REGRESSION IN REPAIR: UNDERSTANDING PLAN REPAIR DYNAMICS IN LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent advances have shown that adding feedback and tools enables large language model (LLM) agents to solve increasingly complex planning problems across diverse domains. One effective paradigm for plan repair involves LLM planners producing a candidate plan, with programs checking every constraint of this plan and feedback or tools fixing the constraints it violates. In this paper, we study plan repair under coupled constraints, which price every choice against one budget so that every infeasible option of a slot is cheaper than every feasible one. We systematically analyze repair dynamics across three dimensions, coupling, the content of the repair message, and the order of the options, through empirical studies on 4,800 synthetic tasks, a second environment with requirements in text, and 900 TravelPlanner plans using two open-source models (4B and 8B parameters), gpt-4.1 and gpt-4.1-mini. Our experiments reveal three key findings about repair effectiveness when models answer in one step, namely (1) Uncoupled tasks allow feedback to more reliably fix failed attempts without breaking another constraint; (2) Exact totals from a calculator tool produce repairs that break fewer constraints than messages that only name the failed check; (3) Option choice is generally correlated with price, but this relationship varies with coupling, under which the cheaper choice is the infeasible one. These findings reveal opportunities for optimizing basic repair strategies in planning applications. First, given the same model, a single call on options sorted by the shared resources can nearly match a calculator tool in pass rate (e.g., with price and duration sorted by their weighted sum, the gap to the calculator tool shrinks from 20.0 and 15.0 points to 1.0 and 4.5 points at a quarter of its cost). Second, we identify cases where four samples at temperature 1.0 offer limited advantages over one attempt, as all samples read the same option order, suggesting that sampling alone cannot overcome position bias.

1 INTRODUCTION

Large language model (LLM) agents have advanced rapidly in solving planning tasks such as travel planning and calendar scheduling, yet their plans remain unreliable, often violating subtle or obvious constraints (Xie et al., 2024; Zheng et al., 2024b). Plan repair (Madaan et al., 2023; Shinn et al., 2023; Kambhampati et al., 2024) has emerged as a central mechanism to fix such violations in a scalable manner. Recent work has increasingly focused on repair with an external checker (Kambhampati et al., 2024; Stechly et al., 2024), which frames repair as a loop in which a program checks each constraint of the plan and the model then revises the plan. One valuable downstream application of plan repair is agent design, where a repair method is added to check candidate plans, fix violations, and return a plan that meets every constraint. This approach underlies techniques such as Self-Refine (Madaan et al., 2023), Reflexion (Shinn et al., 2023), LLM-Modulo (Kambhampati et al., 2024), and program-aided language models (Gao et al., 2022).

Current practice in agent evaluation often deploys per-constraint pass rates as the measure of a repair method (Xie et al., 2024; Shao et al., 2026). This practice rests on the assumption that the constraints of a plan fail independently (i.e., a removed violation is a removed failure), an assumption made in recent reliability analyses (Khanal et al., 2026; Rabanser et al., 2026). However, this practice may be misleading given that the constraints of a planning task often draw on one resource, a structure referred to as coupling constraints in mathematical optimization. This coupling appears in the mul-

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.

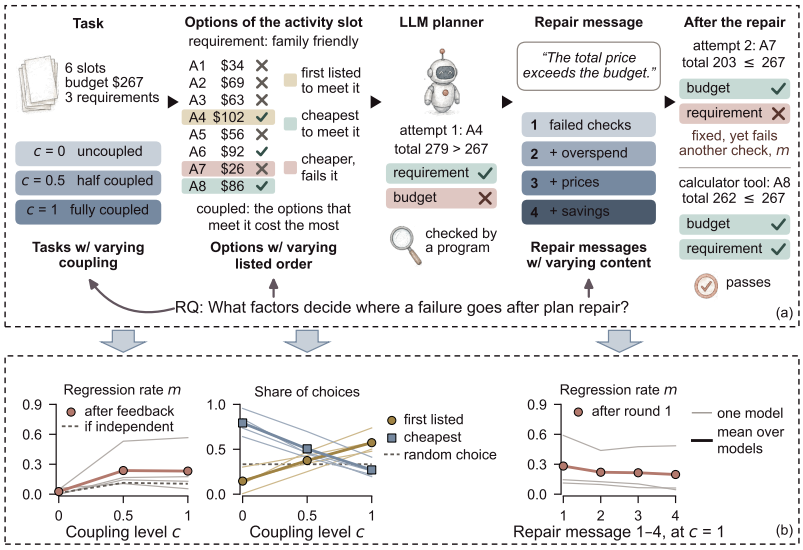

Figure 1: **Overview of our study on plan repair under coupled constraints.** (a) We consider plan repair with constraints checked by a program: an LLM planner takes one option per slot, a program checks every constraint, and a repair message lets the planner revise. Controlled experiments vary the coupling of the task, the listed order of the options and the content of the repair message; the example is one logged task (gpt-4.1, $c = 1$). (b) Our analysis reveals three patterns: coupling governs the regressions that follow feedback; under coupling models take the first listed feasible option instead of the cheapest; and more exact repair messages leave fewer fixed attempts failing another check. In each pattern the cheaper choice under coupling is the infeasible one.

tidimensional knapsack problem, where every item draws on the same capacities (Chu & Beasley, 1998), and in travel planning, where every stay, meal and attraction draws on one budget (Xie et al., 2024). Thus, it is worth investigating plan repair as a change in which constraints fail rather than merely a reduction in how many fail.

Despite extensive research on self-correction dynamics and the factors influencing repair quality (Huang et al., 2023; Kamoi et al., 2024; Liu, 2026), the dynamics of plan repair under coupled constraints remain largely unexplored. In particular, little is known about how coupling, the content of the repair message, and the order of the options interact to determine where a failure goes after a repair. Without understanding repair dynamics, one can risk misallocating calls by defaulting to more samples or more tools when a single call on a sorted list might suffice. This gap in understanding motivates our central research question, namely what factors decide where a failure goes after plan repair.

In this paper, we present a systematic study of plan repair across three dimensions, coupling, the content of the repair message, and the order of the options, shown in Figure 1. We quantify repair performance by measuring the probability of a fixed attempt failing another constraint and of a passing attempt failing after the repair in controlled experimental settings. We focus on planning tasks whose constraints are checked by programs and answered in one step, in a synthetic budget-planning environment with one coupling parameter, a second environment that states each requirement in text, and TravelPlanner (Xie et al., 2024). This allows us to objectively measure the outcome of every check, while simulating the pass-rate evaluation in which repair methods are typically compared in practice. While our experiments use these environments as a testbed, we believe the insights should extend to any domain where several constraints draw on one resource and can be reliably checked.

Main Findings. While prior work showed that the order of the options influences the choices of language models (Zheng et al., 2024a; Liu et al., 2024), we reveal that two additional factors, coupling and the content of the repair message, also critically influence plan repair, as illustrated in Figure 1. Our analysis reveals the following. Coupling primarily governs the regressions that follow feedback; fixed attempts are more likely to fail another constraint on coupled tasks than on uncou-

pled ones. The content of the repair message influences regression; repairs made with exact totals are less likely to break another constraint than those made with a notice that the budget is exceeded. Option order correlates with the chosen option in a manner dependent on coupling; weak without coupling, where gpt-4.1 takes the cheapest option that meets a requirement, and strong under coupling, where the GPT models and Qwen3-4B take the first listed option that meets the requirement while Mistral-7B misjudges feasibility instead. Our empirical analysis includes 33,600 attempts on 4,800 synthetic tasks, 200 tasks per method in the second environment, and 900 TravelPlanner plans, evaluated across two open-source models and two GPT-4.1 models, with two models of other families as a check.

Application to Planning Agents. We demonstrate the practical implications of our findings for planning agents. First, given the same model, the pass rate of a single call on sorted options can nearly match the pass rate of a calculator tool. For instance, gpt-4.1-mini achieves comparable pass rates with options sorted by a weighted sum of price and duration and with the calculator tool, at a quarter of the cost. Second, we identify regimes where four samples at temperature 1.0 offer limited additional benefit over one attempt and can be replaced by four samples under shuffled option orders, with temperature providing limited gains. This occurs with coupled constraints or with options listed in a fixed order.

2 Related work

Plan Repair and Evaluation. The deployment of feedback as a repair signal has emerged as a central mechanism for LLM planning, with efforts focusing on revision loops driven by feedback (Madaan et al., 2023; Shinn et al., 2023). Beyond self-refinement, repair approaches include external checkers (Kambhampati et al., 2024; Stechly et al., 2024), solver-based planners (Hao et al., 2024), where a solver satisfies the constraints jointly, reinforcement learning on constraints (Qi et al., 2026), multi-agent decomposition (Zhang et al., 2024; Lu et al., 2025), where a merge step settles the shared constraints, and constraint-aware agents (Choi et al., 2025; Shi et al., 2025; Chen et al., 2026a). Planning and instruction-following benchmarks score outputs against many constraints (Shao et al., 2026; Zheng et al., 2024b; Zhou et al., 2023; Jiang et al., 2023b; Wen et al., 2024; Chen et al., 2026b; Wang et al., 2026), where per-constraint pass rates decay gradually while joint success collapses (Vasileva, 2026). Prior work identifies several factors influencing repair performance, with the reliability of the feedback being particularly important. Huang et al. (2023) find that intrinsic self-correction does not improve reasoning, and Kamoi et al. (2024) that correction works when feedback is reliable and external. Qiu et al. (2025) show that a code-structured workflow on TravelPlanner trades commonsense violations for hard-constraint violations. That a repair creates a new violation is the classic difficulty of repair-based search for constraint satisfaction (Minton et al., 1992). Our work extends these findings by identifying unexplored factors and showing that the violation a repair creates is driven by how a language model reads a list. Tank & Nama (2026) is a concurrent work with a similar research objective, finding that the best skills added to office agents win mainly by regressing less, complementing our analysis of how coupling and the repair message shape regressions.

Option Order in LLM Decisions. Early studies explore how to effectively remove order effects to improve LLM decisions. Zheng et al. (2024a) show that language models prefer some answer positions in multiple-choice questions, while Liu et al. (2024) find that models use information at the start and end of a long context better than in the middle. Recent work varies the order; Tang et al. (2024) aggregate rankings over shuffled lists to cancel such effects and Bito et al. (2025) select items one at a time. Training can also reward consistent answers across permuted options (Zheng et al., 2026), which sorting and shuffling complement at inference time. Our work studies the factors driving plan repair and explores their implications for option order in planning agents.

3 Experimental setup

3.1 Preliminaries

Task and Check Space. Let t denote a task with a set of checks J_t. An attempt at t under method a yields $F_{taj} \in \{0, 1\}$ for each applicable check j, where 1 is a failure, and a check whose inputs are

invalid is not applicable; the attempt passes if no check fails. As discussed in Section 1, our study uses tasks whose checks are programs, allowing us to rigorously evaluate every attempt against every constraint while simulating the pass-rate evaluation of planning benchmarks.

Baseline and Method. A baseline a maps a task t to an attempt, denoted by its set of failed checks $\Phi_a(t) = \{j : F_{taj} = 1\}$, with failure count $Y_{ta} = |\Phi_a(t)|$. The baseline is single generation. A method b differs from a in one module, and the checks it addresses, T_b, are the checks that the module's input or instruction names, fixed from the design before any result. A calculator tool returns a total price, so it addresses the budget check, a checklist addresses the requirement checks, and feedback addresses the checks the first attempt failed, $T_b(t) = \Phi_a(t)$, so its products over $j \notin T_b$ run over $j \notin T_b(t)$. The prompt templates are provided in Appendix A.

