.18653/v1/2026.acl-long.335. URL https://aclanthology.org/2026.acl-long.335/.

Pengyu Zhu, Lijun Li, Yaxing Lyu, Qianxin Luo, Jingyi Yang, Yi Liu, Tingfeng Hui, Xinyu Yuan, Li Sun, Sen Su, and Jing Shao. UniACE: A unified framework for evaluating LLM agentic capabilities. *arXiv preprint arXiv:2605.27898*, 2026. doi: 10.48550/arXiv.2605.27898. URL https://arxiv.org/abs/2605.27898v3.

A Interfaces and Implementations

A.1 Contract rules

Contract repair in the A1 implementation. The A1 rule-level ablation tests three deterministic repairs before validation and execution. These are the repair families evaluated in that study, rather than an exhaustive list of compiler functions:

1. **Normalize the confidence convention.** Values in $[0, 1]$ are retained; integer percentages from 2 to 100 are divided by 100. Boolean, nonfinite, and other out-of-range values are rejected.
2. **Resolve direct references.** A valid step ID is retained. A tool name or registered alias in `final_from`, dependencies, or group membership is replaced by its producer's step ID only when that producer is unique.
3. **Bind placeholders.** An argument can reference an earlier result, for example `${bundle.total_price}`. The compiler rewrites its root to the unique producer's step ID and checks that the first field in its path belongs to the producer's declared outputs. References must occupy the whole argument value, not be embedded in other text.

Unknown or ambiguous references, duplicate step IDs, and malformed placeholders are rejected rather than guessed. The normalized plan must then pass the full contract validator and execute successfully; otherwise the processor retains Raw's prediction and endpoint. These rules change representation, not the requested products or tool operations, and use no model calls. Product-selection processing is a separate stage after execution. Examples and ablations appear in Appendix C.

Schema-guided type binding. The GLM study in Appendix C.5 evaluates an additional rule over complete argument references. A singleton list containing one reference can introduce an extra nesting level when that reference already returns an array. The processor obtains the referenced type from the producer's public output schema and compares it with the consumer's declared input type. It removes a singleton-reference wrapper only when the original whole argument fails this type-compatibility check and the rewritten whole argument passes. Unknown or ambiguous types are left unchanged. Ordinary contract validation and execution follow; the rule does not alter domain-tool implementations or use task labels. This is separate from A1's three-rule ablation above.

A.2 Domain operations

From task requirements to executable operations. The model and domain tools have complementary roles. The model expresses requirements as operation arguments and dependencies, or proposes named travel choices. The runtime binds references; the tools then carry out catalog filtering, combinatorial selection, checkout, or scheduling on public data. These domain algorithms are part of COPA's implementation, distinct from the representation repairs in Appendix C.

Shopping implementation and capability. The model specifies item constraints, delivery limits, budget, coupon use, and the dependencies among four operations:

1. `find_products` applies recognized structured filters to the public catalog; a free-text-only request instead starts from search results. It returns product IDs and records for each requested item.
2. `filter_by_delivery` obtains delivery times to the user's public destination and filters the supplied IDs by the requested limit.
3. `choose` searches distinct one-product-per-item combinations. Price bounds prune the search; when coupons are enabled, owned-coupon combinations and threshold rules determine the final-price objective.
4. `checkout` adds the selected products to the cart, applies eligible owned coupons, and returns the resulting cart and totals.

Thus the model specifies the selection problem rather than enumerating every basket or coupon combination. Fallback filters return relaxation metadata, and the coupon-aware search reports

truncation if it exceeds 400,000 nodes; task success is determined from the resulting cart by the official evaluator. The GPT-5.4 plain run executes all 120 plans and achieves 101 official successes, all without extra products. The shared-tool comparison in Appendix B.1 gives every controller these same four operations, holding domain computation fixed across controllers.

The historical ReAct reference solves 82/120 Shopping tasks with GPT-5.4 high reasoning and the benchmark's native tool interface. The 101/120 COPA run uses GPT-5.4 plain and the four domain operations above. Both counts use the official evaluator, but this historical comparison does not hold tools and reasoning settings fixed. Controller-efficiency estimates instead come from the shared-tool comparisons.

The planning responses used for evaluation account for 129,963 input and 50,594 completion tokens, totaling 180,557. This capability run is separate from the shared-tool comparisons used to estimate relative cost.

Travel implementation and capability. In the open-model evaluation, the model extracts origin, destination, dates, party size, and room count from the public query and proposes outbound and return transport, a hotel, attractions, and meals from visible database candidates in the same response. The materializer derives trip duration and weekdays from these dates, binds the proposed names and route identifiers to records, incorporates requirements from the public query, and schedules activities using opening hours, travel times, meal spacing, and departure buffers. For each day it compares a greedy schedule with a beam search of width 32, prioritizing required visits and meal coverage while checking time constraints. Public-data fallbacks may supplement or revise the proposed choices. This is constructive scheduling, whereas candidate return only selects complete answers already in a response.

The same fixed materializer is used with all three open-model backbones; only public queries and database records are available during execution. Model configurations, complete-roster outcomes, and resource totals are reported in Appendix B.6.

A.3 Six-operation Shopping interface

The DeepPlanning orchestration study gives both controllers GPT-5.4 plain and the same six operations for user context, product selection, bundle construction, coupon optimization, budget evaluation, and result summarization. COPA uses one planning call followed by reference binding, contract repair, validation, and deterministic execution. ReAct chooses one tool per model turn, with at most 12 turns. COPA's optional model selection, verification, and replanning stages are disabled. The two Shopping interfaces expose different selection procedures. In the six-operation library, `find_best_product` ranks matching products and retains up to eight candidates per item; `build_product_bundle` selects a basket using uniqueness, candidate ranks, and supplied pre-discount budget bounds; `optimize_coupons` then optimizes its discounts. The four-operation `choose` routine instead evaluates coupon discounts while searching product combinations. Controller comparisons hold the corresponding library fixed.

A.4 AppWorld controller configuration

Both controllers evaluate the same ordered roster of 90 public-train tasks. Both are configured with `gpt-5.4-2026-03-05` at temperature 0 and with reasoning disabled. COPA uses the same model for planning and synthesis. The comparator is AppWorld's official Simplified ReAct code agent, not a one-tool-per-turn reimplementation of ReAct.

COPA executes through its tool facade and may repair an ephemeral tool or a task-local bank copy; it does not update the shared bank across tasks. Simplified ReAct executes generated code using the benchmark environment. The reported 201 and 1,144 calls count returned model turns, excluding provider retries that yield no response; token totals include cached prompt tokens. The scored ReAct run reports no provider failures.

Table A1: **AppWorld controller settings.** The task roster and declared model are matched; the proposal interfaces and controller limits differ. Limits are configured ceilings, not observed calls. The Simplified ReAct output-token cap was not recorded.

Setting	COPA	Simplified ReAct
Action interface	Typed plans over 69 tools: 59 bank tools, 9 generic transforms, and one task-completion operation	Executable code actions in AppWorld
Control	Planning, pre-execution review, and bounded task-local repair	Interleaved code generation and execution feedback
Per-task ceiling	10 model calls; at most 9 before execution	50 agent steps
Review / execution	One ready-plan semantic review; at most two task execution attempts	Feedback consumed by subsequent agent steps
Retry policy	One provider attempt per logical call; SDK retries disabled	Up to 30 provider attempts per logical call; 2-second retry delay
Shared evaluation	Independent task-world resets; serial execution; official evaluation after the complete task run; no task reruns using evaluator feedback.	

B System Evaluations

B.1 Shared-tool Shopping protocol

The original four-method comparison assigns all 120 public DeepPlanning Shopping tasks to each of four systems: COPA, ReAct, CodeAct, and an adapted LLMCompiler. All receive the same four operations, `find_products`, `filter_by_delivery`, `choose`, and `checkout`, with the same public product data, user context, and numeric-filter semantics. The operations can filter or select products and compute checkout results; each method can invoke the same implementation. Private scoring inputs are unavailable during execution.

All four systems use Qwen3.8-27B, temperature 1.0, top-p 0.95, top-k 20, and an 8,192-token generation cap, with thinking and native reasoning-history preservation enabled. The frozen service uses low reasoning effort and a 65,536-token context window. Each task has a preassigned base sampler seed shared across arms; later phase calls add their call index. There is one trajectory per task and method, not multiple independent generation runs.

COPA generates one plan that its runtime executes. ReAct interleaves tool actions and observations, and CodeAct generates executable code over the shared operations. The latter two controllers and LLMCompiler permit up to 32 model requests per task. All arms share ceilings of 128 domain-tool calls and 900 seconds per task; transport retries are disabled. LLMCompiler allows up to eight plan/join rounds, including the initial plan, and executes dependency-ready nodes after receiving the complete planner output. It does not stream planner tokens into execution. Prompts follow each method's interface rather than matching byte for byte.

The transport retains native assistant responses and new observations without duplicating cumulative history. Action parsers consume provider final content rather than concatenated reasoning; empty CodeAct finals receive environment feedback on the next turn. Numeric-range semantics are included in every initial request.

B.2 Outcomes and online resources

Table B2 retains all four arms and all assigned failures. The official binary endpoint matches required product IDs and coupon quantities; it reports extra products separately. All 95, 93, and 94 successes for COPA, ReAct, and CodeAct also contain no extra products. Of LLMCompiler's 62 official successes, 56 contain no extra products and 60 also have successful controller termination. Two correct carts precede a controller exception. Composite scores are the official evaluator's graded endpoint, not a second count of solved tasks.

The ReAct/COPA and CodeAct/COPA total-token ratios are 23.53 and 32.36. Mean task time is reduced by factors of 1.53 and 1.77, respectively. Request reductions are 89.25% and 90.26%. Input-

Table B2: **Full matched Shopping outcomes.** Each arm has 120 tasks. Solved uses the official task evaluator; Complete counts successful controller terminations, whether or not the task is solved.

System	Solved	Composite	Complete	Requests	Tool calls	Mean s
COPA	95	0.9326	119	120	1,009	19.96
ReAct	93	0.9265	120	1,116	964	30.52
CodeAct	94	0.9306	120	1,232	1,003	35.35
LLMCompiler	62	0.6778	97	491	935	61.91

Table B3: **Online model-token accounting.** Input plus completion equals total. Completion includes reasoning under the service's usage accounting; reasoning and cached input are not separately enumerated. One LLMCompiler HTTP 400 request has unknown usage, so its total is a known lower bound.

System	Input	Completion	Total	Unknown-usage requests
COPA	132,943	263,799	396,742	0
ReAct	8,963,660	373,231	9,336,891	0
CodeAct	12,405,152	434,879	12,840,031	0
LLMCompiler	4,589,002	458,279	$\geq$5,047,281	1

token totals explain most of the token difference: iterative controllers repeatedly process growing histories, whereas COPA does not return to the model for each operation.

Table B4 reports paired task outcomes. The 95% intervals resample 120 task identities with 10,000 bootstrap draws and describe task variation on this roster.

Table B4: **Paired official successes relative to COPA.** Gain/loss denotes COPA-only/baseline-only success. Difference and case-bootstrap intervals are in percentage points.