Coupling. We measure the coupling of a task using its share of coupled slots. A task asks for one option in each of six slots, and each option has a price; a slot with a requirement is constrained, and an option is feasible when it meets its slot's requirement. A constrained slot is coupled when every feasible option costs more than every infeasible one, so that meeting the budget and meeting the requirement trade against each other. The separation is strict by construction; on TravelPlanner the cheapest accommodation meets the requirement in 36.9% of 103 constrained slots (Appendix C.7), so a real task couples more mildly. The coupling level c is the share of constrained slots that are coupled, $c \in \{0, 0.5, 1\}$, and for each model we generate 200 tasks in each cell of coupling level and number of constrained slots K. These 200 tasks per model and cell are used to estimate the regression rates and the pass rate of every method.

Task Difficulty. We define the difficulty of a setting as the pass rate p_0 of single generation on its tasks. For each model a preregistered pilot (60 tasks per setting) chose the budget slack, and at $c = 0$ the number of feasible options, to equalize p_0 across coupling levels. Matching within 5 points failed at $K = 3$ for all four models (largest gaps 0.10 to 0.33) and at $K = 6$ for both GPT models, so coupling and difficulty are partly confounded (Appendix B.2). It provides a model-specific way to compare coupling levels at similar difficulty, extending prior work (Xie et al., 2024), which set difficulty by the number of constraints.

Regression Metrics and Evaluation. We evaluate methods using the regression rate among fixed attempts, m_b, the probability that an attempt whose addressed failures are fixed fails another check, and the regression rate among passing attempts, r_b, the probability that an attempt that passes under the baseline fails under b. The fixed attempts of b are $S_b = \{t : \emptyset \neq \Phi_a(t) \subseteq T_b$ and $F_{tbj} = 0$ for all $j \in T_b\}$, and

$$m_b = \frac{|\{t \in S_b : \Phi_b(t) \neq \emptyset\}|}{|S_b|}, \qquad r_b = \frac{|\{t : \Phi_a(t) = \emptyset,\ \Phi_b(t) \neq \emptyset\}|}{|\{t : \Phi_a(t) = \emptyset\}|}. \tag{1}$$

As a reference, m_b^{ind} averages $1 - \prod_{j \notin T_b}(1 - p_{bj})$ over S_b, where p_{bj} is the failure rate of check j over all attempts of b in the same cell to which the check applies; it is the value of m_b expected if the other checks failed independently of the repair. Intervals are 95% percentile intervals over 10,000 task resamples within cells (Efron, 1979), and tests within a research question are corrected with Benjamini–Hochberg (Benjamini & Hochberg, 1995). The hypotheses were preregistered before the first paid call; Appendix B.1 lists each with its outcome, including failures, and marks the analyses defined after the results. Each method runs on all 200 tasks of every cell at temperature 0 except for sampling, unless a controlled subset is specified.

Option Order and Sorting. We consider the setting of showing each slot's options as a list and letting the model choose one option per slot. We measure how often a model takes each option in Section 4.3; the rule used below idealises that measurement, with the model taking the cheapest option when it is feasible and otherwise the first listed option it judges feasible. Sorting shows each slot's options in ascending order of the shared resource and relabels them by position; it reads only the resource field, never the requirements, and adds no model call. If the model follows this position bias and judges feasibility correctly, then under ascending price order it returns the cheapest plan that meets every requirement, which fits the budget whenever any such plan does (Proposition 1 in Appendix C.3). With two shared resources, price and duration, we sort by price plus λ times duration, a Lagrangian (Everett, 1963), with λ the budget divided by the duration limit; no guarantee holds when a duality gap separates the two limits. This position-based view captures the plan a model returns under a given order without tying performance to a specific solver.

3.2 TASKS AND MODELS

Budget Planning. We generate a total of 4,800 tasks from a budget-planning environment with one coupling parameter, 1,200 for each model. Each of the six slots offers eight options with a price and two yes/no attributes, a task names $K \in \{3, 6\}$ constrained slots, each with a requirement on one attribute, and the budget is $B = \max(C^*, \lfloor C^*(1 + \delta) \rfloor)$, where C^* is the price of the cheapest plan that meets every requirement and δ is the slack. The scorer runs $K + 2$ checks, one per requirement, a budget check and a format check, so per-constraint outcomes need no judge. Table 4 in Appendix B.2 lists the seven methods, single generation, feedback, a calculator tool, a checklist, both together, per-requirement sub-agents and best-of-four sampling. We use 200 tasks in each of the six cells of coupling level and K, except for the pilot, from which we take 60 tasks per setting to balance difficulty across coupling levels.

Text Requirements. We use a second version of the environment as our test of option order. It keeps the six slots and the coupling parameter, states each requirement in words, and describes each of 12 options per slot without naming the attribute, so only the model can judge whether an option meets a requirement, while price and duration stay structured fields that a sort can read. The budget is set so that 5% of the plans meeting every requirement fit, and a mode with two resources adds a duration limit and places the feasible options of a coupled slot on a price–duration frontier, so no sort by one resource meets both limits. The methods are those of Table 2, each on the same 200 tasks at $c = 1$ (Appendix C.3). The second environment consists of six-slot tasks whose requirements only the model can judge, spanning one and two shared resources.

Travel Planning. We collect 900 plans for 180 validation queries from TravelPlanner (Xie et al., 2024), five gpt-4.1 samples at temperature 1.0 per query in sole-planning mode, and re-grade them check by check with the benchmark's evaluator, which leaves 12 checks per plan (Appendix C.7). The 900 plans serve the dispersion analysis (Appendix C.4), one repair round and the sorting and repair runs use gpt-4.1 (Appendix C.6), and 60 extraction tasks with 40 independent nodes each, a set that shares no resource between nodes, serve as a negative control.

Models. We use two open-source models from one family, Qwen3-8B and Qwen3-4B (Yang et al., 2025), served from the official bf16 weights on vLLM 0.29.0 (Kwon et al., 2023) with thinking disabled, and two closed-source models, gpt-4.1 and gpt-4.1-mini, through the OpenAI API at temperature 0. Each model runs every method on 200 tasks in each of the six cells of coupling level and K, 33,600 attempts in total; the second environment uses gpt-4.1, gpt-4.1-mini and Qwen3-4B, and two models of other families, Phi-4-mini and Mistral-7B, run single generation and feedback on the tasks of gpt-4.1-mini as a check. We use the provider names of the models in figures and tables; see Appendix C.1 for the results of every model.

4 EXPERIMENTAL RESULTS

Our experiments focus on how coupling and the content of the repair message influence plan repair, and how the order of the options influences the chosen plan. We present the three research questions and main findings below.

RQ1. How does coupling affect plan repair? (Section 4.1) The share of fixed attempts that fail another check increases steadily with coupling after feedback, meaning repairs break another constraint more often on coupled tasks. However, regressions after the calculator tool show no predictable relationship with coupling on the GPT models. This indicates that coupling primarily influences the regressions that follow feedback.

RQ2. How does the content of the repair message influence regressions? (Section 4.2) As the message carries more price information, the share of fixed attempts failing another check decreases substantially while stating the overspend changes first-round repairs only slightly. This reveals that the content of the repair message primarily determines where a fixed attempt lands.

RQ3. How does coupling change which option a model takes? (Section 4.3) Option choice and price are generally correlated, as gpt-4.1 takes the cheapest feasible option without coupling. However, the form of correlation depends heavily on coupling; under coupling the GPT and Qwen models take the first listed feasible option, while Mistral-7B misjudges feasibility instead.

Table 1: Where the choice of each method lands: the share of its outputs that newly violate a requirement the input met, at $c = 0$ and $c = 1$. Feedback counts fixed attempts and the aggregator passing attempts (gpt-4.1, gpt-4.1-mini, Qwen3-8B, Qwen3-4B); the solver counts the slots of both GPT models with a wrongly listed option; the TravelPlanner row gives each share over the queries to which the check applies.

Choice made by	$c = 0$	$c = 1$
Feedback repair	0.0%, 0.0%, 4.7%, 4.6%	17.5%, 10.5%, 3.9%, 55.3%
Aggregator over sub-agent reports	0.0%, 0.0%, 2.2%, 0.0%	0.0%, 2.1%, 69.1%, 46.6%
Exact solver on the model's labels	8 of 52 slots	40 of 40 slots
Sorted list on TravelPlanner	no coupling level; default vs ascending price: min-night failures 47.3% vs 55.6%, budget failures 35.5% vs 15.4%	

Throughout this section, we use single generation defined in Section 3.1 as the baseline, which fixes the attempts a method can repair before the method runs, to avoid selection by the method. In Appendix D.1, we validate all key findings using seven checks, from two difficulty controls and the price relation reversed to an exact solver on the model's labels and two models of other families, and show that the trends remain consistent (Table 20).

4.1 How does coupling affect plan repair?

To examine how coupling influences plan repair, we generate tasks at three coupling levels, $c = 0$, 0.5 and 1, termed "uncoupled", "half coupled" and "fully coupled".

Coupling primarily influences whether a repair breaks another constraint after feedback. Our analysis is conducted at two levels of granularity, the attempt level and the check level. Both analyses reveal that coupling mainly shapes the regressions that follow feedback, while not consistently affecting the regressions that follow a calculator tool.

At the attempt level, we compute m and m^{ind} of all fixed attempts within each coupling level. As shown in Figure 1b (per model in Figure 3), m increases steadily as coupling rises after feedback, while m after the calculator tool shows no clear trend. At $c = 0$ none of the fixed attempts of gpt-4.1 ($S = 22$, and 0 of 75 with 1,000 added tasks) or gpt-4.1-mini ($S = 114$) fails another check, and at $c = 1$ the shares are 17.5% [8.2, 28.3], 12.9% [7.4, 19.0] and 56.7% [48.9, 64.5] on gpt-4.1, gpt-4.1-mini and Qwen3-4B, against 4.0%, 4.8% and 31.0% expected under independence. This pattern is consistent across three of the four models and both added families, while Qwen3-8B shows no trend (Appendices C.1 and D.2). At the check level, we pool the fixed attempts of each model at $c = 1$ and find that 10 of 10, 13 of 16 and 83 of 85 of those failing another check had failed only the budget and then violate a requirement. In contrast, regressions after the calculator tool show no consistent correlation with coupling, with no fixed attempt failing another check on either GPT model at any level (S from 42 to 184) and 24.5% and 17.1% on Qwen3-4B, which rarely calls the tool (Appendix C.1). In Appendix D.1, we show that the main dynamics about m we identified hold under three difficulty controls (Table 20), while the null result on Qwen3-8B reflects which attempts it fixes. We additionally confirm that this trend persists without conditioning on the repair in Appendix D.1 and that 63 of 108 plans fixed by one round of feedback on TravelPlanner fail another check in Appendix C.2.

To understand this pattern, the failure transitions in Appendix D.1 show that models meet the budget by switching a slot to an infeasible option. In a coupled slot every infeasible option is cheaper, and the budget message names no slot, so the cheaper choice becomes a violation of a requirement the attempt used to meet. We support this explanation with a direct test, in which we swap the price blocks of every coupled slot so that every feasible option is cheaper than every infeasible one, with experimental details provided in Appendix D.1. As we show in Table 1, the choice of a method lands on a cheaper infeasible option more often at $c = 1$ than at $c = 0$ for the feedback repair on three of the four models, for the aggregator over sub-agent reports on the Qwen models and for an exact solver on the options both GPT models wrongly list, and with the price relation reversed none of the 71 and 26 fixed attempts of gpt-4.1-mini and gpt-4.1 fails another check, against 12.9% and 17.5% at $c = 1$. This shows that the cheaper infeasible options of coupled slots are a large driving factor of regressions.