Baseline	Both	Gain	Loss	Neither	Difference	95% interval
ReAct	84	11	9	16	+1.67	$[-5.83, 9.17]$
CodeAct	85	10	9	16	+0.83	$[-6.67, 7.50]$
LLMCompiler	58	37	4	21	+27.50	$[18.33, 36.67]$

All assigned tasks contribute to success rates. Resource totals sum recorded input and completion tokens for the trajectories used to score all assigned tasks, including unsuccessful tasks. The Qwen3-235B extension's recovery accounting is specified below in Appendix B.3.2.

B.2.1 SINGLE-CALL COMPLETE-PROGRAM CONTROL

To test whether one-shot delegation alone recovers the system result, we run a complete-program CodeAct variant on all 120 original Shopping tasks. It receives the same four public operations and model settings as the main Qwen comparison, but is explicitly instructed to generate the entire program, including checkout, in one response. Loops, branches, exception handling, and multiple tool calls within the program are permitted. The original CodeAct final-content parser and isolated Python executor are retained; there is no later model turn or generated-program repair. The limits are one 8,192-token generation, 128 domain-tool calls, and 900 seconds per task. Each sampler seed matches the corresponding historical CodeAct first request. Scoring uses the actual final cart, including effects before an exception.

Table B5 binds the control to the main 95/94 COPA/CodeAct batch. All 120 programs parse, no response is truncated, 94 programs exit normally, and 87 exit normally after checkout. There are 58 jointly successful tasks, 37 COPA-only successes, six single-call-CodeAct-only successes, and 19 joint failures. The observed difference is 25.83 percentage points, with a task-paired 10,000-draw bootstrap interval of $[16.67, 35.00]$.

The control identifies an interface-use failure mode: 23 programs compare a returned dictionary with a scalar. The shared output schema declares product records as objects but does not expand their

Table B5: **Complete-program delegation control.** All methods use the same original 120 tasks and four operations. The complete-program variant is distinct from standard multi-turn CodeAct; all successful carts have no extra products.

System	Solved	Requests	Input tokens	Completion tokens
COPA	95	120	132,943	263,799
Single-call CodeAct	64	120	412,953	290,300
Multi-turn CodeAct	94	1,232	12,405,152	434,879

internal rating structure. One further exception follows an upstream error response; the last two error tails are unavailable under the executor's existing observation-length limit. Of the 37 COPA-only successes, 21 involve argument or return-type misuse, 14 are normally executed but incorrect carts, and two involve other or unresolved failures. The comparison therefore evaluates the frozen proposal interfaces, their public documentation, and their execution paths together.

Single-call CodeAct uses 94.52% fewer tokens than multi-turn CodeAct; COPA uses 43.58% fewer than the single-call variant. Of this latter token difference, 91.35% is input tokens, reflecting prompt and interface description length rather than fewer requests: both methods make 120. Three non-target toy requests, including one unsuccessful formatting check, consume another 11,329 tokens outside the table; total new usage is 714,582.

B.3 Additional backbones and constructed instances

Matched-model extension. Gemma-4-31B-it evaluates COPA, ReAct, and CodeAct on the original 120 Shopping tasks with the same four operations within the comparison. Gemma uses temperature 1.0, top-p 0.95, an 8,192-token generation cap, and a 65,536-token context window; no explicit top-k or thinking budget is set. Qwen3-235B denotes Qwen3-235B-A22B-Thinking-2507, which evaluates the same three methods on both rosters with the four shared operations. It uses temperature 1.0, top-p 0.95, top-k 20, an 8,192-token generation cap, and a 65,536-token context window, with thinking enabled and preserved and reasoning effort set to low. A separate Qwen3.8-27B run uses the configuration in Appendix B.1 and yields 94/93/95 successes, with 95.7%/97.0% token reductions relative to ReAct/CodeAct (Table B6). Within each extension, COPA has one planning request; ReAct and CodeAct permit up to 32 requests, with 128 tool calls and 900 seconds per task for every arm.

Constructed task roster. All three models also evaluate the same 120 additional Shopping instances: 50 level-1, 50 level-2, and 20 level-3 tasks. Each level-1 task has four required items; levels 2 and 3 have five, with an exact pre-coupon budget or a coupon-sensitive objective, respectively. Public attribute and delivery filters leave two candidates per item. An independent public-input enumerator, which does not call the tested `choose` operation, considers 16 or 32 combinations and verifies a unique optimum matching the private validation answer for all 120 tasks. This constructed task family is not an official held-out benchmark split and is distinct from the six-slot fixed-answer studies.

Within each model comparison, all methods receive the same public task inputs and four domain operations. Reference answers are used only for scoring. Each model retains its generation settings across the original and constructed rosters. The Qwen3.8-27B constructed-roster comparison also includes LLMCompiler under the shared-tool protocol in Appendix B.1. Seven assignments without a scored final state remain unsuccessful in the 120-task denominators: two Qwen CodeAct, two Qwen LLMCompiler, one Gemma ReAct, and two Gemma CodeAct. Their recorded requests and tokens remain counted. For constructed-roster LLMCompiler, 238 requests include two without recorded usage; known input and completion totals are 1,251,857 and 252,216 tokens.

Paired interpretation. Table B8 pairs methods by task identity. Post-hoc descriptive 95% intervals resample 120 tasks with replacement 10,000 times. Qwen retains high task success and low model cost on the constructed roster. Gemma's original comparison and the interface-clarification study below distinguish the initial outcome from subsequent development.

Interface clarification and repeatability with Gemma. A trace audit locates the earliest visible issue in all 43 unsuccessful constructed-roster tasks: 20 unsupported expressions or bindings, 19

Table B6: **Resources for the shared-tool extensions.** Every arm has 120 assigned tasks. Tokens are recorded input plus completion tokens. The repeat uses the original task roster; constructed instances use the additional roster.

System	Solved	Requests	Total tokens	Tool calls
Original roster: Qwen3.8-27B repeat				
COPA	94	120	399,579	1,000
ReAct	93	1,105	9,248,015	966
CodeAct	95	1,253	13,226,487	992
Original roster: Gemma				
COPA	90	120	250,032	1,008
ReAct	89	1,079	9,022,839	948
CodeAct	84	499	4,362,399	968
Constructed instances: Qwen				
COPA	116	120	374,355	2,590
ReAct	119	1,053	7,549,409	899
CodeAct	116	1,237	12,291,120	889
LLMCompiler	98	238	$\geq$1,504,073	1,074
Constructed instances: Gemma				
COPA	77	120	263,315	1,688
ReAct	109	1,009	7,278,283	883
CodeAct	83	537	3,880,858	1,137

incorrectly typed product-ID arrays in delivery filtering, three contract/reference rejections, and one parsing failure. All responses stop normally. Unsupported `concat`/`filter`/`source` objects and comma-joined strings can yield empty candidates or missing slots, not evidence of task infeasibility. The terminal partition remains 36 tool-step failures, three contract rejections, one parsing failure, and three completed but incorrect outputs; the upstream audit changes no outcomes.

A developmental follow-up on the same tasks uses paired new seeds with unchanged model, tools, and scoring. The revised prompt supplies full public input/output schemas, type-preserving reference syntax, and execution guidance for per-item candidate grouping and delivery filtering. This clarification yields 120/120 official successes versus 73/120 with the original prompt (47 gains, no losses). Each condition uses 120 requests, totaling 535,418 and 262,557 tokens, respectively; type-binding on/off leaves each condition's scores unchanged. The intervention changes both interface documentation and execution guidance, rather than just the parser.

We then freeze the clarified prompt and run four complete repeats with new preallocated per-task sampler seeds (Table B7). The model, tools, scoring, and sampling settings remain fixed; only seeds change. Each repeat solves all 120 tasks without extra products, averaging 534,549 total tokens per run. All 480 responses stop normally, and every request has recorded usage. Type-binding on/off gives identical outputs with zero rewrites; the clarified proposals do not require this binding correction. These are repeatability tests on the same 120 previously observed tasks, not 480 independent tasks or a new matched-method comparison. The original 77/109/83 comparison remains in Tables 1 and B6.

Table B7: **Gemma repeats with clarified schemas and execution guidance.** Each preallocated seed index covers the same 120 constructed tasks with one request per task. Solved uses the official evaluator; all successful carts have no extra products. Completion includes all reported output tokens.

Seed index	Solved /120	Requests	Input	Completion	Total tokens
0	120	120	396,144	138,669	534,813
1	120	120	396,144	138,310	534,454
2	120	120	396,144	138,587	534,731
3	120	120	396,144	138,052	534,196

Table B8: **Task-paired successes in the extensions.** Gain/loss denotes COPA-only/baseline-only success. Difference and descriptive intervals are percentage points. Each comparison has 120 tasks.

Baseline	Both	Gain	Loss	Neither	Difference	95% interval
Original roster: Qwen3.8-27B repeat						
ReAct	85	9	8	18	+0.83	$[-5.83, 7.50]$
CodeAct	88	6	7	19	-0.83	$[-6.67, 5.00]$
Original roster: Gemma						
ReAct	79	11	10	20	+0.83	$[-6.67, 8.33]$
CodeAct	77	13	7	23	+5.00	$[-2.50, 12.50]$
Constructed instances: Qwen						
ReAct	115	1	4	0	-2.50	$[-6.67, 0.83]$
CodeAct	113	3	3	1	0.00	$[-4.17, 4.17]$
Constructed instances: Gemma						
ReAct	70	7	39	4	-26.67	$[-36.67, -16.67]$
CodeAct	49	28	34	9	-5.00	$[-17.50, 7.50]$

B.3.1 Trajectory-refinement and recursive-code baselines

We additionally evaluate autonomous PIVOT (Zhang et al., 2026a) and ReCode (Yu et al., 2025) on both 120-task rosters with Qwen3.8-27B and the same four operations. Model settings and request, tool-call, and time limits follow Appendix B.1. All planning, execution, inspection, and revision requests contribute to the reported costs.

PIVOT follows the paper's deployed inline procedure: an initial planning-only request, inspection after every three tool-call rounds, up to three revisions triggered by tool errors or empty results, and verification before the final answer. This is a reproduction of the autonomous workflow described in the paper's Appendix A, without human feedback or private scoring information. It does not add the candidate-pool search and environment rollback from the paper's general trajectory-search algorithm.

ReCode uses the official recursive agent and code executor, with a Shopping environment bridge that exposes the shared tools through JSON commands. Recursive expansion, code-block handling, and syntax rewriting retain the upstream implementation; the bridge adds task inputs, model transport, isolated tool execution, and termination handling. The default depth, retry, and rewrite limits are 10, 5, and 5. Its results below describe this recursive-agent and Shopping-bridge combination as a supplementary evaluation.

Table B9: **Autonomous PIVOT reproduction and ReCode Shopping-bridge evaluation.** Each row covers 120 tasks. Input and completion tokens sum all scored trajectories; $\geq$ marks a known lower bound.