4.2 How does the content of the repair message influence regressions?

We study how the content of the repair message affects regressions by giving each model the same failing first attempts with different messages. We continue the feedback loop for three rounds from the same saved first attempts and vary only what the message carries, the failed checks alone, the overspend, the price of every chosen option, or the saving available in each slot. Details are provided in Appendix C.2.

As shown in Table 10 in Appendix C.2, the first-round repair rate remains nearly unchanged when the message states the overspend and increases further when the message lists prices or savings. For gpt-4.1 at $c = 1$, stating the overspend changes first-round repairs by -3.0 points $[-7.3, 1.2]$, while listing the saving available in each slot adds 36.4 points [27.9, 44.8]. Naming only the slots that hold a cheaper feasible option, with no number, adds 34.5 points [26.7, 42.4]. This suggests that the size of the overspend influences plan repair in a relatively mild way.

The content of the repair message correlates with regression in plan repair. In Table 10, moving from the failed checks alone to the saving in each slot, the share of fixed attempts that fail another check falls. For example, for gpt-4.1 at $c = 1$, the share is 14.5% after one round, 3.1% after two and 1.3% after three with the failed checks alone, against 4.5%, 0.0% and 0.6% with the saving available in each slot, and no attempt fixed by a message that only names the slots fails another check. The overall pattern is consistent across both GPT models and all three rounds. These results show that the content of the repair message strongly modulates regressions. We further verify that this trend remains robust with the calculator tool, which gives exact totals and leaves no fixed attempt failing another check on either GPT model, with the calculator tool plus checklist on all four models (Appendix C.8) and with plan-and-solve prompting (Appendix C.2).

We interpret this phenomenon through the logged attempts in Appendix C.2. A total lets the model price a cheaper feasible plan before committing to it, and a list of savings names the slots in which the budget can be met without breaking a requirement. The budget notice gives no price to check, so the model meets the budget by taking a cheaper option that may be infeasible. We support this explanation with a large-scale analysis. We count, for the method with one sub-agent per requirement, the violated slots in which the aggregator picked an infeasible option that a report listed (Appendix B.3). The results in Table 1 show a clear trend; the aggregator newly violates a requirement in 69.1% and 46.6% of the passing attempts of the Qwen models at $c = 1$, and the reports list the cheap infeasible option it picks in 98.6% of the violated slots on Qwen3-8B. This is consistent with Wu et al. (2023), who find that fine-grained feedback trains better reward models than one holistic score.

4.3 How does coupling change which option a model takes?

We measure which option a model takes in each constrained slot and evaluate the pass rate using the random order of the options as a natural experiment. Each model is evaluated on single attempts at $c = 1$ in constrained slots whose first listed feasible option is not the cheapest one, and we report the choices against random choice among the three feasible options. To characterize the relationship between order and the pass rate, we weight single attempts by the probability, computable from the prices, that the plan taken under this position bias fits the budget. We also report the choices on TravelPlanner with shuffled reference tables as another measure of position bias.

Our counts show a strong overall preference for the first listed feasible option under coupling, with gpt-4.1 showing more price comparison than the other models. gpt-4.1, gpt-4.1-mini, Qwen3-8B and Qwen3-4B take the first listed feasible option in 50.2%, 73.8%, 47.8% and 57.6% of these slots and the cheapest feasible one in 41.2%, 19.5%, 25.7% and 21.7%, against 33.3% for each under random choice. While this global trend validates findings in prior work showing that language models prefer some answer positions (Zheng et al., 2024a; Liu et al., 2024), a closer inspection of the trend reveals that position matters only once the model has filtered for a requirement.

Option order influences the pass rate differently based on coupling. Stratified analysis by coupling reveals that the order matters only under coupling. Without coupling, gpt-4.1 takes the cheapest feasible option in 95.5% of constrained slots, so the order has little effect. Under coupling, whether the plan this position bias produces fits the budget depends on the order alone. Weighting by this propensity, single attempts at $c = 1$ whose order lets that plan fit pass more often by 31.4 points [18.9, 43.4] on gpt-4.1, 53.3 [44.3, 62.0] on gpt-4.1-mini and 24.8 [8.7, 40.8] on Qwen3-4B.

Table 2: Sorting the options in the second environment at $c = 1$ (200 paired tasks per cell; 105 and 104 in the program-writing cells of gpt-4.1). Entries are pass rates (cost per task in thousandths of a USD). With two resources the weighted sum of price and duration replaces descending price and orders the checklist on sorted options. The last three rows are a pipeline with an exact solver, a program the model writes, and plan-and-solve prompting (Appendix C.3, with Qwen3-4B).

Method	mini, one resource	mini, two resources	gpt-4.1, one resource	gpt-4.1, two resources
Default order	0.315 (0.88)	0.230 (1.01)	0.525 (4.43)	0.410 (5.08)
Ascending price	0.570 (0.88)	0.215 (1.01)	0.740 (4.43)	0.350 (5.08)
Descending price	0.160 (0.88)	–	0.605 (4.44)	–
Weighted sum	–	0.440 (1.01)	–	0.515 (5.08)
Calculator	0.740 (3.75)	0.430 (3.68)	0.930 (15.82)	0.560 (21.49)
Checklist	0.940 (1.99)	0.710 (3.05)	0.970 (9.43)	0.830 (13.04)
Checklist, sorted options	0.930 (1.89)	0.800 (2.89)	0.985 (8.41)	0.900 (11.30)
Best-of-4, temperature 1.0	0.365 (3.54)	0.305 (4.05)	0.605 (17.73)	0.485 (20.32)
Best-of-4, shuffled orders	0.540 (3.54)	0.465 (4.05)	0.775 (17.73)	0.575 (20.32)
Model lists, solver picks	0.885 (0.93)	0.880 (1.06)	0.955 (4.65)	0.960 (5.28)
Model writes the program	0.800 (2.14)	0.815 (2.71)	0.981 (11.82)	0.904 (13.97)
Plan-and-solve prompting	0.910 (3.38)	0.755 (4.78)	0.940 (13.90)	0.885 (21.73)

The calculator tool and checklist shrink the effect on the GPT models to 1.4 to 3.8 points, since with a total in hand gpt-4.1 takes the cheapest feasible option in 96.3% of these slots, and plan-and-solve prompting (Wang et al., 2023) removes the preference as well (Appendix C.2), while best-of-four sampling at temperature 1.0 keeps the effect at 25.0 and 43.8 points because every sample reads the same order. The exception is Mistral-7B, which takes the first listed feasible option in 25.8% of these slots and misjudges feasibility instead (Appendix D.2). On TravelPlanner with shuffled reference tables, gpt-4.1 takes the first listed accommodation of a city in 23.6% of 1,187 stays against 8.4% by chance. Training Qwen3-4B with a reward for consistent answers across permuted options raises its pass rate but leaves the first listed feasible option at 55.7% of these slots and m after feedback above its reference (Appendix D.4). These analyses were defined after the results. Appendix C.3 provides additional results (Table 12), including a simulation of why each order yields the pass rate it does under this position bias.

5 Application to planning agents

Our analysis in Section 4 is conducted with regressions themselves as the end goal. However, our findings have direct implications for planning agents. We analyze two research questions in agent settings that naturally arise out of our previous findings, and present our results below.

RQ4. Given a fixed model, can a single call on sorted options match methods with more calls? (Section 5.1) A single call on options sorted by price can beat best-of-four sampling and, sorted by a weighted sum of the shared resources, nearly match the calculator tool. Sorting gains peak with one shared resource by listing the cheapest feasible option first while adding no call.

RQ5. Can temperature sampling match the gains of shuffled option orders? (Section 5.2) The gain gap between shuffled orders and temperature 1.0 stays wide under coupling and with a fixed order, where every temperature sample reads the same order.

The following sections present results on the second environment, with complete results including Qwen3-4B in Appendix C.3. In Appendix C.6, we show that these findings can guide plan repair on TravelPlanner and lead to large gains in pass rate.

5.1 Can a single call on sorted options match methods with more calls?

We evaluate sorting with a fixed model by varying the order of the options and reporting pass rates before and after sorting, along with the cost per task.

Sorting gains peak with one shared resource, enabling a single call to approach methods with more calls. As shown in Table 2, both GPT models start from low pass rates under the default order but improve markedly with ascending price, reaching levels above best-of-four sampling. For example, ascending price raises the pass rate over the default order by 25.5 points [18.5, 33.0] on

gpt-4.1-mini and 21.5 [14.0, 28.5] on gpt-4.1 and beats best-of-four sampling at a quarter of its cost. With two shared resources, the gap to the calculator tool shrinks from 20.0 and 15.0 points under the default order to 1.0 and 4.5 points under a weighted sum of price and duration, at a quarter of its cost. RQ3 explains this phenomenon; under ascending price the first listed feasible option is the cheapest one, whereas with a duration limit ascending price no longer helps (-1.5 and -6.0 points), because budget failures turn into duration failures. The checklist remains the strongest method at two to three times the cost of a single call, and a pipeline in which the model lists the feasible options and an exact solver picks the plan loses 3.5 to 9.0 points from $c = 0$ to $c = 1$, where the solver takes every infeasible option the model wrongly lists (Appendix C.3). Two limits remain; at $c = 0$ ascending order still adds 36.0 and 5.0 points, so part of the gain comes from easier price comparison and two registered predictions fail (Appendix C.3), and a program the model writes and plan-and-solve prompting reach comparable pass rates at a higher cost per task (Table 2). These results suggest that a single call on sorted options, when the sort reads every shared resource, can serve as a cost-effective alternative to sampling and to the calculator tool. One might question, however, whether sorting helps when a requirement the resource trades against is not a price, as on TravelPlanner, where ascending price lowers budget failures by 19.9 points and raises minimum-night failures by 8.7, paired over the 161 queries to which both checks apply, leaving the pass rate unchanged. In Appendix C.6, we address this with TravelPlanner runs showing that removing infeasible options before sorting and repairing with feasible alternatives can raise the pass rate of gpt-4.1 from 10.6% to 41.7% with one generation and one repair call, also when gpt-4.1 reads the constraint fields from the query text (Table 20).

5.2 Can temperature sampling match the gains of shuffled option orders?

We analyze the gain gap between best-of-four sampling over shuffled option orders and best-of-four sampling at temperature 1.0 across models and the number of shared resources. The gap is the difference in pass rate at equal calls. Our goal is to identify when this gap stays wide, as such regimes suggest temperature sampling cannot substitute for shuffled orders.

The gap stays wide when every sample reads the same order. As shown in Table 2, four samples that each see a different random order beat four temperature samples in all four cells, by 17.5 [10.5, 24.5], 16.0 [9.5, 22.5], 17.0 [10.0, 24.0] and 9.0 [2.0, 16.0] points at equal calls. As discussed in RQ3, adding temperature samples leaves the option order unchanged, resulting in only marginal performance differences between one attempt and four temperature samples.

The results here are obtained from the second environment, and we show the findings generalize to TravelPlanner in Appendix C.6, where four samples with shuffled reference tables contain a passing plan for 27.8% of the 180 queries against 16.7% for four temperature samples ($+11.1$ points [5.6, 16.7]). In regimes of coupled constraints or a fixed option order, four temperature samples provide limited gains over one attempt. However, these regimes coincide with the largest gains from shuffled orders. Thus, while temperature and shuffling both add samples, only shuffling changes the order the model reads. This reveals that scaling samples from one to four at temperature 1.0 fails to overcome position bias, with four samples providing limited improvement over one attempt in the identified regimes.