Task set	System	Solved	Requests	Input	Completion	Total tokens
Original	PIVOT	93	910	8,182,050	954,004	$\geq$9,136,054
	ReCode	6	733	1,932,698	615,680	$\geq$2,548,378
Constructed	PIVOT	119	858	5,758,703	731,048	6,489,751
	ReCode	8	860	2,213,743	577,475	2,791,218

Each original-roster baseline has one failed request without recorded usage; every returned response and every constructed-roster request has usage. These failed requests remain in the request counts, and their unknown tokens are not imputed. PIVOT has 1,079 and 935 tool calls on the original and constructed rosters, respectively; ReCode has 405 and 473. Of PIVOT's 93 and 119 official successes, 91 and 117 have no extra products. All ReCode successes have no extra products. Paired with COPA, PIVOT has 84 common successes, 11 COPA-only successes, and nine PIVOT-only successes on the original roster; the constructed counts are 115, one, and four. The small success-count differences accompany large reductions in model-token use, rather than establishing statistical equivalence.

B.3.2 QWEN3-235B EVALUATION

Table B10 gives scored-trajectory resources for the Qwen3-235B comparisons in Table 1A/B. For the original roster, COPA solves 81 tasks versus 74 for ReAct and 80 for CodeAct, using 92.2% and 79.6% fewer tokens. On the constructed roster, the counts are 115, 117, and 120, with token reductions of 92.3% and 63.8%. Every success rate uses 120 assigned tasks.

Table B10: **Qwen3-235B resources on scored trajectories.** Completion includes reasoning tokens. Resources follow the trajectory selection rule below.

System	Calls	Input	Completion	Total tokens	Tool calls
Original Shopping					
COPA	120	126,361	743,387	869,748	872
ReAct	1,240	9,524,392	1,613,456	11,137,848	1,022
CodeAct	416	3,064,565	1,208,442	4,273,007	2,028
Constructed Shopping					
COPA	120	128,409	658,712	787,121	1,978
ReAct	1,371	8,657,868	1,614,838	10,272,706	1,076
CodeAct	268	1,501,220	673,546	2,174,766	1,371

With level denominators 50/50/20, original-roster successes are 38/27/16 for COPA, 41/32/1 for ReAct, and 41/32/7 for CodeAct. Constructed-roster successes are 49/48/18, 49/48/20, and 50/50/20, respectively. These are descriptive differences across task levels.

Table B11 pairs the final task outcomes. Descriptive 95% intervals resample 120 task identities with replacement 10,000 times and describe task variation on each roster.

Table B11: **Qwen3-235B paired official successes.** Gain/loss denotes COPA-only/baseline-only success. Differences and descriptive 95% intervals are in percentage points; each comparison has 120 tasks.

Baseline	Both	Gain	Loss	Neither	Difference	95% interval
Original Shopping						
ReAct	56	25	18	21	+5.83	$[-4.17, 16.67]$
CodeAct	66	15	14	25	+0.83	$[-8.33, 10.00]$
Constructed Shopping						
ReAct	112	3	5	0	-1.67	$[-6.67, 2.50]$
CodeAct	115	0	5	0	-4.17	$[-8.33, -0.83]$

Two original-roster ReAct tasks without returned results and one CodeAct timeout count as unsuccessful, with their recorded usage included. When a recovery trajectory replaces an interrupted attempt, its outcome and recorded usage enter the totals; the replaced attempt is excluded. Shared response prefixes are counted once, and distinct model requests are counted separately. Resource totals thus cover the selected scoring trajectories rather than cumulative expenditure across all attempts.

B.3.3 GLM SHARED-TOOL COMPARISON

The GLM-5.3-Flash low/8K comparison uses the original 120 Shopping tasks and the same four public domain operations. Sampling uses temperature 1.0 and top-p 0.95, with an 8,192-token completion cap and low reasoning effort. COPA includes the type-binding rule evaluated in Appendix C.5; ReAct and CodeAct retain iterative execution and feedback. Table B12 reports these three controllers, using the same COPA outputs as the binding study.

COPA uses 97.99% fewer tokens than ReAct and 97.47% fewer than CodeAct, with five and six fewer task successes, respectively. Paired with CodeAct, 72 tasks succeed under both methods, eight only under COPA, 14 only under CodeAct, and 26 under neither. Successes without extra products are 80, 84, and 86 for COPA, ReAct, and CodeAct, respectively.

Table B12: **GLM shared-tool outcomes and model-token use.** Each method has 120 assigned tasks; resource totals include unsuccessful tasks.

System	Solved	Requests	Total tokens	Tool calls
COPA	80	120	180,858	672
ReAct	85	1,103	9,007,190	946
CodeAct	86	706	7,152,483	1,032

B.4 Scoring reconciliation for six-operation outputs

The six-operation study uses 120 versus 989 model calls and 279,191 versus 3,755,444 total tokens for COPA and ReAct, respectively.

The DeepPlanning COPA-minus-ReAct risk difference on the post-hoc official endpoint is $+0.83$ percentage points with a task-paired bootstrap 95% interval $[-6.67, +8.33]$. The official evaluator was applied post hoc to the same outputs previously scored by the registered ordered-reference rule (Table B13).

Ordered-reference and official scoring. The DeepPlanning strict (ordered-reference) and official counts score the same saved outputs. Both registered strict predicates require equality of the ordered product-ID list and coupon map; COPA additionally requires successful execution, whereas ReAct requires a null runtime error. The official evaluator scores the converted saved cart, ignores product order, deduplicates product IDs, counts matches of required IDs and coupon quantities, and reports extra products separately without penalizing them in the binary case-success decision.

Rescoring all 120 saved outputs per controller reproduces every strict and official success flag. For COPA, all 50 official-only successes (6 strict versus 56 official) have the same product multiset and coupon map as the reference, no extra or duplicate products, successful execution, and only a different product order. Of ReAct's 48 official-only successes (7 versus 55), 47 meet the same order-only description. The remaining case also has the correct product multiset and coupons but combines an order mismatch with a runtime error and a false execution flag. There are no strict-only successes in either controller. The two endpoints therefore differ mainly in order sensitivity, with no change to the evaluated outputs.

Table B13: **Two scoring rules applied to the same saved Shopping outputs.** Each system has 120 outputs. The ordered-reference rule is preregistered; official scoring is post hoc.

System	Ordered-reference success	Official task success
COPA	6/120	56/120
ReAct	7/120	55/120

The six-operation study evaluates saved normalized plans and their resulting outputs.

B.5 AppWorld controller and paired outcomes

The reported comparison uses the complete 90-task roster. COPA uses 82.43% fewer calls, 81.61% fewer tokens, and 84.88% less wall time, with seven fewer task successes and the same scenario-success count.

The 201 COPA calls comprise 90 initial plans, 89 pre-execution semantic reviews, and 22 further grounding, synthesis, replanning, or repair calls. The system combines deterministic execution with bounded model review and repair. Runtime repair can update an ephemeral tool or a task-local copy of a frozen-bank tool using public execution feedback. For a copied tool, the model edits its implementation while preserving the name and input/output schemas; the result is checked before re-execution. These local changes are not merged into the shared bank across tasks.

Exact token totals are 1,622,207 for COPA and 8,820,221 for Simplified ReAct; total elapsed times are 986.62 and 6,523.39 seconds, respectively. Controller settings are given in Appendix A.4.

On the 90 paired tasks, 75 succeed under both, three only under COPA, ten only under Simplified ReAct, and two under neither. Of the 30 scenarios, 22 are completed by both, three only by each controller, and two by neither.

B.6 Open-model Travel evaluation

All conditions evaluate the original 120 English Travel tasks with the same public-input prompts and the fixed scheduler described in Appendix A.2. Each task receives one model request, with no retries or model feedback during scheduling. The input contains the user query and public database candidates, including route prices, seat availability, and nearby distances. The response supplies trip context and named choices in JSON. The runtime parses final content; reasoning is saved but not read for candidate return. The official evaluator scores the resulting itinerary after execution. Solved requires all commonsense and personalized constraints to pass, with all 120 tasks in the denominator.

Model identities and sampling settings match Appendices B.1 and B.3: temperature 1.0 and top-p 0.95 for all models, with top-k 20 and low reasoning effort for the Qwen configurations. The initial conditions use an 8,192-token completion cap and a 65,536-token context window. The complete Qwen3-235B 32k run instead uses a 32,768-token cap and a 131,072-token context window; its per-task input messages are unchanged.

Table B14: **COPA Travel outcomes and model usage.** Each row is a complete 120-task run with the same scheduler. Cap is the completion-token limit; Completion includes reasoning tokens. Resource totals cover all task requests, including unsuccessful tasks, and exclude one content-free health check per run.

Model	Cap	Solved /120	Calls	Input	Completion	Total tokens
Qwen3.8-27B	8k	83	120	3,223,264	473,753	3,697,017
Gemma-4-31B-it	8k	72	120	3,320,294	57,943	3,378,237
Qwen3-235B	8k	30	120	3,237,103	921,057	4,158,160
Qwen3-235B	32k	72	120	3,237,103	1,021,163	4,258,266

Qwen3.8-27B produces 120 materialized itineraries; Gemma produces 116, with four final-content JSON parsing failures. Qwen3-235B's initial 8k run reaches the completion cap on 70 tasks and materializes the other 50. The subsequent 32k evaluation is an independent run on all 120 tasks, not a continuation of the capped responses. All 120 responses stop normally and produce itineraries, and the solved count rises from 30 to 72. The paired outcomes are 28 solved in both runs, 44 only at 32k, two only at 8k, and 46 in neither. Actual task-token use increases by 100,106 (2.41%); the 32k run's median completion is 8,541 tokens and its maximum is 13,598. A higher cap allows longer proposals without requiring every task to consume the full allowance.

C Frozen-Response Plan Processing

Contract rules appear in Appendix A.

C.1 Collection and endpoint accounting

A1 uses all 120 public DeepPlanning Shopping tasks under each of four selected open-weight serving configurations. Its six-operation proposal interface is distinct from the main four-operation system comparison: `get_user_context`, `find_best_product`, `build_product_bundle`, `optimize_coupons`, `evaluate_budget`, and `build_shopping_summary`. Each task and configuration pair receives one native strict-schema request with a fixed requested seed, a 32,768-token cap, and no retry. Private targets and scoring code are used for scoring only after all model requests have terminated. The 480 assigned records contain 479 model responses and one request failure; the failed request remains in every processing view's denominator. Raw already uses native schema decoding, so this comparison changes post-response processing rather than structured versus free-form generation. A1 uses prospective response collection followed by private scoring; its task identities and processing rules are benchmark-informed.

Table C15 gives the delivery-stage partition. The full A1 arm (contract + selection) applies product selection after COPA (contract) execution and inherits its endpoint. Together with Raw fallback, this makes monotone COMPLETE status a design property; the number and content of repairs are empirical. Selection can still change strict-reference agreement. The one GLM request failure remains in the assigned denominator.

Table C15: A1 delivery endpoints. C is contract-only; F is contract + selection. Both processing stages have identical endpoints. Each row closes to 120 tasks.

Configuration	View	COMPLETE	Contract invalid	Execution failure	Plan non-return	Request failure
DeepSeek-R1-Distill-Qwen-32B	Raw	3	101	16	0	0
	C / F	15	89	16	0	0
Seed-OSS-36B-Instruct	Raw	12	54	53	1	0
	C / F	28	40	51	1	0
Qwen3.8-27B	Raw	88	11	18	3	0
	C / F	93	6	18	3	0
GLM-5.3-Flash	Raw	60	4	55	0	1
	C / F	67	4	48	0	1

Table C16: Complete A1 stage results, with 120 assigned records per configuration. R/C/F denotes Raw, contract-only, and contract + selection. Sensitivity adds the independent constraint checks to strict-reference matching; pooled totals are descriptive.