6 Conclusion

We study plan repair across coupling, the content of the repair message, and the order of the options, revealing that where a failure goes depends on their interactions. We find that, when models answer in one step, coupling primarily shapes the regressions that follow feedback, the repair message influences whether a fixed attempt breaks another constraint, and option order correlates with the chosen option in coupling-dependent patterns, in each of which the cheaper choice under coupling is the infeasible one. We examine the implications of these findings for plan repair in planning agents, identifying both opportunities and limitations (Appendix E). More calls may not be necessary, as a single call on sorted options can approach methods with more calls when the sort reads every shared resource. Our results also identify regimes where investing in more temperature samples yields limited benefit, such as when constraints are coupled or when every sample reads the options in the same order.

AI USE STATEMENT

We used generative AI tools in two ways: (1) to write and run the experiment and analysis code, including the task generators of the two synthetic environments; and (2) to edit grammar and style and to suggest alternative phrasings during manuscript preparation. All conceptual contributions, study design, and analysis were carried out by the authors. We have reviewed all AI-assisted work: an audit script checks every number in the paper against the logged model outputs. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work exclusively evaluates large language models on a generated planning environment and the publicly available TravelPlanner benchmark, whose constraints are checked by programs. All experiments involve automated evaluation of model outputs without human subject participation. The datasets used are a synthetic environment and an established research benchmark designed for travel-planning tasks. Our study aims to understand plan repair dynamics to improve the reliability and cost of LLM planning agents, posing no ethical concerns regarding privacy, harmful content generation, or potential misuse.

REPRODUCIBILITY STATEMENT

We provide comprehensive details to ensure reproducibility of our findings. All experiments use a generated environment, a publicly available benchmark and open-weight or commercial LLMs. We specify the model names, versions, environment settings and inference hyperparameters in Section 3. Complete prompt templates for all methods are provided in Appendix A. The mathematical formulations of all metrics, along with estimation procedures and aggregation methods, are formally defined and clearly described in Section 3.1 and Appendices C.4 and C.5. The supplementary material contains the environment generator, exhaustive solver and scorer, the runners with every prompt, the logs of all 33,600 synthetic attempts, of every sorting method and of the 900 re-graded TravelPlanner plans, an anonymised copy of the preregistration with its six amendments, and two audit scripts that check every number in the paper against the logged results without API calls.

REFERENCES

Abdelrahman Abouelenin, Atabak Ashfaq, Adam Atkinson, et al. Phi-4-mini technical report: Compact yet powerful multimodal language models via mixture-of-LoRAs. *arXiv preprint arXiv:2503.01743*, 2025.

Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society: Series B (Methodological)*, 57(1):289–300, 1995. doi: 10.1111/j.2517-6161.1995.tb02031.x.

Ethan Bito, Yongli Ren, and Estrid He. Evaluating position bias in large language model recommendations. arXiv preprint arXiv:2508.02020, 2025.

Meilin Chen, Zepeng Zhai, Jiaxuan Zhao, and Yuan Lu. SafeRun: Enabling determinism in LLM planning for running. arXiv preprint arXiv:2606.09027, 2026a.

Weiyi Chen et al. Traveleval: A simulation-based benchmark for globally optimized travel planning. *arXiv preprint arXiv:2606.01046*, 2026b.

Jihye Choi, Jinsung Yoon, Jiefeng Chen, Somesh Jha, and Tomas Pfister. ATLAS: Constraints-aware multi-agent collaboration for real-world travel planning. arXiv preprint arXiv:2509.25586, 2025.

P. C. Chu and J. E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4(1):63–86, 1998. doi: 10.1023/a:1009642405419.

Stefano DellaVigna, Devin Pope, and Eva Vivalt. Predict science to improve science. *Science*, 366 (6464):428–429, 2019. doi: 10.1126/science.aaz1704.

B. Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7, 1979. doi: 10.1214/aos/1176344552.

Hugh Everett. Generalized Lagrange multiplier method for solving problems of optimum allocation of resources. *Operations Research*, 11(3):399–417, 1963. doi: 10.1287/opre.11.3.399.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. *arXiv preprint arXiv:2211.10435*, 2022. doi: 10.48550/arXiv.2211.10435.

Yilun Hao, Yongchao Chen, Yang Zhang, and Chuchu Fan. Large language models can solve real-world planning rigorously with formal verification tools. *arXiv preprint arXiv:2404.11891*, 2024. doi: 10.48550/arXiv.2404.11891.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*, 2023. doi: 10.48550/arXiv.2310.01798.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023a.

Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. FollowBench: A multi-level fine-grained constraints following benchmark for large language models. *arXiv preprint arXiv:2310.20410*, 2023b. doi: 10.48550/arXiv.2310.20410.

Kumar Joag-Dev and Frank Proschan. Negative association of random variables with applications. *The Annals of Statistics*, 11(1), 1983. doi: 10.1214/aos/1176346079.

A. R. Jonckheere. A distribution-free k-sample test against ordered alternatives. *Biometrika*, 41 (1–2):133–145, 1954. doi: 10.1093/biomet/41.1-2.133.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. LLMs can't plan, but can help planning in LLM-modulo frameworks. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *PMLR*, 2024. arXiv:2402.01817.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. When can LLMs actually correct their own mistakes? a critical survey of self-correction of LLMs. *Transactions of the Association for Computational Linguistics*, 12:1417–1440, 2024. doi: 10.1162/tacl_a_00713.

Aaditya Khanal, Yangyang Tao, and Junxiu Zhou. Beyond pass@1: A reliability science framework for long-horizon LLM agents. *arXiv preprint arXiv:2603.29231*, 2026. doi: 10.48550/arXiv.2603.29231.

Tobias Kiecker, Eik Reichmann, Hosung Kang, Gabin An, and Lars Grunske. Is three the magic number? an empirical evaluation of LLM-based repair loops. arXiv preprint arXiv:2607.05197, 2026.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, 2023. arXiv:2309.06180.

Ming Liu. More is not always better: Cross-component interference in LLM agent scaffolding. *arXiv preprint arXiv:2605.05716*, 2026. doi: 10.48550/arXiv.2605.05716.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638.

Zhengdong Lu et al. Decompose, plan in parallel, and merge: A novel paradigm for large language models based planning with multiple constraints. *arXiv preprint arXiv:2506.02683*, 2025.

Xiaoliang Luo, Akilles Rechardt, Guangzhi Sun, Kevin K. Nejad, Felipe Yáñez, Bati Yilmaz, Kangjoo Lee, Alexandra O. Cohen, et al. Large language models surpass human experts in predicting neuroscience results. *Nature Human Behaviour*, 9(2):305–315, 2024. doi: 10.1038/s41562-024-02046-9.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023. doi: 10.48550/arXiv.2303.17651.

Nathan Mantel and William Haenszel. Statistical aspects of the analysis of data from retrospective studies of disease. *Journal of the National Cancer Institute*, 22(4):719–748, 1959. doi: 10.1093/jnci/22.4.719.

Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947. doi: 10.1007/BF02295996.

Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1–3):161–205, 1992. doi: 10.1016/0004-3702(92)90007-K.

Qiuyi Qi, Jinjian Zhang, Mutian Bao, Tian Liang, Guocong Li, Dongnan Liu, Wei Zhou, Jie Liu, Ming Kong, Linjian Mo, Feng Zhang, and Qiang Zhu. CARL: Constraint-aware reinforcement learning for planning with LLMs. *arXiv preprint arXiv:2607.04854*, 2026.

Libin Qiu, Yuhang Ye, Zhirong Gao, Xide Zou, Junfu Chen, Ziming Gui, Weizhi Huang, Xiaobo Xue, Wenkai Qiu, and Kun Zhao. Blueprint first, model second: A framework for deterministic LLM workflow. *arXiv preprint arXiv:2508.02721*, 2025. doi: 10.48550/arXiv.2508.02721.

Libin Qiu, Zhirong Gao, Junfu Chen, Yuhang Ye, Liangyu Li, Weizhi Huang, Xiaobo Xue, Wenkai Qiu, and Shuo Tang. AutoRefine: Compiling trajectories into validated typed agent artifacts. arXiv preprint arXiv:2601.22758, 2026.

Stephan Rabanser, Sayash Kapoor, Peter Kirgis, Kangheng Liu, Saiteja Utpala, and Arvind Narayanan. Towards a science of AI agent reliability. *arXiv preprint arXiv:2602.16666*, 2026. doi: 10.48550/arXiv.2602.16666.

Jie-Jing Shao, Bo-Wen Zhang, Xiao-Wen Yang, Baizhi Chen, Si-Yu Han, Jinghao Pang, Wen-Da Wei, Guohao Cai, Zhenhua Dong, Lan-Zhe Guo, and Yu-Feng Li. ChinaTravel: An open-ended travel planning benchmark with compositional constraint validation for language agents. In *International Conference on Learning Representations*, 2026. arXiv:2412.13682.

Weichun Shi, Minghao Liu, Wanting Zhang, Langchen Shi, Fuqi Jia, Feifei Ma, and Jian Zhang. ConstraintLLM: A neuro-symbolic framework for industrial-level constraint programming. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 16010–16030. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.809.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023. doi: 10.48550/arXiv.2303.11366.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks. *arXiv preprint arXiv:2402.08115*, 2024. doi: 10.48550/arXiv.2402.08115.

Raphael Tang, Xinyu Zhang, Xueguang Ma, Jimmy Lin, and Ferhan Ture. Found in the middle: Permutation self-consistency improves listwise ranking in large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics*, 2024. arXiv:2310.07712.

Darshan Tank and Baran Nama. The regression tax: Decomposing why skills help and hurt LLM agents. *arXiv preprint arXiv:2607.22520*, 2026.

Mariya I. Vasileva. Large language models can follow instructions, but not many at once: Phase transitions in compositional constraint satisfaction. *arXiv preprint arXiv:2608.12426*, 2026.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2609–2634. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-long.147.

Zexuan Wang, Chenghao Yang, Yingqi Que, Zhenzhu Yang, Huaqing Yuan, Yiwen Wang, Zhengxuan Jiang, Shengjie Fang, Zhenhe Wu, Zhaohui Wang, Zhixin Yao, Jiashuo Liu, Jincheng Ren, Yuzhen Li, Yang Yang, Jiaheng Liu, Jian Yang, Zaiyuan Wang, Ge Zhang, Zhoufutu Wen, and Wenhao Huang. WorldTravel: A realistic multimodal travel-planning benchmark with tightly coupled constraints. arXiv preprint arXiv:2602.08367, 2026.

Bosi Wen, Pei Ke, Xiaotao Gu, Lindong Wu, Hao Huang, Jinfeng Zhou, Wenchuang Li, Binxin Hu, Wendy Gao, Jiaxin Xu, et al. Benchmarking complex instruction-following with multiple constraints composition. *arXiv preprint arXiv:2407.03978*, 2024. doi: 10.48550/arXiv.2407.03978.

Zeqiu Wu et al. Fine-grained human feedback gives better rewards for language model training. *arXiv preprint arXiv:2306.01693*, 2023.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. TravelPlanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*, 2024. doi: 10.48550/arXiv.2402.01622.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. doi: 10.48550/arXiv.2505.09388.

Cong Zhang, Derrick Goh Xin Deik, Dexun Li, Hao Zhang, and Yong Liu. Planning with multi-constraints via collaborative language agents. *arXiv preprint arXiv:2405.16510*, 2024.