Configuration	COMPLETE R/C/F	Strict ref. R/C/F	Sensitivity R/C/F
DeepSeek-R1-Distill-Qwen-32B	3/15/15	0/2/3	0/2/2
Seed-OSS-36B-Instruct	12/28/28	2/6/6	2/6/6
Qwen3.8-27B	88/93/93	19/19/21	16/16/17
GLM-5.3-Flash	60/67/67	13/16/16	11/13/13
Fixed-roster total	163/203/203	34/43/46	29/37/38

The prespecified strict-reference endpoint compares the predicted product-ID multiset and positive coupon-quantity map with the frozen benchmark reference, ignoring product order. The sensitivity endpoint additionally requires the independent candidate-identity, duplicate, slot-binding, coupon, budget, and price checks. Eight full-processing records across five tasks match the reference but fail only the frozen slot-binding check (DeepSeek 1, Qwen 4, GLM 3); all other listed checks pass. Slot binding uses bipartite matching, so output order does not explain this discrepancy. Both historical endpoints remain separate scoring views pending semantic adjudication; the disagreements alone neither establish model errors nor change COMPLETE outcomes.

The 40 new COMPLETE records comprise 31 Contract-Invalid and nine Execution-Failure responses; all 163 original COMPLETE records are retained. Strict reference gains/losses are 9/0 from Raw to COPA (contract), 3/0 from COPA (contract) to COPA, and 12/0 from Raw to COPA; the constraint-check sensitivity gives 8/0, 1/0, and 9/0, respectively.

Of the three contract-to-full reference gains, one is a newly executable DeepSeek record and two are already executable Qwen records; one also gains the constraint-check sensitivity.

C.2 Processing-stage controls

This post-hoc component analysis replays the saved A1 responses through 13 processing variants fixed before replay, without new model calls. The core variants appear below.

The syntax/schema control performs extraction and validation without value or reference mutation: 475 responses are accepted, four rejected, and one request failure retained. It changes two failure labels but no predictions, strict-reference counts, or COMPLETE outcomes. The retrospective replay's 48 prespecified per-model contrasts have Holm-adjusted $p = 1$ (minimum unadjusted $p = .125$).

Additional post-execution selection. In the frozen-response Shopping study, the full-processing arm can revise product combinations using the executable trace and public task data, then recompute

Table C17: Summary arms from the retrospective zero-call replay. Entries are strict-reference matches; each model column has denominator 120. COMPLETE uses the fixed-roster denominator 480. Pooled totals are descriptive.

Replay arm	DeepSeek	Seed	Qwen	GLM	Total match	COMPLETE
Raw strict	0	2	19	13	34/480	163/480
Syntax/schema control	0	2	19	13	34/480	163/480
COPA (contract)	2	6	19	16	43/480	203/480
Selection only	0	2	21	13	36/480	163/480
COPA (contract + selection)	3	6	21	16	46/480	203/480

coupons and totals. This differs from both contract repair and candidate return, which selects complete assignments already in a response. Raw-to-contract processing raises strict-reference matches from 34 to 43; the additional selection stage raises them to 46 without changing COMPLETE counts. The isolated selection arm requires an executable Raw trace. Table C17 retains both isolated stages and their combination.

Delivery-rule dependencies and actual prediction changes. Among the 40 new COMPLETE records, the mutually exclusive dependency sets are percentage normalization alone (6), direct-reference resolution alone (17), placeholder/alias resolution alone (9), percentage plus direct reference (5), and direct reference plus placeholder/alias (3). Their union is 40; the overlapping leave-one-out losses are 11, 25, and 12, respectively. Removing a rule need not isolate an independent causal contribution because the remaining ordered processor can depend on its output.

Percentage normalization, for example, changes `plan_confidence=100` to `1.0` to restore the declared numeric convention. A post-hoc single-rule example is a Seed-OSS plan with one producer with ID `step7` and tool `build_shopping_summary`. COPA (contract) changes only `final_from[0]` from that tool name to `step7`: Raw is contract-invalid, whereas COPA (contract) and COPA are COMPLETE and pass both frozen reference and constraint checks. Omitting direct-reference resolution leaves this record contract-invalid.

C.3 Prespecified paired inference

For endpoint Y and configuration m, the paired finite-roster estimate is

$$\widehat{\Delta}_m^Y = \frac{1}{120}\sum_{i=1}^{120}\big[Y_{im}(\mathrm{COPA}) - Y_{im}(\mathrm{Raw})\big]. \tag{2}$$

Prespecified strict-reference inference uses Newcombe method-10 paired intervals and exact McNemar/binomial tests, with Holm correction over the four configurations (Newcombe, 1998). COMPLETE, sensitivities, and pooled fixed-roster totals are descriptive. Table C18 reports all four configuration-level reference comparisons; their intervals include zero.

Table C18: A1 prespecified paired COPA-minus-Raw strict-reference inference for the full contract + selection arm. Each configuration has 120 assigned records; RD is the paired risk difference in percentage points, and Holm correction covers all four configurations.

Configuration	COPA-only / Raw-only	RD (95% CI), pp	Holm p
DeepSeek	3/0	+2.50 [−1.01, +7.09]	.750
Seed	4/0	+3.33 [−0.60, +8.43]	.500
Qwen	2/0	+1.67 [−1.65, +5.25]	.750
GLM	3/0	+2.50 [−1.20, +6.70]	.750

C.4 Deterministic processing cost

Timing starts before contract normalization and ends after selection, covering the combined Python processing path rather than pure compilation or a model request. Across 479 responses, median CPU

time is 1.50 ms and P95 is 144.11 ms (empirical nearest rank). Configuration medians range from 1.24 to 96.43 ms; maximum traced Python allocation is 528.46 KiB. The one request failure has no response to process. These CPU measurements differ from the candidate reader's stage-specific elapsed times.

C.5 SCHEMA-GUIDED REFERENCE BINDING

This study evaluates the additional type-binding rule in Appendix A on saved GLM-5.3-Flash Shopping plans using the four shared operations. For example, an array-valued reference can be wrapped in an extra singleton list. The processor removes such a wrapper only when the original whole argument fails its public type-compatibility check and the rewritten whole argument passes. It changes the binding, not the proposed operations or the domain-tool implementations.

The complete low/8K cohort contains 120 responses: 34 diagnostic responses and 86 collected after the component freeze. The latter are new responses on the original task roster, not new tasks. Both processing views reuse the same response, tool environment, and official evaluator.

Table C19: **Schema-guided binding on saved GLM plans.** Off/on toggles only the binding rule. The 86-response row is a subset of the complete cohort. Gains/losses refer to official task success.

Response set	n	Executable off/on	Solved off/on	Gains/losses
Complete low/8K	120	104/115	72/80	8/0
Post-freeze responses	86	74/81	50/56	6/0

The rule changes 58 bindings in 11 plans in the complete cohort. All 11 become executable, and eight yield additional official successes, without additional model calls. All 80 successes contain no extra products. Across all tasks, extra-product counts change from 35 to 39 as more plans execute. The complete cohort's generation uses 180,858 model tokens. The response partition, component freeze, and paired scoring records are retained with the experiment artifacts; the system comparison using these same COPA outputs appears in Appendix B.3.3.

D CANDIDATE VALIDATION AND RETURN

The two studies below each use 48 newly constructed stress blocks and 768 assigned model calls. Their response sets are independent; processing views within a study reuse the same saved responses. Inference concerns these selected stress cohorts, with requested repeats treated as measurements within blocks.

Task construction and reference verification. An independent solver searches product combinations from public task inputs and reproduces the reference optimum for all 48 blocks in each study. These runs check the task references; their solutions are not supplied to the candidate processor, which validates and selects complete assignments already in model responses.

D.1 NATIVE-CANDIDATE RETURN CONFIRMATION

Fresh paired design. This confirmation uses a new set of 48 directed stress blocks, with 24 early and 24 late expansion constructions. Only expanded inputs are evaluated. Each block crosses Qwen3.8-27B and GLM-5.3-Flash, two generation prompts, 16,384/32,768 output-token caps, and two prespecified requested sampler seeds. The resulting 768 assigned calls all have raw responses. The two configurations receive identical public inputs, prompts, and output schemas within each condition. Each model/prompt/budget condition contains 96 responses: two repeated calls for each of 48 blocks. These calls are distinct from the 120 constructed system-evaluation tasks in Appendix B.3. Temperature is 1.0 and top-p is 0.95; Qwen also uses top-$k = 20$ and enabled, preserved low-effort thinking. Each cell has one provider attempt and no application retry; budgets use separate requests.

G0 is the original generation prompt. G1 appends the frozen candidate-recording instruction: obtain a complete feasible choice before an exhaustive proof, record it as a standalone JSON object, record at most eight incumbents in total, and finish with the best complete choice in final JSON. This

instruction was developed on a separate 128-call pilot and is unchanged in the confirmation. The pilot is excluded from every confirmation denominator. No exact input or normalized-block matches were found in the earlier task sets checked. The construction uses prior design knowledge.

Processing variants. Raw scores the unprocessed final answer. COPA (JSON) and COPA (JSON+list) instantiate the minimum-cost rule in Section 3 on the same saved response. JSON reads complete assignments encoded as output-schema objects, slot-to-product mappings, or lists of slot/product objects. The public validator checks product identities and uniqueness, slot/category and delivery constraints, and the minimum eco count, then computes the discounted price. The processor includes a publicly feasible Raw final and returns the candidate with the lowest public cost. It may replace an already complete final, but never joins partial choices or creates new product combinations.

The list extension adds contiguous six-line lists of the form `- s_...: p_...`, with native hexadecimal slot and product IDs. All six slots must appear exactly once, with distinct visible products. Incomplete and overlong segments are rejected as a whole, rather than searched with sliding windows. Public category, cutoff, eco, and cost checks then apply. The minimum-cost variant selects among feasible native lists and the feasible JSON result, using a sorted slot/product tuple to break cost ties. The first and last controls instead consider only the first or last feasible list; both still retain a cheaper feasible JSON result. Thus these controls vary candidate selection after the same list scan. Including a feasible JSON result ensures that list selection cannot increase its public cost.

Conflicting nonempty reasoning fields are rejected. Fixed limits bound the input length, scan size, candidate count, and processing time. The JSON stage falls back to Raw if no candidate qualifies or a resource limit is reached; a JSON resource rejection also disables list adaptation. The list stage retains the JSON result if it finds no feasible list or rejects the scan. Fallback preserves the earlier output, including an empty or invalid one, rather than counting it as a successful return. Private optimum labels are used only for subsequent scoring.

Table D20: Candidate processing across all confirmation conditions, 96 assigned calls per row. Raw and every COPA variant are offline views of the same responses. Entries are Exact/complete; first and last show Exact only and have the same complete counts as JSON+list. All complete outputs in this trial are publicly feasible. G0/G1 denote original/candidate-recording prompts.

Configuration	Prompt	Raw	COPA JSON	COPA first	COPA last	COPA JSON+list
Qwen 16k	G0	4/4	4/4	22	26	30/40
Qwen 32k	G0	57/59	57/59	65	69	72/77
Qwen 16k	G1	4/4	6/6	24	25	31/42
Qwen 32k	G1	65/67	65/67	77	78	80/86
GLM 16k	G0	0/0	0/1	0	0	0/1
GLM 32k	G0	23/24	27/30	27	27	27/30
GLM 16k	G1	0/0	0/0	0	0	0/0
GLM 32k	G1	5/5	6/6	7	7	7/7

Separating processing from prompting. Under the original prompt, list reading adds 26 correct Qwen returns at 16k and 15 at 32k. The processing benefit therefore does not depend on candidate-recording instructions. Changing those instructions is a separate intervention: with JSON+list fixed, G1 adds one and eight Qwen returns at 16k and 32k but reduces GLM's 32k returns from 27 to seven. Recording is therefore optional; processing gains are evaluated within each prompt. Table D21 reports the contrasts and intervals.

First-list selection already recovers 18 and 12 Qwen G1 answers beyond the JSON baseline at 16k and 32k. Minimum public cost adds another seven and three over first-list selection. Both candidate coverage and selection among existing complete choices therefore matter. The final JSON+list outputs still include 11 feasible suboptimal Qwen answers at 16k and six at 32k under G1; feasibility does not establish optimality.

Recovery sources and cost. The 25 primary Qwen 16k recoveries all originate from strict token caps, span 22 blocks, and split 12/13 across the two repeats. All 15 Qwen 32k recoveries originate

from stopped responses with empty finals, span 12 blocks, and split 7/8 across repeats. GLM's single G1 32k recovery is a strict cap. Across both prompts and budgets, Qwen has 81 list-over-JSON recoveries: 50 caps and 31 stop-empty finals. These are recovered requests, not 81 different tasks, and they do not all belong to the primary G1 comparison. The records identify the observed termination and missing final, not the internal cause of that delivery failure.

The 768 calls use 20,955,237 reported tokens: 9,778,535 for Qwen and 11,176,702 for GLM. All reported processing views use zero additional model calls. For Qwen G1 at 16k/32k, the original run measures median list-stage wall times of 0.537/0.953 ms and process CPU times of 0.537/0.949 ms; total postprocessing wall times are 2.321/2.635 ms. These are measured function times.

Prespecified inference. The four primary contrasts are the G1 Exact changes from the JSON variant to the JSON+list variant, separately for each configuration and budget:

$$\Delta_{m,h} = \frac{1}{48}\sum_{b=1}^{48}\frac{1}{2}\sum_{s=1}^{2}\left(Y_{b,s,m,h}^{\mathrm{G1,JSON+list}} - Y_{b,s,m,h}^{\mathrm{G1,JSON}}\right).$$

The bootstrap uses 10,000 resamples, independently resampling 24 blocks within each early/late stratum and retaining both repeats and all conditions within a block. Percentile intervals use linear interpolation. We report 95% descriptive intervals and the prespecified 98.75% intervals for four Bonferroni-adjusted primary comparisons. These are approximate block-bootstrap intervals for the selected cohort. The two Qwen primary estimates exceed the prespecified $+10$ percentage-point practical target; the two GLM estimates do not. The target applies to point estimates, not to interval lower bounds, and did not govern stopping or sample expansion. GLM's all-zero 16k sample yields a degenerate interval, not proof of a zero effect in every population.

Table D21: All 20 prespecified decomposition contrasts in percentage points, with descriptive 95% block-bootstrap intervals below each estimate. A: G1 JSON+list minus G0 Raw; B: G1 minus G0 under Raw; C/D: JSON+list minus JSON under G1/G0; E: G1 minus G0 under JSON+list. C is the primary same-response contrast; its 98.75% intervals appear in Table 3. A combines generation and processing changes.

Configuration	A: Combined	B: Prompt/Raw	C: List/G1	D: List/G0	E: Prompt/List
Qwen 16k	+28.1 [18.8, 37.5]	0.0 [−5.2, 6.3]	+26.0 [17.7, 35.4]	+27.1 [17.7, 36.5]	+1.0 [−13.5, 15.6]
Qwen 32k	+24.0 [13.5, 34.4]	+8.3 [−3.1, 19.8]	+15.6 [7.3, 24.0]	+15.6 [9.4, 22.9]	+8.3 [−2.1, 18.8]
GLM 16k	0.0 [0.0, 0.0]	0.0 [0.0, 0.0]	0.0 [0.0, 0.0]	0.0 [0.0, 0.0]	0.0 [0.0, 0.0]
GLM 32k	−16.7 [−25.0, −8.3]	−18.8 [−28.1, −10.4]	+1.0 [0.0, 3.1]	0.0 [0.0, 0.0]	−20.8 [−29.2, −12.5]

D.2 Candidate-recording and formatting-scope factorial

Question and design. The native-return confirmation found useful candidate-reading gains but mixed effects of its candidate-recording prompt (Appendix D.1). This follow-up asks whether clarifying the scope of the JSON requirement changes that prompt's effect. It uses 48 new synthetic Shopping blocks, 24 early and 24 late expansion constructions, evaluated only at their expanded cutoff in the original time encoding. Each block crosses two serving configurations, four prompt conditions, and two requested repeats at 32,768 output tokens: 768 assigned calls, all with provider responses and one attempt. These are new calls, separate from the preceding 768-call confirmation and from the 3,504-request fixed-answer budget studies. The analysis distinguishes two questions: whether changing the prompt improves generated responses, and whether processing a fixed response improves its returned answer. The prespecified prompt contrasts address the first; the within-prompt processing comparisons address the second.

The original system instruction ($S = 0$) requests exactly one JSON object without markdown or commentary. $S = 1$ adds: "This formatting requirement applies only to the final answer, not to intermediate reasoning." $P = 0$ uses the original task prompt; $P = 1$ appends the unchanged

candidate-recording instruction from Appendix D.1. It requests an initial complete feasible candidate, at most eight incumbents in total, and the best complete choice in final JSON. Only these instruction bytes change across arms: S does not change the actual output channels, reasoning parser, response schema, or model weights.

The configurations inherit the preceding confirmation's serving settings: temperature 1.0, top-p 0.95, Qwen top-k 20 with preserved low-effort thinking, and the same GLM serving configuration and parser. Two prespecified requested sampler seeds are paired across arms within each block. Same-seed arms are separate requests, not guaranteed identical trajectories. No exact input or normalized-block matches were found in the earlier task sets checked, including the preceding 48 confirmation blocks.

Unchanged public processing. The processing views reuse the frozen reader and selection rules in Appendix D.1. Raw uses the native final; JSON minimum-cost and native-list variants use the same saved response and public task data. No model repair, cross-segment assembly, or new assignment search is introduced. First/last controls retain a cheaper feasible JSON result. Candidate processing precedes private Exact scoring.

Table D22: Candidate processing across all prompt conditions. Each row contains 96 responses from 48 blocks and two repeats at 32k. Entries are Exact/complete except First/Last, which report Exact; their complete counts equal JSON+list. Every complete output here passes the public feasibility checks. Non-Raw columns are COPA variants.

Config.	Prompt	Raw	JSON	First	Last	JSON+list
Qwen	$S0P0$	62/67	62/67	68	69	70/80
Qwen	$S0P1$	70/75	70/75	77	82	83/90
Qwen	$S1P0$	65/73	65/73	70	70	73/83
Qwen	$S1P1$	74/77	74/77	81	83	85/90
GLM	$S0P0$	13/14	21/23	22	22	22/24
GLM	$S0P1$	1/2	6/8	7	7	7/10
GLM	$S1P0$	14/15	21/25	21	21	21/25
GLM	$S1P1$	5/5	10/10	13	13	13/13

Fixed responses: processing gains and recovery sources. Across the four arms, Qwen has 40 Raw-to-JSON+list Exact gains and no observed losses; all 40 come from adding native lists. GLM has 30 gains and no observed losses, comprising 25 from JSON candidate processing and five from native lists. These within-prompt gains in Table D22 isolate processing rather than instruction changes. Counts pool repeated responses to 48 tasks. The minimum-cost rule includes a recognized, publicly feasible baseline among the candidates considered for return.

HTTP-to-reader verification. Of Qwen's 40 list gains, 37 have provider-recorded stops and empty finals, while three hit the strict token cap. A post-hoc audit compares all 384 Qwen raw HTTP responses with their saved envelopes and normalized content and reasoning fields; all comparisons preserve the returned fields. All 37 stop-empty recoveries already have null final content in the HTTP payload, with complete candidates in the returned reasoning. This rules out client-side loss of a nonempty final in these records. The saved payload follows server-side channel parsing, so it does not distinguish model generation from server parsing or stopping behavior. All five GLM list gains come from strict caps.

Minimum-cost selection improves on both first-list and last-list returns in every Qwen prompt condition (Table D22). GLM's three list-selection policies have the same Exact counts in each arm. In the best observed Qwen condition, 85/96 is the JSON+list Exact result; the same-response gain is 74 to 85, not the cross-prompt change from 62 to 85.

Prompt changes: prespecified contrasts. Let Y_{sp} be JSON+list Exact, averaged over the two repeats within a block. The two primary contrasts for each configuration are

$$A = Y_{11} - Y_{10}, \qquad I = (Y_{11} - Y_{10}) - (Y_{01} - Y_{00}).$$

A tests the recording instruction under clarified formatting scope; I tests whether clarification changes its effect. The analysis resamples 24 blocks within each early/late stratum, preserving every arm and repeat in each sampled block. It uses 10,000 shared bootstrap draws and linearly interpolated percentile intervals. The four primary comparisons use 98.75% intervals; 95% intervals are descriptive. A confirmed active-prompt benefit requires $A \geq 5$ pp and a strictly positive adjusted lower bound.

Table D23: Four primary prompt-factorial contrasts, in percentage points. The 98.75% intervals implement the prespecified four-comparison adjustment. These compare generated responses under different instructions, with JSON+list held fixed; none excludes zero under the adjusted criterion.

Config.	Contrast	Estimate	95% interval	98.75% interval
Qwen	A	$+12.50$	$[3.12, 21.88]$	$[0.00, 23.96]$
Qwen	I	-1.04	$[-17.71, 14.58]$	$[-22.92, 18.75]$
GLM	A	-8.33	$[-19.79, 3.12]$	$[-23.96, 7.29]$
GLM	I	$+7.29$	$[-6.25, 20.83]$	$[-10.42, 26.04]$

Qwen's A varies from zero to $+25$ pp across the two repeats; GLM's is negative in both. The adjusted intervals do not establish a prompt benefit or formatting-scope interaction; the within-prompt processing gains reported above concern a separate intervention on the same responses.

Processing cost. Full-function median CPU times range from 2.24 to 2.37 ms for Qwen and 4.71 to 6.00 ms for GLM. The 768 requests consume 25,220,689 reported tokens. All reported processing views require zero additional model calls.

E Fixed-Answer and Budget Studies

E.1 Study construction and contrast definitions

These studies evaluate direct answer generation without candidate-return processing. The optimum is held fixed across eligibility conditions; model outputs are collected in separate requests for the assigned conditions.