Xikai Zhang, Bo Wang, Likang Xiao, Yongzhi Li, Quan Chen, Wenjun Wu, and Liu Liu. IMAGINE: Integrating multi-agent system into one model for complex reasoning and planning. arXiv preprint arXiv:2510.14406, 2025.

Chujie Zheng, Hao Zhou, Fandong Meng, Jie Zhou, and Minlie Huang. Large language models are not robust multiple choice selectors. In *International Conference on Learning Representations*, 2024a. arXiv:2309.03882.

Huaixiu Steven Zheng, Swaroop Mishra, Hugh Zhang, Xinyun Chen, Minmin Chen, Azade Nova, Le Hou, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, and Denny Zhou. NATURAL PLAN: Benchmarking LLMs on natural language planning. *arXiv preprint arXiv:2406.04520*, 2024b. doi: 10.48550/arXiv.2406.04520.

Jinquan Zheng, Jia Yuan, Jiacheng Yao, Chenyang Gu, Pujun Zheng, and Guoxiu He. Mitigating selection bias in large language models via permutation-aware GRPO. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, 2026. arXiv:2603.21016.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023. doi: 10.48550/arXiv.2311.07911.

A PROMPT TEMPLATES

Budget planning. The system prompt is "You are a careful trip-planning assistant." The user prompt lists the requirements, the budget and one table per slot (ID, price, two attributes). Single generation ends with an instruction to reply with only a JSON object mapping each slot to an option ID. The calculator protocol lets the model reply CALC with a JSON plan, returns the total price, allows two uses and then asks for FINAL. The checklist instruction asks the model to name, for each requirement, the slot, the chosen option and whether it satisfies the requirement, then give the JSON object. Feedback returns lines such as "The dinner choice D8 does not satisfy the requirement: vegan = yes." or "The total price exceeds the budget." and asks for a revised JSON object. Each sub-agent sees only its slot's table and requirement and is asked to list every satisfying option with its price, cheapest first, without choosing; the aggregator is the same model at temperature 0, sees the full task and all reports, and writes the plan in one call with no rule on how to choose among the listed options. Sampling draws four plans at temperature 1.0 with fixed seeds, and the verifier keeps the first plan with six valid IDs and a total within budget, or the first plan if none qualifies. Calls cut off at the token cap are kept (12, 124, 55 and 75 calls for the four models, mostly checklists), and a final answer that cannot be parsed fails the format check. The second environment states each requirement in words with its own prompt (Appendix C.3), and the TravelPlanner runs use the benchmark's sole-planning prompt (Appendix C.6).

B ADDITIONAL PRELIMINARIES AND SETUP

B.1 PREREGISTRATION, AMENDMENTS AND DEVIATIONS

The analysis plan fixed three research questions before any analysis of the existing TravelPlanner logs and before any call in the synthetic environment. Table 3 lists every preregistered hypothesis with its outcome. Six amendments followed, each committed before the calls or analyses it affected. Amendment 1 replaced community-quantized open-weight models with the API models of the original plan and official bf16 open-weight models. Amendment 3 raised the output caps of the calculator (400 to 1,200 tokens) and of checklist and calculator plus checklist (1,200 to 2,000 tokens) after a 24-task cost probe showed truncated calculator calls; the probe logs were set aside. Amendments 4 to 6 changed the open-weight inference engine for installation and throughput reasons, without viewing main-run results; on 120 pilot prompts, vLLM and the static-batch engine gave identical text on 0.883 and identical check outcomes on 0.983 of prompts, with pass rates of 0.492 and 0.483.

Two analysis flaws were found after the results. First, the preregistered Jonckheere–Terpstra test (Jonckheere, 1954) for a decreasing variance ratio ranks the task-level quantity $(Y_i - \bar{Y})^2/\mathbb{E}_0[S^2]$, which takes two values in cells where attempts fail at most one check, so its ranks follow the cell pass rates. We report its results in Table 3 and base conclusions on direct bootstrap differences of the variance ratio. Second, the preregistered independence reference multiplies in precondition checks that would make an addressed check not applicable, although such attempts cannot enter the fixed attempts; removing them lowers the TravelPlanner reference for the budget verifier from 0.806 to 0.774 and changes no conclusion. The existing TravelPlanner logs lacked per-constraint fields for the planned 87-architecture analysis, so per-constraint analyses use the 900 sampled plans, and the cross-family hypothesis could not be computed. A second annotator for the checks addressed by TravelPlanner methods and the human forecast questionnaire were planned and not completed.

B.2 ENVIRONMENT AND DIFFICULTY MATCHING

Generation. An exhaustive search over the 8^6 plans confirms that every task is feasible, and every model reply is logged verbatim. Prices are distinct integers per slot drawn from lodging 90–320, breakfast 6–28, lunch 10–42, dinner 18–75, transport 12–85 and activity 15–110. Each constrained slot requires one of two attributes (for example pets allowed or free cancellation for lodging), the other attribute is a distractor drawn with probability 0.5, and a fixed number of options satisfy the requirement. At $c = 0.5$ three of the six constrained slots are coupled when $K = 6$; when $K = 3$ the tasks alternate between one and two coupled slots by task index, so 100 tasks of a cell have one and 100 have two. In a decoupled slot prices are assigned at random and the cheapest price is swapped onto a feasible option; in a coupled slot the highest prices go to the feasible options.

Table 3: Preregistered hypotheses and outcomes. RQ1 is measurement on TravelPlanner, RQ2 the cause in the synthetic environment and RQ3 prediction.

ID	Hypothesis	Outcome
H1.1	TravelPlanner failures are spread more evenly than both null models predict, and the share of single-check failures exceeds the permutation null	Not supported under the permutation null after the parser correction (0.974), holds under the null model that keeps task difficulty (Table 17); the single-check share exceeds the permutation null on the pooled plans
H1.2	Independent extraction nodes are not under-dispersed	Holds (Table 18)
H1.3	On TravelPlanner, the share of fixed attempts failing another check exceeds the independence and repeated-run references	Untestable on the logged runs (3 and 7 repaired queries); a later repair round leaves 63 of 108 fixed plans failing another check (App. C.2)
H1.4	Budget repairs cause failures on spending-related checks	Untestable for the budget verifier
H2.1	The single-attempt variance ratio falls with coupling	Falls on gpt-4.1-mini only (Table 9); the fallback regression finds no effect on either GPT model; the trend test is invalid
H2.2	Competing account, limited attention lowers the variance ratio at $c = 0$, $K = 6$	Not supported
H2.3	The share of fixed attempts failing another check rises with coupling	Feedback on gpt-4.1-mini and Qwen3-4B; not on Qwen3-8B; $p = 0.069$ on gpt-4.1
H2.4	Calculator and checklist interact negatively only under coupling	Not supported; point estimates negative at every level and dominated by ceilings
H2.5	After feedback this share is highest at $c = 1$	Holds on three models, not on Qwen3-8B
H2.6	Budget is more negatively associated with spending checks than with other checks	Direction as predicted, -0.264, $p = 0.238$
H3.1	The prediction from the theory of constraints beats all baselines	Not supported (Table 15)
H3.2	Its advantage over the share of violations grows with coupling	Direction as predicted on GPT, $p = 0.079$; not on Qwen
H3.3	A cost estimate computed by code makes sub-agent reports redundant	Same-run pass rate 0.244 against 0.189 after the evaluator correction, McNemar $p = 0.227$ (0.200 against 0.122, $p = 0.065$ before it)
H3.4	A selector gains little because candidates share budget verdicts	Holds, agreement 0.730 against 0.226

Table 4: Methods in the budget-planning environment. All methods share the task text and use greedy decoding except best-of-four sampling. Feedback reuses the single-generation plan as its first round.

Method	Change from single generation	Checks addressed T	Calls
Single generation	One call returns the plan as a JSON object	none	1
Feedback	The scorer lists the failed checks, for example "the total price exceeds the budget"; the model revises once	failed checks	1–2
Calculator tool	The model may ask for the total price of a proposed plan, at most twice	budget	1–3
Checklist	The model checks each requirement against its chosen option before answering	requirements	1
Both	Calculator tool and checklist together	both sets	1–3
Sub-agents	One sub-agent per requirement lists the feasible options with prices; an aggregator writes the plan	requirements	$K + 1$
Best-of-four sampling	Four plans at temperature 1.0; a verifier keeps the first plan within budget	budget	4

Unconstrained slots are priced at random. Tasks are rebuilt from their seeds, and pilot and main seeds do not overlap.

Matching. The pilot crossed coupling, K and four slack values (0.02, 0.08, 0.20, 0.40), plus two alternative numbers of feasible options at $c = 0$, with 60 tasks per setting and single generation only. For each K the rule chose parameters that minimise the largest pairwise pass-rate gap across coupling levels, with the $c = 1$ pass rate between 0.25 and 0.75. The largest gaps were 0.333 and 0.167 (gpt-4.1, $K = 3$ and 6), 0.150 and 0.067 (gpt-4.1-mini), 0.100 and 0.017 (Qwen3-8B) and 0.183 and 0.033 (Qwen3-4B). Simulated power for a 10-point interaction was below 0.5 with 120 tasks per cell for every model, so all main runs used 200. The count of feasible options is a matching knob only at $c = 0$ because the budget is set with slack above the cheapest plan that meets every requirement, and at $c = 0$ that plan is also the cheapest plan overall, so slack alone cannot lower p_0 there; the knob changes the share of feasible options at $c = 0$ only, a second difference between the levels that the difficulty controls of Appendix D.1 address.

B.3 Per-constraint sub-agents

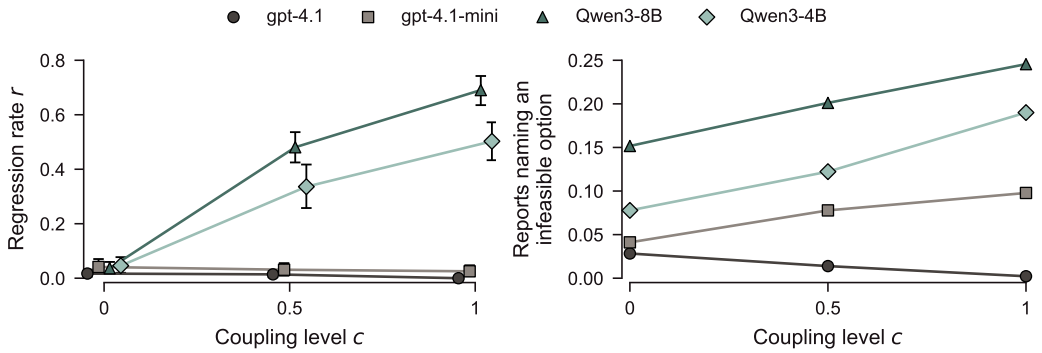

Figure 2: Per-constraint sub-agents. Left, regression rate r, the share of attempts that pass under single generation and fail with sub-agents, with 95% task-bootstrap intervals. Right, share of sub-agent reports that mention at least one infeasible option ID (1,800 reports per model and level). On both Qwen models, sub-agents mention infeasible options more often as coupling rises, and under coupling those options are the cheap ones the aggregator picks. Exploratory analysis.

Per-constraint sub-agents break passing attempts under coupling on the Qwen models. Figure 2 (left) shows the regression rate of the sub-agent condition. On gpt-4.1 and gpt-4.1-mini it stays at or below 4.1% at every coupling level. On Qwen3-8B it rises from 3.6% at $c = 0$ to 48.1% and 69.1% at $c = 0.5$ and $c = 1$, and on Qwen3-4B from 4.6% to 33.6% and 50.3%. Every Qwen3-8B attempt that regresses at $c \geq 0.5$ violates a requirement, and none exceeds the budget. In 98.6% of the violated slots at $c = 1$ the report of the sub-agent for that slot mentions the infeasible option that the aggregator chose (94.4% on Qwen3-4B). Qwen3-8B sub-agents mention an infeasible option in 15.2% of reports at $c = 0$ and 24.6% at $c = 1$ (Figure 2, right). At $c = 0$ the cheapest option of each slot is feasible, so an aggregator that favours cheap options still chooses a feasible one; under coupling the cheap options are exactly the infeasible ones that sub-agents wrongly list. This analysis was defined after the results were in.

B.4 Forecast elicitation

The questionnaire described both task families, the seven conditions, the variance ratio, the share of fixed attempts failing another check, the interaction and the four predictions of the gain, and asked for directions and point estimates with 80% intervals. gpt-6-astra, claude-fable-5-1 and claude-opus-5 answered five times each; all 15 replies parsed. On the TravelPlanner variance ratio, 15 of 15 answered larger than 1 (median 1.5, interquartile range 1.5 to 1.6; observed 0.974). For the synthetic environment at $c = 0$, 13 of 15 answered about 1; at $c = 1$, 13 of 15 answered smaller than 1. The median regression rate m on TravelPlanner was 0.50 (interquartile range 0.35 to 0.55). For the gpt-

4.1 calculator, the median regression rate m was 0.06 at $c = 0$ and 0.40 at $c = 1$ (observed 0 and 0). For the calculator and checklist interaction on gpt-4.1, 14 of 15 expected about zero at $c = 0$ and 13 of 15 a positive value at $c = 1$ (median $+5$ points; observed -9.5 and -37.8 points). All 15 named the prediction from the theory of constraints as the most accurate. The questionnaire was written after the pilot and the pooled TravelPlanner variance ratio had been seen, and its description of the predictions follows the framing of the study; the human questionnaire was not sent.

B.5 Costs and audit

The TravelPlanner fixes cost 33.16 USD (Appendix C.6). The three review questions run on 2026-09-20 cost 21.58 USD, plan-and-solve prompting and the pipeline with a solver 14.59 USD (Appendices C.2 and C.3), and the three runs that answer the review of 2026-09-21 3.75 USD (Appendices D.1, C.2 and C.6), and the runs of 2026-09-22 13.40 USD (Appendices D.1 and C.3, including the discarded first attempt), and the two runs of 2026-09-23 9.82 USD (Appendices D.1 and C.3), and the repair round on samples 2 to 5 of TravelPlanner 23.88 USD (Appendix C.2). The sorting study cost 50.91 USD in API calls, of which 1.89 USD went to the discarded unpaired pilot. The API calls of the main study cost 59.59 USD in total, including the pilot, a discarded cost probe, the main runs of the two GPT models, the same-run TravelPlanner comparison and the forecasts. Open-weight runs used one rented RTX 5090. The script `verify_paper_numbers_2026-09-17.py` recomputes the numbers in this paper from the saved result files and the logs.

C Extended analysis and additional results

C.1 Regression rates and dispersion by model

Tables 7, 8 and 9 give, for every model, the share m of fixed attempts that fail another check for every method, the regression rate r of every method and the single-attempt variance ratio, each by coupling level. The preregistered trend test over the three levels gives $p = 0.0006$ for gpt-4.1-mini and $p = 0.0002$ for Qwen3-4B (corrected $q = 0.0021$ and 0.0007) and $p = 0.069$ for gpt-4.1.

Calculator calls. The calculator tool may be called twice per attempt. gpt-4.1 calls it once in 1,081 of its 1,200 attempts and twice in 119; gpt-4.1-mini once in 684 and twice in 494, and its 22 attempts without a call all fail; Qwen3-8B twice in 1,179; Qwen3-4B does not call it in 1,193 of 1,200 attempts, so its calculator condition is in effect a single generation under the calculator prompt, which is why its regressions with the calculator (Section 4.1) do not fall. The amendment that raised the token limit of a calculator turn concerns turns cut off at the limit, which happened in 70 gpt-4.1-mini attempts and in none of the other models' (`coupling_review6_fixes_2026-09-23.py`).

What single attempts fail. Table 5 gives the share of single attempts that violate any requirement and the share that exceed the budget in each cell. Table 6 gives the pass rate of every condition in every cell.

Table 5: Single generation, share of attempts violating any requirement / share exceeding the budget, per cell.

Model	c=0, K=3	c=0, K=6	c=0.5, K=3	c=0.5, K=6	c=1, K=3	c=1, K=6
gpt-4.1	0.000 / 0.205	0.000 / 0.035	0.005 / 0.440	0.000 / 0.135	0.005 / 0.635	0.005 / 0.180
gpt-4.1-mini	0.000 / 0.520	0.000 / 0.510	0.025 / 0.360	0.000 / 0.490	0.040 / 0.295	0.005 / 0.475
Qwen3-8B	0.005 / 0.200	0.040 / 0.395	0.120 / 0.110	0.305 / 0.045	0.185 / 0.080	0.410 / 0.000
Qwen3-4B	0.005 / 0.455	0.010 / 0.560	0.015 / 0.605	0.025 / 0.705	0.030 / 0.420	0.020 / 0.605

C.2 Multi-round repair, feedback content and two TravelPlanner runs

These runs were added in answer to review, after the results of the main study were known. They reuse the saved first attempts, the scorer and the prompts, and every reply is logged verbatim (`binding_multiround_2026-09-18.py`, `binding_tp_runs_2026-09-18.py`, analysis in `binding_followup_analysis_2026-09-18.py`).

Table 6: Pass rate of every condition in every cell (200 tasks each).

Model	Method	c=0, K=3	c=0, K=6	c=0.5, K=3	c=0.5, K=6	c=1, K=3	c=1, K=6
gpt-4.1	single	0.795	0.965	0.555	0.865	0.360	0.815
	feedback	0.885	0.985	0.665	0.930	0.530	0.880
	calculator	0.975	0.995	0.985	1.000	0.975	1.000
	checklist	0.955	0.995	0.960	1.000	0.950	1.000
	both	0.970	1.000	0.985	1.000	0.995	1.000
	sub-agents	0.830	1.000	0.830	0.990	0.890	0.995
	sampling	0.815	0.980	0.615	0.925	0.460	0.875
gpt-4.1-mini	single	0.480	0.490	0.615	0.510	0.665	0.520
	feedback	0.705	0.835	0.870	0.850	0.860	0.865
	calculator	0.860	1.000	0.925	0.980	0.930	0.955
	checklist	0.850	0.980	0.965	1.000	0.970	1.000
	both	0.985	1.000	0.990	1.000	0.990	0.990
	sub-agents	0.640	0.995	0.900	0.975	0.915	0.975
	sampling	0.535	0.590	0.710	0.610	0.725	0.605
Qwen3-8B	single	0.800	0.575	0.770	0.655	0.735	0.590
	feedback	0.905	0.870	0.925	0.965	0.960	0.970
	calculator	0.880	0.805	0.840	0.695	0.785	0.685
	checklist	0.855	0.810	0.840	0.715	0.865	0.630
	both	0.885	0.805	0.870	0.755	0.850	0.700
	sub-agents	0.940	0.915	0.600	0.405	0.420	0.170
	sampling	0.880	0.665	0.760	0.675	0.760	0.615
Qwen3-4B	single	0.540	0.440	0.380	0.275	0.565	0.380
	feedback	0.895	0.815	0.680	0.430	0.760	0.510
	calculator	0.570	0.440	0.425	0.320	0.560	0.395
	checklist	0.820	0.775	0.675	0.485	0.735	0.540
	both	0.645	0.425	0.460	0.305	0.615	0.450
	sub-agents	0.755	0.880	0.440	0.420	0.450	0.260
	sampling	0.615	0.500	0.435	0.315	0.625	0.440

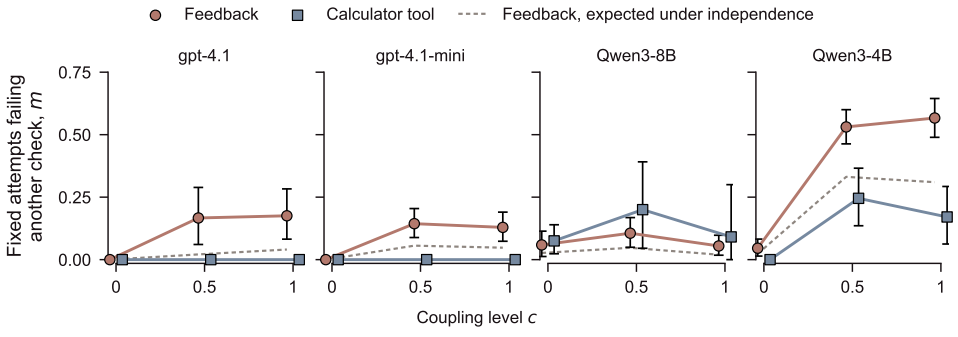

Figure 3: Share m of fixed attempts that fail another check, after one round of feedback (circles) and with a calculator tool (squares), by coupling level c. The dashed line is the share expected for feedback under independence. Bars are 95% task-bootstrap intervals over 400 tasks per level.

Table 7: Share m of fixed attempts that fail another check, with 95% task-bootstrap interval, followed by the number of fixed attempts S and the value m^{ind} expected under independence, pooled over K.

Model	Method	$c = 0$	$c = 0.5$	$c = 1$
gpt-4.1	feedback	0.000 [0.000, 0.000] (22; 0.000)	0.167 [0.061, 0.289] (42; 0.021)	0.175 [0.082, 0.283] (57; 0.040)
	calculator	0.000 [0.000, 0.000] (42; 0.000)	0.000 [0.000, 0.000] (112; 0.000)	0.000 [0.000, 0.000] (158; 0.000)
	checklist	$S = 0$	0.000 [0.000, 0.000] (1; 0.020)	0.000 [0.000, 0.000] (2; 0.025)
	both	0.000 [0.000, 0.000] (43; 0.000)	0.000 [0.000, 0.000] (113; 0.002)	0.000 [0.000, 0.000] (164; 0.002)
	sub-agents	$S = 0$	0.000 [0.000, 0.000] (1; 0.090)	0.000 [0.000, 0.000] (2; 0.057)
	sampling	0.000 [0.000, 0.000] (11; 0.000)	0.000 [0.000, 0.000] (24; 0.000)	0.000 [0.000, 0.000] (33; 0.002)
gpt-4.1-mini	feedback	0.000 [0.000, 0.000] (114; 0.000)	0.144 [0.088, 0.204] (139; 0.056)	0.129 [0.074, 0.190] (124; 0.048)
	calculator	0.000 [0.000, 0.000] (184; 0.070)	0.000 [0.000, 0.000] (161; 0.047)	0.000 [0.000, 0.000] (144; 0.057)
	checklist	$S = 0$	0.000 [0.000, 0.000] (4; 0.017)	0.000 [0.000, 0.000] (9; 0.015)
	both	0.000 [0.000, 0.000] (204; 0.007)	0.000 [0.000, 0.000] (173; 0.005)	0.000 [0.000, 0.000] (161; 0.010)
	sub-agents	$S = 0$	0.000 [0.000, 0.000] (5; 0.047)	0.000 [0.000, 0.000] (8; 0.022)
	sampling	0.000 [0.000, 0.000] (34; 0.000)	0.061 [0.000, 0.139] (49; 0.027)	0.065 [0.000, 0.161] (31; 0.022)
Qwen3-8B	feedback	0.059 [0.013, 0.114] (85; 0.027)	0.106 [0.050, 0.168] (104; 0.047)	0.055 [0.018, 0.098] (128; 0.020)
	calculator	0.075 [0.024, 0.140] (80; 0.063)	0.200 [0.045, 0.391] (20; 0.169)	0.091 [0.000, 0.300] (11; 0.152)
	checklist	0.000 [0.000, 0.000] (2; 0.145)	0.027 [0.000, 0.091] (37; 0.057)	0.023 [0.000, 0.078] (43; 0.025)
	both	0.000 [0.000, 0.000] (78; 0.005)	0.000 [0.000, 0.000] (58; 0.000)	0.000 [0.000, 0.000] (61; 0.002)
	sub-agents	0.000 [0.000, 0.000] (6; 0.055)	0.000 [0.000, 0.000] (35; 0.012)	0.000 [0.000, 0.000] (33; 0.005)
	sampling	0.059 [0.000, 0.150] (34; 0.021)	0.000 [0.000, 0.000] (4; 0.248)	0.091 [0.000, 0.300] (11; 0.145)
Qwen3-4B	feedback	0.046 [0.014, 0.082] (153; 0.023)	0.531 [0.463, 0.600] (194; 0.331)	0.567 [0.489, 0.645] (150; 0.310)
	calculator	0.000 [0.000, 0.000] (41; 0.005)	0.245 [0.136, 0.366] (53; 0.049)	0.171 [0.062, 0.293] (41; 0.070)
	checklist	$S = 0$	0.000 [0.000, 0.000] (5; 0.238)	0.333 [0.000, 1.000] (3; 0.220)
	both	0.000 [0.000, 0.000] (50; 0.002)	0.000 [0.000, 0.000] (43; 0.000)	0.000 [0.000, 0.000] (45; 0.000)
	sub-agents	0.000 [0.000, 0.000] (1; 0.180)	0.000 [0.000, 0.000] (4; 0.312)	0.000 [0.000, 0.000] (3; 0.220)
	sampling	0.000 [0.000, 0.000] (27; 0.007)	0.048 [0.000, 0.158] (21; 0.025)	0.000 [0.000, 0.000] (26; 0.027)