Eligibility expansion. The unique optimal assignment, its cost, runner-up margin, visible product order, and output schema stay fixed within a block. Relaxing a cutoff admits more choices without improving the existing optimum; Appendix E.1.2 gives a concrete example.

Matched cutoff and budget contrasts. The matched Qwen/GLM construction uses cutoffs of 36, 60, and 84 hours. Twelve early blocks expand from two to six eligible products per slot from 36 to 60 hours and preserve eligibility from 60 to 84; twelve late blocks reverse these roles. Both steps change the cutoff by 24 hours with the remaining prompt bytes fixed. Each configuration/cohort cell crosses three cutoffs, H16/H32 (16k/32k-token) caps, and three pre-frozen requested seeds: 18 calls per block and 432 per cell. Budget siblings are separate requests, not continuations. Let $\bar{Y}_{bch}$ average three runs in block b, cutoff c, and budget h. The expansion-minus-preserving contrast compares admitting products with changing only the cutoff literal:

$$D^{Y}_{bh} = \begin{cases} (\bar{Y}_{b,60,h} - \bar{Y}_{b,36,h}) - (\bar{Y}_{b,84,h} - \bar{Y}_{b,60,h}), & b \text{ early}, \\ (\bar{Y}_{b,84,h} - \bar{Y}_{b,60,h}) - (\bar{Y}_{b,60,h} - \bar{Y}_{b,36,h}), & b \text{ late}. \end{cases} \tag{3}$$

For Exact and complete delivery, $\tau^{Y}_{b} = D^{Y}_{b,H32} - D^{Y}_{b,H16}$; strict-cap reduction uses the reverse budget difference. Each cell has 24 analysis blocks. Decompose the matched contrast as $M = U - P = \tau$, where U is the expansion-step budget interaction and P the preserving-step interaction. Positive τ means a greater budget benefit for the expansion step than for the preserving step. Neither U nor τ equals the expanded-input H32-minus-H16 success difference.

The prespecified criteria require at least 18/24 positive blocks for Exact, complete delivery, and cap reduction, together with checks on prompt-token matching and infrastructure failures. Q and G label stress cohorts originally assigned to Qwen and GLM, respectively. Both are constructed without target-model outputs to expose repair-greedy failure after expansion. Original Qwen/Q and GLM/G

trials are followed by crossed Qwen/G and GLM/Q trials, frozen after the original outcomes were known; same-roster model differences are descriptive.

The equivalent-encoding study and its paired contrast are specified in Appendix E.5.

E.1.1 STRESS-COHORT CONSTRUCTION

The fresh generation seed produces 72 inputs and prompts and 24 cutoff-normalized blocks, with no exact matches in the earlier task sets checked. The unchanged, target-output-blind admission rule accepts 24 of 373 constructions (6.43%): the small surface admits the repair-greedy path, while expansion defeats it with at least 1,000 cents of regret and the same unique optimum. This defines a selected stress cohort.

GLM cohort. The GLM study uses the same three co-primary estimands, $18/24$ criteria, and task-generation and admission rules as the Qwen fresh design. It draws a new task-generation seed and accepts 24 of 245 output-blind construction attempts (9.80%). Its 72 inputs, 72 prompts, and 24 normalized blocks have no exact match in the earlier task sets checked or either Qwen stepped cohort. These are new tasks from the same engineered stress family.

GLM-5.3-Flash uses one fixed service with three prespecified requested seeds in contiguous batches; block remains the analysis unit. The protocol was specified before all 432 one-attempt calls, which were completed before scoring. A separate enumerator checks 1,103,760 feasible assignments across the 72 inputs and verifies every target's unique optimum, cost, and margin. Native numeric reasoning telemetry is unavailable.

E.1.2 OUTCOME CLASSIFICATION AND CONSTRUCTION EXAMPLES

The outcome definitions below accompany the crossed-cohort results in Appendix E.3.

Outcome classification. The Qwen scorer first separates requests without a provider response into provider/transport failures or local infrastructure failures. For a parsed, schema-conforming response, it checks equality with the exact label first, then assigns complete non-Exact if all required slots are present and every selected ID is a nonempty string in the supplied product evidence; remaining outputs in this branch are visible incomplete/semantic-invalid. The public schema already requires all six named slots and `abstain=false`. Only outside this parsed/schema-conforming branch does the scorer test strict cap: empty final content, length stop, completion tokens equal to the assigned cap, reasoning content present, and one provider attempt. Other records are contract/interface failures. Complete delivery combines Exact and complete non-Exact and does not require global eco feasibility. The same outcome categories apply to all 432 GLM requests under its prespecified endpoint definitions.

Cancellation and its limits. Using the contrast in Appendix E.1, an additive scalar trend $a_h + b_h c$ at a fixed budget cancels: the early weights satisfy $-1 + 2 - 1 = 0$ and $-36 + 2(60) - 84 = 0$; the late weights reverse both signs. This algebra does not establish that real prompt-literal effects are affine. Nonlinear dependence on the numeric coordinate, tokenization, and interactions with the remaining prompt can survive. Early/late balancing and the negative control do not identify latent computation as a mediator.

Construction examples. For Figure E1, we selected the first early and late blocks in the fixed index order after observing the results: GLM blocks 0 and 1, respectively. Independent enumeration from the public product inputs verifies their unique target assignments at all three cutoffs. The early block has optimal cost 61,274 cents and runner-up margin 118 cents. Expansion increases its feasible set from 42 to 30,618 baskets while retaining this optimum and margin. The late block has cost 61,409 and margin 5 cents. Early Exact counts across c36/c60/c84 are $(3, 0, 0)$ at both budgets, giving $D_{H16} = D_{H32} = -1$ and $\tau = 0$. Late counts are $(3, 3, 0)$ at H16 and $(2, 3, 0)$ at H32, giving $D_{H16} = -1$, $D_{H32} = -4/3$, and $\tau = -1/3$.

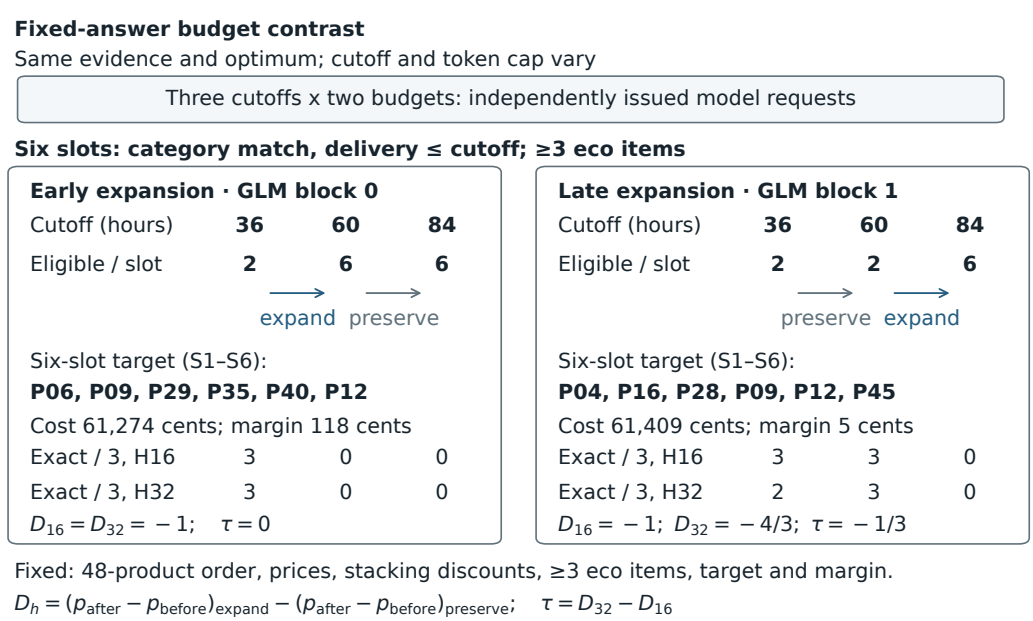

Figure E1: Early/late construction examples with fixed targets and equal cutoff steps. Product aliases are local to each example. Cutoff and budget conditions use independently issued model requests.

E.2 GPT AND MULTI-CONFIGURATION BUDGET STUDIES

Prespecified GPT studies. Both GPT studies fixed their designs before generation. We report their outcome partitions descriptively here. Both use selected synthetic blocks whose construction does not observe target-model outputs. The low, medium, and high labels refer to two, four, and six eligible products per slot, not model reasoning effort: the GPT-5.4-high configuration is held fixed, requesting the `gpt-5.4-2026-03-05` endpoint with temperature unset and a strict provider response schema. Each scheduled cell receives one provider attempt, without target-exposed retry or replacement.

The earlier study has 12 blocks and 36 calls, one per eligibility tier under a 16,384-token completion cap. Within each block, the unique optimal selection, its cost and nearest-distinct margin, visible evidence, product order, bundle rules, schema, and prompt byte length are fixed; six delivery-cutoff literals change the eligible joint set from $2^6 = 64$ to $4^6 = 4{,}096$ to $6^6 = 46{,}656$ assignments. Provider prompt-token counts match in 11/12 blocks; the medium prompt differs by one token in the remaining block. Reasoning-token lower bounds increase strictly across eligibility tiers in 10/12 blocks. Five blocks exhaust the budget only at the high tier, and one also exhausts it at a lower tier. One high-tier provider failure has no token-usage information and is not counted as a capped response.

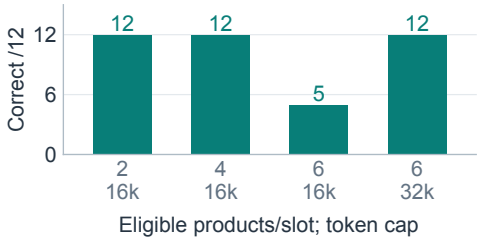

Figure E2: **GPT fixed-answer study.** Twelve blocks; fixed evidence and optimum. Each condition uses separate requests without candidate-return processing.

The successor uses 12 new prospectively frozen blocks and 48 calls: low@16k, medium@16k, high@16k, and the byte-identical high input@32k. All 12 high inputs receive both budgets in the pre-outcome schedule, rather than allocating H32 after observing which H16 calls fail. The target tuple, cost, margin, evidence, schema, and provider prompt-token count remain fixed within each block. The four arms deliver 12/12, 12/12, 5/12, and 12/12 Exact answers (Figure E2), with reported reasoning-token lower-bound medians 1,775, 10,183.5, 16,384, and 19,255. All seven non-returns occur at high16, with empty final content, a length stop, and both completion and reasoning usage equal to 16,384. Capped reasoning counts are lower bounds; H16/H32 siblings are separately issued calls, not continuations. Both studies are separate from the later equal-step Qwen/GLM experiment.

Prespecified criteria. H1 requires strictly increasing reasoning-token lower bounds across tiers in at least 10/12 blocks. H2 requires at least six blocks with high-tier-only budget exhaustion and at most one with exhaustion at a lower tier. The successor adds H3: at least six high16 exhaustions, six paired high32 Exact recoveries, ten high32 Exact answers, and at most one high16-to-high32 Exact loss. The earlier study meets H1 but not H2. The successor meets all three: H1 holds in 12/12 blocks; H2 has seven high-only and zero lower-tier exhaustions; the four H3 counts are 7, 7, 12, and 0. Criteria are evaluated separately for each cohort.