Table 8: Regression rate r with 95% task-bootstrap interval and counts (regressions / passing under single generation), pooled over K. Feedback revises only attempts that fail, so an attempt that passes under single generation is returned unchanged and its r is 0 by construction; the row is omitted.

Model	Method	$c = 0$	$c = 0.5$	$c = 1$
gpt-4.1	calculator	0.000 [0.000, 0.000] (0/352)	0.000 [0.000, 0.000] (0/284)	0.000 [0.000, 0.000] (0/235)
	checklist	0.000 [0.000, 0.000] (0/352)	0.004 [0.000, 0.011] (1/284)	0.000 [0.000, 0.000] (0/235)
	both	0.003 [0.000, 0.009] (1/352)	0.000 [0.000, 0.000] (0/284)	0.000 [0.000, 0.000] (0/235)
	sub-agents	0.017 [0.006, 0.031] (6/352)	0.014 [0.003, 0.029] (4/284)	0.000 [0.000, 0.000] (0/235)
	sampling	0.011 [0.003, 0.023] (4/352)	0.004 [0.000, 0.011] (1/284)	0.009 [0.000, 0.022] (2/235)
gpt-4.1-mini	calculator	0.031 [0.010, 0.057] (6/194)	0.044 [0.020, 0.073] (10/225)	0.055 [0.028, 0.085] (13/237)
	checklist	0.052 [0.022, 0.084] (10/194)	0.013 [0.000, 0.030] (3/225)	0.000 [0.000, 0.000] (0/237)
	both	0.005 [0.000, 0.017] (1/194)	0.000 [0.000, 0.000] (0/225)	0.008 [0.000, 0.022] (2/237)
	sub-agents	0.041 [0.016, 0.070] (8/194)	0.031 [0.009, 0.055] (7/225)	0.025 [0.008, 0.047] (6/237)
	sampling	0.015 [0.000, 0.036] (3/194)	0.031 [0.009, 0.057] (7/225)	0.013 [0.000, 0.029] (3/237)
Qwen3-8B	calculator	0.051 [0.026, 0.078] (14/275)	0.102 [0.069, 0.138] (29/285)	0.068 [0.039, 0.100] (18/265)
	checklist	0.080 [0.050, 0.114] (22/275)	0.088 [0.056, 0.122] (25/285)	0.079 [0.048, 0.113] (21/265)
	both	0.055 [0.029, 0.082] (15/275)	0.063 [0.036, 0.093] (18/285)	0.060 [0.034, 0.091] (16/265)
	sub-agents	0.036 [0.016, 0.060] (10/275)	0.481 [0.425, 0.536] (137/285)	0.691 [0.635, 0.742] (183/265)
	sampling	0.004 [0.000, 0.011] (1/275)	0.025 [0.007, 0.043] (7/285)	0.057 [0.030, 0.086] (15/265)
Qwen3-4B	calculator	0.179 [0.126, 0.234] (35/196)	0.191 [0.125, 0.259] (25/131)	0.180 [0.128, 0.237] (34/189)
	checklist	0.071 [0.038, 0.110] (14/196)	0.099 [0.052, 0.154] (13/131)	0.127 [0.082, 0.176] (24/189)
	both	0.163 [0.114, 0.216] (32/196)	0.160 [0.100, 0.226] (21/131)	0.111 [0.068, 0.158] (21/189)
	sub-agents	0.046 [0.020, 0.077] (9/196)	0.336 [0.258, 0.417] (44/131)	0.503 [0.433, 0.572] (95/189)
	sampling	0.000 [0.000, 0.000] (0/196)	0.015 [0.000, 0.040] (2/131)	0.016 [0.000, 0.036] (3/189)

Table 9: Single-attempt variance ratio under the permutation null by coupling level (pooled over K), direct bootstrap differences between levels (exploratory), and the coupling coefficient of the preregistered fallback regression over pilot settings.

Model	$c = 0$	$c = 0.5$	$c = 1$	c=1 minus c=0	c=0.5 minus c=0	Pilot c coefficient
gpt-4.1	1.000 [1.000, 1.000]	0.989 [0.967, 1.000]	0.983 [0.955, 1.000]	−0.017 [−0.046, 0.000]	−0.011 [−0.033, 0.000]	−0.002 [−0.039, 0.048]
gpt-4.1-mini	1.000 [1.000, 1.000]	0.964 [0.933, 0.992]	0.943 [0.907, 0.976]	−0.057 [−0.094, −0.023]	−0.036 [−0.067, −0.008]	−0.009 [−0.031, 0.015]
Qwen3-8B	0.991 [0.933, 1.056]	0.904 [0.821, 0.995]	1.052 [0.913, 1.231]	0.061 [−0.095, 0.247]	−0.087 [−0.192, 0.023]	−0.315 [−0.409, −0.218]
Qwen3-4B	1.008 [0.981, 1.036]	0.911 [0.844, 0.973]	0.982 [0.924, 1.041]	−0.026 [−0.089, 0.039]	−0.097 [−0.171, −0.027]	−0.346 [−0.478, −0.069]

Rounds and message content. Table 10 gives both GPT models and Qwen3-4B, served from the official weights as in the main study. The conversation keeps every earlier round, the temperature is 0 and a loop stops once the plan passes. Most of the gain comes in the first rounds, as in repair loops for code (Kiecker et al., 2026). For gpt-4.1-mini at $c = 1$ the first-round repair rate moves by -5.5 points with the overspend, $+11.0$ with the prices and $+16.0$ with the savings. The message with savings names, for each slot, how much the cheapest option meeting the slot's requirement would save, which the program can compute here because the attributes are structured. The first round repairs 55 of the 165 failing gpt-4.1 attempts at $c = 1$ against 57 in the main run, which used the same attempts and temperature; the difference comes from re-running the call through the API. Qwen3-8B was not run in this study.

Table 10: Three rounds of feedback from the saved first attempts. Pass is the share of failing first attempts that pass after each round, in per cent. Failing another check gives the fixed attempts that fail another check over all fixed attempts, after each round. Cycles is the share of attempts whose set of failed checks returns to an earlier set after leaving it, in per cent. The message variants are checks, which lists the failed checks; overspend, which adds the overspend; prices, which adds the price of every chosen option; savings, which adds the saving available in each slot; and slots, which adds to checks only the names of the slots that hold a cheaper option meeting the slot's requirement, with no number (GPT models only).

Model	c	Message	n	Pass after each of three rounds			Failing another check after each round			Cycles
				Round 1	Round 2	Round 3	Round 1	Round 2	Round 3	
gpt-4.1	0	checks	48	45.8	62.5	72.9	0/22	0/30	0/35	0.0
		overspend	48	33.3	54.2	62.5	0/16	0/26	0/30	0.0
		prices	48	64.6	72.9	79.2	0/31	0/35	0/38	0.0
		savings	48	75.0	87.5	93.8	0/36	0/42	0/45	0.0
		slots	48	62.5	68.8	72.9	0/30	0/33	0/35	0.0
	1	checks	165	28.5	37.6	47.9	8/55	2/64	1/80	6.1
		overspend	165	25.5	35.8	45.5	6/48	3/62	3/78	4.8
		prices	165	42.4	55.8	61.8	8/78	4/96	2/104	4.8
		savings	165	64.8	83.0	92.7	5/112	0/137	1/154	1.8
		slots	165	63.0	78.8	85.5	0/104	0/130	0/141	0.0
gpt-4.1-mini	0	checks	206	55.8	77.7	86.4	0/115	0/160	0/178	0.5
		overspend	206	49.0	68.4	82.5	0/101	0/141	0/170	0.0
		prices	206	46.1	75.2	85.9	0/95	0/155	0/177	0.0
		savings	206	70.9	91.7	95.1	1/147	0/189	0/196	0.5
		slots	206	77.2	90.3	92.2	1/160	0/186	0/190	0.5
	1	checks	163	68.1	78.5	83.4	14/125	11/139	6/142	8.6
		overspend	163	62.6	71.8	81.0	11/113	13/130	11/143	6.7
		prices	163	79.1	87.7	95.7	9/138	5/148	2/158	1.8
		savings	163	84.0	95.1	96.9	9/146	3/158	1/159	0.0
		slots	163	84.7	93.3	95.7	6/144	2/154	0/156	1.8
Qwen3-4B	0	checks	204	74.0	82.4	86.8	6/157	1/169	2/179	4.4
		overspend	204	73.5	85.3	89.7	6/156	2/176	0/183	4.4
		prices	204	71.6	87.7	93.6	8/154	3/182	1/192	7.4
		savings	204	78.9	90.2	91.7	7/168	1/185	1/188	4.4
	1	checks	211	29.4	53.1	55.9	89/151	12/124	22/140	35.1
		overspend	211	34.6	54.5	57.8	57/130	27/142	21/143	33.2
		prices	211	41.2	65.4	74.4	79/166	16/154	28/185	26.1
		savings	211	41.2	66.8	72.5	82/169	13/154	24/177	25.1

Naming the slots without a number. A review asked whether the message with savings helps because it says where the plan can change or because it gives numbers. A fifth message therefore adds to the failed checks only the names of the slots that hold a cheaper option meeting the slot's

requirement (`coupling_review3_runs_2026-09-21.py`, three predictions fixed before the first paid call; rows "slots" of Table 10). At $c = 1$ it raises the first-round repair rate over the failed checks alone by $+34.5$ points [26.7, 42.4] on gpt-4.1 and $+16.6$ [10.4, 23.3] on gpt-4.1-mini, and it differs from the message with savings by -1.8 [-9.1, 4.8] and $+0.6$ [-4.9, 6.1]. No fixed gpt-4.1 attempt fails another check in any round (0 of 104, 130 and 141), against 8 of 55 in the first round with the failed checks alone, and 6 of 144 do in the first round on gpt-4.1-mini, against 14 of 125. Naming the slots therefore accounts for the first-round gain of the message with savings under coupling. The numbers matter in later rounds and without coupling: after three rounds gpt-4.1 reaches 85.5% with the slots and 92.7% with the savings at $c = 1$, and 72.9% against 93.8% at $c = 0$ ($n = 48$). Two of the three predictions hold; we expected the slots to stay below the savings in the first round on both models, which fails on gpt-4.1-mini. The run cost 0.98 USD.