Table E24: Complete assigned-call outcome partitions. Rows are separate finite schedules. "Other" is model/interface non-return; the provider-failure column (Prov.) is separate and does not require token-usage information. Configuration labels are Qwen (Qwen3.8-27B), GLM (GLM-5.3-Flash), and DeepSeek (DeepSeek-R1-Distill-Qwen-32B).

Study	Configuration	Assigned	Exact	Wrong	Cap	Other	Prov.
GPT earlier	GPT-5.4	36	28	0	7	0	1
GPT successor	GPT-5.4	48	41	0	7	0	0
Replication	Qwen	48	21	2	21	4	0
Replication	GLM	48	13	0	35	0	0
Replication	DeepSeek	48	1	33	14	0	0

The three-model replication uses a new, non-overlapping 12-block cohort. H1 cannot be evaluated because these configurations do not expose reliable integer reasoning-token counts; completion tokens are not substituted.

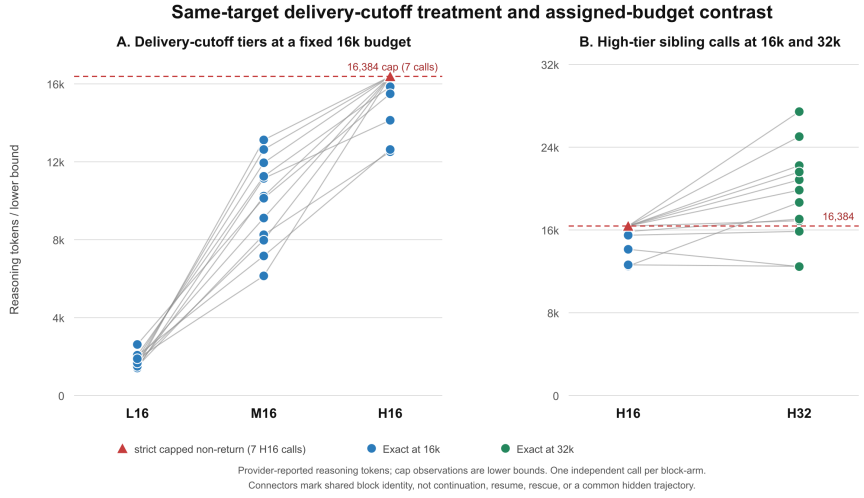

Figure E3: Successor GPT reasoning-token lower bounds. Lines join calls sharing a block identity. H16 and H32 are separately issued common-seed siblings. Red triangles are strict capped non-returns at 16,384 tokens and hence lower bounds.

E.3 CROSSED-COHORT RESULTS

The crossed follow-up evaluates each fixed serving configuration on the other configuration's existing stress cohort, completing the two original diagonal cells without generating or rescreening blocks. The two new cells were specified jointly after the original diagonal outcomes were known and before either new evaluation began. Within each cohort, request identities, cutoff/budget assignments, repeats, requested sampler seeds, and global call order are paired.

Estimands and analysis units. Each cell follows the 24-block, 432-call design and contrast definitions in Appendix E.1, giving 1,728 calls across four cells. Rates and signs use exact rational arithmetic before decimal display. As a difference of two rate changes, D_h can lie outside $[-1, 1]$. The original criterion requires positive interactions in at least 18/24 blocks for each endpoint. Neither the calls nor the within-block repeats are independent block-level observations.

Table E25: Four-cell Exact contrasts. Means summarize 24 blocks per cell; signs are Exact τ positive/zero/negative. E/C/S gives positive-block counts for Exact, complete delivery, and strict-cap; each original threshold is 18/24. Qwen denotes its low-thinking configuration.

Config.	Cohort	D_{H16}	D_{H32}	τ	$+/0/-$	E/C/S
Qwen	Qwen	-0.9861	-0.1667	0.8194	22/1/1	22/23/24
Qwen	GLM	-0.9167	-0.2500	0.6667	22/1/1	22/22/23
GLM	Qwen	-0.9722	-0.7778	0.1944	9/10/5	9/7/7
GLM	GLM	-1.0556	-0.6667	0.3889	12/9/3	12/11/11

Qwen meets the prespecified criteria in both cohorts, whereas GLM does not. The H16 Exact contrast is negative in all 24 blocks in each of the other three cells, but in 23 blocks for Qwen on the GLM cohort.

Same-cohort paired contrasts. The post-hoc mean of block-paired $\tau_{\text{Qwen}} - \tau_{\text{GLM}}$ for Exact is $5/8 = 0.6250$ in the Qwen cohort, with positive/zero/negative counts $17/3/4$, and $5/18 = 0.2778$ in the GLM cohort, with counts $16/4/4$. These post-hoc contrasts describe variation in the configuration gap across the two stress cohorts and their blocks.

Table E26: Complete-delivery and strict-cap contrasts across the four cells. Signs refer to the oriented τ, not to D_{H16} or accuracy. Complete delivery includes Exact; strict-cap uses the reverse budget orientation defined above.

Config. / cohort	Endpoint	D_{H16}	D_{H32}	τ	$+/0/-$
Qwen / Qwen	Complete	-0.9861	-0.1528	0.8333	23/0/1
Qwen / Qwen	Strict-cap	0.9583	-0.0139	0.9722	24/0/0
Qwen / GLM	Complete	-0.9167	-0.1667	0.7500	22/1/1
Qwen / GLM	Strict-cap	0.8889	0.0417	0.8472	23/0/1
GLM / Qwen	Complete	-1.0000	-0.8472	0.1528	7/14/3
GLM / Qwen	Strict-cap	1.0000	0.8611	0.1389	7/14/3
GLM / GLM	Complete	-1.0139	-0.6667	0.3472	11/11/2
GLM / GLM	Strict-cap	1.0139	0.6667	0.3472	11/11/2

At H16 the expansion-before Exact counts, in the order of Table E25, are $72, 71, 70, 71$ out of 72; after expansion they are $2, 4, 0, 0$. At H32 the Qwen configurations deliver $52/72$ and $51/72$ Exact answers, but retain $18/72$ and $15/72$ contract failures, respectively. Reduction of strict-cap non-return is therefore not equivalent to correct delivery. Across all 432 assigned calls per cell, the corresponding (E,N,S,F) partitions are $(292, 4, 104, 32)$, $(289, 10, 104, 29)$, $(220, 14, 197, 1)$, and $(229, 11, 192, 0)$; the other three terminal categories are again zero.

Model settings. Qwen uses a low-thinking template and omits the temperature request field; omission does not mean temperature zero. GLM requests temperature 1.0 and top-p 0.95. Thus the comparison concerns serving configurations, including sampling and runtime, rather than isolated model weights.

Table E27: Complete expanded-surface accounting at each budget. Each row has denominator 72 (one expansion-after cutoff per block, three repeats): c60 for early and c84 for late expansion. E is Exact, N is complete non-Exact, S is strict-cap non-return, and F is model/interface contract failure. Other terminal categories are zero, so E+N+S+F=72. N excludes E and does not imply public feasibility.

Config. / cohort	Budget	E	N	S	F
Qwen / Qwen	H16	2	0	69	1
Qwen / Qwen	H32	52	2	0	18
Qwen / GLM	H16	4	0	67	1
Qwen / GLM	H32	51	4	2	15
GLM / Qwen	H16	0	0	72	0
GLM / Qwen	H32	11	0	61	0
GLM / GLM	H16	0	0	72	0
GLM / GLM	H32	17	1	54	0

E.3.1 Matching and requested-seed sensitivity

A post-hoc analysis of all 1,728 scored responses pairs block, cutoff, budget, and requested-seed identities within each cohort. The mean Qwen-minus-GLM Exact difference is $1/2$ under expansion-only U and $5/8$ under matched M on Q; it is $5/12$ under U and $5/18$ under M on G. The mean preserving-step contrast P is zero in each new off-diagonal cell, but its positive/zero/negative block counts are 5/14/5 for Qwen/G and 7/12/5 for GLM/Q. A zero mean therefore combines positive and negative block-level effects.

The one-seed and leave-one-seed-out ranges are reported in Table E30B. Each range covers three summaries on the same 24 blocks, and every minimum is positive.

The positive ranges show consistency across the requested seeds, which were evaluated in separate batches within each serving configuration.

E.3.2 Initial and diagonal-cohort detail

Table E28 compares the initial Qwen cohort with the Qwen and GLM confirmation cohorts under the same 24-block, 432-call design. The two confirmation cohorts supply the diagonal cells of Appendix E.3; their calls are counted only once.

Table E28: Aligned small and eligibility-expanded surfaces in the three stepped instantiations. Every row has 72 calls over blocks and seeds. Instantiations are separate; calls within one are repeated observations over 24 blocks.

Cohort	Surface	Budget	Exact	Complete non-Exact	Contract	Wrong	Cap
Qwen initial	Small	H16	71	0	0	1	0
Qwen initial	Small	H32	71	0	0	1	0
Qwen initial	Expanded	H16	1	0	1	0	70
Qwen initial	Expanded	H32	46	0	23	3	0
Qwen fresh	Small	H16	72	0	0	0	0
Qwen fresh	Small	H32	72	0	0	0	0
Qwen fresh	Expanded	H16	2	0	1	0	69
Qwen fresh	Expanded	H32	52	2	18	0	0
GLM fresh	Small	H16	71	1	0	0	0
GLM fresh	Small	H32	70	2	0	0	0
GLM fresh	Expanded	H16	0	0	0	0	72
GLM fresh	Expanded	H32	17	1	0	0	54

In the initial cohort the strict-cap expansion-minus-placebo budget interaction is positive in 24/24 blocks. Applying the later definitions post hoc gives 19/24 positive Exact and 21/24 positive complete-terminal blocks; these were not co-primary endpoints of the initial protocol. In the Qwen fresh cohort, all three were frozen jointly before target output. Table E29 reports both prospectively frozen trials' block distributions. Both diagonal trials match prompt-token counts in 24/24 blocks and have zero

infrastructure-failure blocks; the prespecified checks require 24/24 matches and at most four affected blocks.

Table E29: Prospectively frozen block interactions after averaging the three seed repeats within each block. The threshold is at least 18 positive blocks; these finite-cohort criteria are not population significance tests.

Trial	Endpoint	Positive	Mean	SD	Median	Range	Positive by seed
Qwen fresh	Exact	22/24	0.819	0.590	0.667	$[-0.667, 2.000]$	18, 18, 14
Qwen fresh	Complete delivery	23/24	0.833	0.573	0.833	$[-0.667, 2.000]$	19, 19, 14
Qwen fresh	Strict cap	24/24	0.972	0.195	1.000	$[0.333, 1.333]$	23, 23, 23
GLM fresh	Exact	12/24	0.389	0.595	0.167	$[-0.333, 2.000]$	6, 10, 5
GLM fresh	Complete delivery	11/24	0.347	0.551	0.000	$[-0.333, 2.000]$	6, 8, 4
GLM fresh	Strict cap	11/24	0.347	0.551	0.000	$[-0.333, 2.000]$	6, 8, 4

No request in the initial or confirmation cohorts is retried or classified as an infrastructure failure. Of the GLM confirmation's 11 complete non-Exact outputs, nine are feasible but suboptimal and two violate the global eco constraint, illustrating the distinction between complete delivery and feasibility.

E.4 Qwen multi-seed budget stability

The stability study reuses the same 12 high-tier blocks; three exogenously derived sampler seeds are repeated observations within block. Each H16/H32 sibling is issued once on the same service with no application retry. All 36 H16 calls satisfy the strict-cap definition. The H32 partition is 26 Exact, eight empty-stop contract/interface failures, one visible wrong, and one strict cap.

The seed-specific Exact counts appear in Table E30A. Native numeric reasoning-token telemetry is unavailable for all 72 calls. All 36 H32 reasoning strings exactly extend their paired H16 strings; none of 72 cross-seed comparisons does.

Table E30: Budget repeatability across requested seeds. **A:** 12 high-tier blocks, three seeds, 72 calls; all 36 H16 calls cap. Other: non-Exact H32 outcomes. **B:** Qwen-minus-GLM Exact recovery gaps; ranges span three estimates on the same 24 blocks per cohort. U: expansion-only; M: matched. Seed subsets reuse the same tasks.

A. Qwen budget replication

Seed	H16 Exact	H32 Exact	Other
1	0/12	10/12	2/12
2	0/12	8/12	4/12
3	0/12	8/12	4/12
Total	0/36	26/36	10/36

B. Crossed-study mean gap ranges

Cohort / subset	U	M
Q / one seed	.4167 to .5833	.4167 to .9167
Q / omit one	.4583 to .5417	.4792 to .7292
G / one seed	.3750 to .4583	.1667 to .5000
G / omit one	.3958 to .4375	.1667 to .3333

E.5 Equivalent numeric encodings and answer delivery

E.5.1 Paired construction and prespecified contrast

The numeric study adds 24 new directed stress blocks, twelve early and twelve late, shared by Qwen and GLM. The original representation uses cutoffs 36/60/84 hours. Adding four hours to every product delivery time and every cutoff gives 40/64/88. For any product, $d \leq c$ if and only if $d + 4 \leq c + 4$; all other predicates, product IDs, order, prices, discounts, and output-schema content remain unchanged. Consequently, each paired encoding has the same feasible assignments and costs, unique optimizer, and runner-up margin. Before generation, enumeration verifies this equivalence for all 72 distinct block and cutoff pairs. Exact-input and normalized-block overlap checks use the earlier task sets.

Both configurations are assigned all encodings and budgets before generation. Each block contributes $2 \times 3 \times 2 \times 3 = 36$ assignments per configuration: two encodings, three cutoffs, H16/H32, and three requested seeds. The two encodings are interleaved in a predetermined order within the balanced schedule. The 1,728 assignments are repeated measurements of 24 base blocks.

For each configuration, the sole new primary contrast is

$$\Gamma_b = \tau_b^{\text{Exact}}(+4) - \tau_b^{\text{Exact}}(\text{original}), \qquad \bar{\Gamma} = \frac{1}{24}\sum_{b=1}^{24}\Gamma_b, \tag{4}$$

where τ retains the expansion-minus-preserving, H32-minus-H16 definition in Appendix E.1. It is not the change in expanded-surface accuracy alone. We report the mean and signs of the 24 paired block contrasts, without a population test or equivalence margin. Complete-delivery and cap contrasts and the earlier $18/24$ criteria are diagnostics.

Table E31: Encoding effects and secondary endpoint contrasts. E/C/K is the count of positive Exact/complete/cap interactions. Secondary counts exclude infrastructure-affected blocks from positive classification, while retaining the fixed $18/24$ threshold; primary Γ includes every block.

Config.	$\bar{\Gamma}$	Positive	Zero	Negative	E/C/K original; $+4$
Qwen	$-5/36$	8	7	9	20/20/20; 18/19/21
GLM	$-1/72$	10	7	7	7/6/6; 8/9/9

On expanded inputs at H16, Qwen returns 6/72 Exact answers under the original encoding and 2/72 under the translated encoding; GLM returns 0/72 under both. The supplementary material reports all paired block contrasts. Both configurations have 17 nonzero block changes, with positive and negative values. GLM's expanded H32 Exact count rises from 9 to 12 under translation even though its mean matched contrast decreases from $5/72$ to $1/18$. The broad delivery pattern recurs under this semantics-preserving transformation, while the matched effect sizes remain encoding-sensitive.

E.5.2 Model settings and missing outcomes

Both Qwen3.8-27B and GLM-5.3-Flash explicitly request temperature 1.0 and top-p 0.95. Qwen additionally sets top-k 20, enables and preserves thinking, and requests low reasoning effort. These request settings are fixed between encodings. In particular, temperature is explicitly set here but omitted in the earlier Qwen budget study. Three prespecified requested sampler seeds are paired across conditions.

Before generation, tokenization gives equal rendered input token counts for each original/translated pair under both frozen tokenizer revisions; token IDs differ. Provider usage is available on both sides for 429 of 432 pairs per configuration, and all 429 have equal prompt counts. The other three pairs lack usage on one side, rather than displaying unequal counts. Native reasoning-token telemetry is unavailable for this study; total tokens are not relabeled as reasoning expenditure.

Table E32: Numeric-study terminal accounting. Top rows cover the entire 864-assignment schedule per configuration. Bottom rows partition each 72-assignment expanded H32 cell. CNE means complete non-Exact, I/F means contract/interface failure, and Int. means an interrupted request.

Config.	Schedule / encoding	Exact	CNE	I/F	Cap	Int.	Assigned
Qwen	All	569	14	74	204	3	864
GLM	All	456	11	0	394	3	864
Qwen	H32 expanded, original	50	3	18	1	0	72
Qwen	H32 expanded, $+4$	44	3	23	2	0	72
GLM	H32 expanded, original	9	0	0	63	0	72
GLM	H32 expanded, $+4$	12	1	0	58	1	72

Of 1,728 assigned records, 1,722 have saved HTTP responses and six were interrupted after starting. The latter retain false endpoint indicators in the primary assigned-record analysis; they are not model caps. A post-hoc sensitivity assigns each unobserved Exact indicator either zero or one. The resulting bounds for $\bar{\Gamma}$ are $[-11/72, -7/72]$ for Qwen and $[-1/36, 1/72]$ for GLM. They bound missing-cell assignments only and are not confidence intervals.

E.6 DIRECT-GENERATION BUDGET STRESS ACROSS 37 MODELS

This study measures direct generation without COPA plan execution or candidate-return processing. Its scores describe the evaluated model-serving configurations, not COPA system success.

Increasing the high-tier budget raises schema-valid decisions from 239 to 323, but Exact answers from zero to only 15 across the 444 paired assignments. The larger budget thus improves output delivery more often than task correctness. The same 12 blocks recur across arms and configurations, which differ in runtime, parser, template, sampling, and topology. Table E33 reports all model-level counts; Tables E34 and E35 give the aggregate outcomes and transitions.

Under the prespecified criteria, none of the seven configurations with the required reasoning-token telemetry meets H1; H1 is unavailable for the other 30. H2 is unmet in all 37. H3 is unmet in 13 and unavailable or lacks the required base signal in 24. These are finite-schedule criteria, not population tests.

Table E33: **Direct-generation budget profiles without COPA processing.** Each model uses its evaluated serving configuration: 12 blocks, four arms, 48 assigned calls per model, 1,776 total. Entries are out of 12. First four columns: Exact; last: schema-valid visible decisions at High16 → High32, which need not be feasible.

Model	Low16	Med16	High16	High32	Valid 16 → 32
Olmo-3-7B-Think	0	0	0	0	2 → 9
Olmo-3.1-32B-Think	0	0	0	0	0 → 12
ERNIE-4.5-21B-A3B-Thinking	0	0	0	0	6 → 11
Seed-OSS-36B-Instruct	11	2	0	3	1 → 12
DeepSeek-R1-Distill-Llama-70B	1	0	0	0	12 → 11
DeepSeek-R1-Distill-Llama-8B	0	0	0	0	8 → 11
DeepSeek-R1-Distill-Qwen-1.5B	0	0	0	0	6 → 8
DeepSeek-R1-Distill-Qwen-14B	0	0	0	0	11 → 12
DeepSeek-R1-Distill-Qwen-32B	3	0	0	0	9 → 12
DeepSeek-R1-Distill-Qwen-7B	0	0	0	0	10 → 9
DeepSeek-V4-Flash	0	0	0	0	12 → 12
gemma-4-12B-it	0	0	0	0	12 → 12
gemma-4-26B-A4B-it	0	0	0	0	12 → 12
gemma-4-31B-it	0	0	0	0	12 → 12
granite-4.2-30b	6	0	0	2	0 → 2
granite-4.2-3b	0	0	0	0	0 → 5
granite-4.2-8b	0	0	0	0	0 → 0
LongCat-Flash-Lite	0	0	0	0	12 → 12
MiniMax-M2.7	11	0	0	0	0 → 0
Ministral-3-14B-Reasoning-2512	0	0	0	0	12 → 12
Ministral-3-3B-Reasoning-2512	0	0	0	0	12 → 12
Ministral-3-8B-Reasoning-2512	0	0	0	0	12 → 12
Mistral-Small-4-119B-2603	0	0	0	0	12 → 12
Nemotron-3-Nano-30B-A3B	1	0	0	0	0 → 2
Nemotron-3-Super-120B-A12B	12	0	0	0	0 → 0
Nemotron-3.5-Lightning-30B-A3B	0	0	0	0	0 → 0
gpt-oss-120b	2	0	0	0	11 → 12
gpt-oss-20b	0	0	0	0	12 → 12
Qwen3-235B-A22B-Thinking-2507	0	0	0	0	9 → 10
Qwen3-30B-A3B-Thinking-2507	4	0	0	1	0 → 12
Qwen3-4B-Thinking-2507	0	0	0	0	0 → 9
Qwen3.8-27B	12	1	0	9	0 → 9
Qwen3.8-Flash-Next	12	0	0	0	0 → 0
Step-3.5-Flash	0	0	0	0	12 → 12
Hunyuan-4B-Instruct	0	0	0	0	10 → 11
Hunyuan-7B-Instruct	0	0	0	0	12 → 12
GLM-5.3-Flash	12	0	0	0	0 → 0
Total (444 calls per arm)	87	3	0	15	239 → 323

Table E34: Direct-generation outcomes by arm over 37 serving configurations. Each arm has 12 repeated block identities per configuration.

Arm	Assigned	Exact	Visible wrong	Strict cap	Length/no valid decision	Request/interface/invalid	Valid decisions
Low16	444	87	261	77	14	5/0/0	348
Medium16	444	3	246	167	26	1/0/1	249
High16	444	0	239	172	32	1/0/0	239
High32	444	15	308	107	11	0/3/0	323
Total	1,776	105	1,054	523	83	7/3/1	1,159

Table E35: Paired direct-generation transitions for the 37-model study.

Contrast	Exact gains	Exact losses	Valid-decision gains	Valid-decision losses
Low16 → Medium16	0	84	19	118
Medium16 → High16	0	3	17	27
High16 → High32	15	0	90	6