Plan-and-solve prompting. A review asked whether the regressions come from answering in one step rather than from the task. We therefore ran plan-and-solve prompting (Wang et al., 2023) on the two GPT models at $c = 0$ and $c = 1$ (`coupling_pat_runs_2026-09-20.py`, four predictions fixed before the first paid call), once as the single generation on the 800 main tasks of each model and once as the revision of the saved failing first attempts, after the same message as the feedback method. As a single generation it passes 0.975 and 0.960 of the tasks on gpt-4.1-mini and 1.000 and 0.998 on gpt-4.1 (Table 11), $+36.8$ [31.8, 41.8] and $+41.0$ [36.2, 45.8] points above single generation at $c = 1$, and the position bias disappears: at $c = 1$ the two models take the first listed feasible option in 2.6% and 1.7% of the slots where it is not the cheapest feasible one, against 73.8% and 50.2%, and the cheapest feasible one in 95.9% and 95.8%. As a revision it fixes 162 of the 163 failing attempts of gpt-4.1-mini and 157 of the 165 of gpt-4.1 at $c = 1$, where feedback fixes 124 and 57, and no fixed attempt fails another check at either level; the pass rate after the revision is 0.994 and 0.952 against 0.663 and 0.285. Two of the four predictions fail: we expected the position bias and the regressions to survive plan-and-solve prompting, and they do not. The regressions of Section 4.1 therefore belong to a revision written in one step; a model that writes out the prices before it answers compares them, as it does with the calculator tool and the checklist. The price is length. A plan-and-solve reply has 754 to 944 tokens against 46, which makes a task 4.1 to 4.9 times as expensive as a single generation, 26 of the 800 gpt-4.1-mini replies ran into the limit of 2,000 tokens and returned no plan, and in 7 of its 213 revisions gpt-4.1 wrongly concluded that no plan fits the budget and returned none. These runs cost 1.88 USD on gpt-4.1-mini and 7.94 USD on gpt-4.1.

Table 11: Plan-and-solve prompting (PS) as the single generation (800 tasks per model) and as the revision of the saved failing first attempts, against single generation and the feedback method on the same tasks and attempts. First listed is the share of choices that take the first listed feasible option in constrained slots where it is not the cheapest feasible one. Fixed counts the attempts whose failed checks all pass after the revision, with those that then fail another check in parentheses.

Model	c	Pass rate single / PS	First listed single / PS	Failing first attempts	Fixed (failing another) feedback / PS	Pass after revision feedback / PS
gpt-4.1-mini	0	0.485 / 0.975	0.116 / 0.000	206	114 (0) / 195 (0)	0.553 / 0.947
	1	0.593 / 0.960	0.738 / 0.026	163	124 (16) / 162 (0)	0.663 / 0.994
gpt-4.1	0	0.880 / 1.000	0.006 / 0.023	48	22 (0) / 39 (0)	0.458 / 0.812
	1	0.588 / 0.998	0.502 / 0.017	165	57 (10) / 157 (0)	0.285 / 0.952

One repair round on TravelPlanner. The logs of earlier runs fix too few plans to estimate m (3 and 7 fixed queries), so a repair round was run for this purpose. The first sample of each of the 180 validation queries is the baseline plan, and the 157 failing plans are revised once by gpt-4.1 at temperature 0. The message lists the failed checks in words, and the second variant also gives the total cost, the budget and the overspend. New failures after the revision fall mostly on the city route, the budget and the minimum nights (11, 10 and 9 plans with the bare message). The run cost 6.01 USD. The same round was later run on samples 2 to 5 of every query (`binding_tp_runs_2026-09-18.py repair_more`), graded with the corrected evaluator, with three predictions fixed before its first paid call. Of 645 failing plans, the message with the failed checks fixes 93, of which 55 fail another check (59.1% [49.0, 68.6]), so M1 (at least 40%)

holds; the fixed plans fail another check in 7 of 11 coupled and 31 of 47 uncoupled queries, so M2 (higher on coupled queries) fails; and 68.7% of the checks they newly fail are the budget, the minimum nights or the city route, so M3 (at least half) holds. Pooled with the first sample, the message with the failed checks leaves 63 of 108 fixed plans failing another check, 58.3% [48.9, 67.2], and the message with the overspend 71 of 116. The round cost 23.88 USD against an approved 23.8, as six requests were in flight when the cap was reached.

Price-ordered reference information. The accommodation, restaurant and flight tables of each query are sorted by their price column, names unchanged, and one plan per query is sampled at temperature 1.0 in each of three arms. Budget failures are 35.5% in the default order, 15.4% ascending and 74.4% descending, with mean costs of $4416, $3515 and $6170. Minimum-night failures are 47.3%, 55.6% and 41.5%, and final pass rates 10.6%, 10.0% and 3.3%. Descending order raises budget failures by 38.6 points [30.4, 46.8] and lowers the final pass rate by 7.2 points [2.8, 12.2]. The order of a list therefore decides what a plan costs, and the cheaper plans meet the budget while more of them break the minimum stay of a cheap accommodation.

C.3 Sorting study: environment, predictions and Qwen3-4B

Environment. The second environment (`coupling_text_env_2026-09-18.py`) keeps the six slots, the price ranges and the meaning of c of Appendix B.2, adds a duration in minutes to every option, drawn from lodging 20–60, breakfast 10–40, lunch 20–70, dinner 30–95, transport 15–80 and activity 40–150, and offers 12 options per slot. Each attribute has a requirement sentence and several option phrasings for and against it that avoid the attribute's name, for example "four-legged guests can stay" against "no animals on the premises". The phrasings are assembled from templates, so a program given the phrase lists could still classify them. The budget is placed so that a fixed share of the plans meeting every requirement fits it, which the generator checks by exhaustive search; the measured share is 0.0496 to 0.0563 across settings, with no model in the loop. When the first feasible option of a shuffled order is not the cheapest one, gpt-4.1-mini takes the first in 56.0% of 2,376 choices and the cheapest in 23.6%, against 33.3% by chance, and gpt-4.1 takes them in 38.6% and 41.0%. In the mode with two resources the budget and the duration limit are set together: a bisection finds the quantile u at which the u-quantile of total price and the u-quantile of total duration, over the plans that meet every requirement, leave 5% of those plans within both limits. At $c = 1$ with six constrained slots, no sort by one resource meets both limits in any of the 40 tasks per cell of the generator's self-test. Every presentation mode relabels the option IDs by displayed position, and the task seed depends only on the coupling level and the task index, so all arms of a model see the same tasks. Qwen3-8B is left out of this environment because its failures in the first one come mainly from the requirements (Appendix D.1), which sorting does not address. A first gpt-4.1-mini pilot drew different tasks per arm; its log is kept under its own name and enters no table.

Predictions and verdicts. The runner fixed four predictions before its first paid call, with point values from a simulation in which the model follows the position bias exactly. With one resource at $c = 1$ the simulation gives 0.480 for the default order, 1.000 for ascending and 0.350 for descending; with two resources it gives 0.140 for the default order, 0.050 for ascending price and 0.435 for the sort by the weighted sum. Table 12 gives the verdicts; H4 is tested only against ascending price, because ascending duration was not run. The threshold of H1 is stated two ways; the method report set it at 15 points, and the runner's description says half of the estimated effect. Against half of the simulated 52 points, neither GPT gain reaches the threshold. Descending price helps gpt-4.1 because it compares prices more than the other models: under that order the first listed feasible option is the most expensive, and gpt-4.1 takes the cheapest feasible option in 57.0% of the constrained slots where the two differ and the first listed one in 12.8%, against 41.0% and 38.6% in the default order, so its budget failures fall from 91 to 73 of 200 tasks, while gpt-4.1-mini, which follows position, loses 15.5 points.

Proposition 1 (Position bias under ascending order)**.** *Let the total price be the sum of slot prices and let every requirement concern one slot. If the model follows this position bias and judges feasibility correctly, then under ascending price order it returns the cheapest plan that meets every requirement, which fits the budget whenever any such plan does.*

Proof of Proposition 1.

Table 12: Registered predictions of the sorting study and their outcomes, as differences in pass rate in points with 95% task-bootstrap intervals (200 paired tasks).

ID	Prediction	Outcome
H1	Ascending beats the default order at $c = 1$, one resource	gpt-4.1-mini $+25.5$ [18.5, 33.0], gpt-4.1 $+21.5$ [14.0, 28.5], Qwen3-4B $+2.5$ [-3.5, 8.0]
H2	Descending falls below the default order at $c = 1$	gpt-4.1-mini -15.5 [-22.5, -8.5], Qwen3-4B -5.5 [-10.0, -1.0]; fails on gpt-4.1, $+8.0$ [0.5, 16.0]
H3	Ascending and the default order differ by at most 3 points at $c = 0$	Falsified, $+36.0$ [28.5, 43.5], $+5.0$ [2.5, 8.5] and $+7.5$ [-0.5, 15.5] with one resource; $+38.0$, -0.5 and -6.0 with two
H4	With two resources the sort by the weighted sum beats every order by one resource at $c = 1$	Over ascending price $+22.5$ [16.0, 29.5] and $+16.5$ [9.0, 24.0] on the GPT models, $+4.0$ [-0.5, 8.5] on Qwen3-4B; ascending duration was not run

Proof. Under ascending order the first feasible option of a slot is its cheapest feasible option. Price and requirements separate by slot, so the plan of per-slot minima has the lowest total price among plans that meet every requirement. □

When feasibility is misjudged. Proposition 1 assumes that the model judges feasibility correctly, and the logs show in which direction it errs. Under ascending price at $c = 1$ with one resource, gpt-4.1-mini and gpt-4.1 take the cheapest feasible option in 59.3% and 64.7% of the 900 constrained slots, a dearer feasible one in 40.3% and 35.3%, and an infeasible one in 0.3% and 0.0%, although under coupling every infeasible option is listed before the first feasible one. Their failures under sorting are therefore budget failures (83 and 52 of 200 tasks) and not requirement failures (3 and 0): the models pass over cheap feasible options, more often than the 10.8% and 3.9% of feasible options they fail to list when asked only to label them. Qwen3-4B errs in both directions, taking an infeasible option in 17.4% of slots, which is why sorting does not help it (104 of its 200 tasks fail a requirement).

Sensitivity to λ. In the simulation of the position bias with two resources at $c = 1$, the sort by the weighted sum meets both limits in 0.435 of tasks at the public ratio, 0.275 and 0.315 at half and twice that ratio, and 0.075 and 0.240 at a quarter and four times; the ratio of the two limits is close to the best fixed choice, and a per-task search over λ was not run.

Checklist on sorted options. With two resources, running the checklist on sorted options adds 9.0 points [1.5, 16.5] and 7.0 [1.5, 12.5] on the GPT models while the cost per task falls by 5% and 13%.

Majority vote. A per-slot majority vote over the shuffled samples, the aggregation of permutation self-consistency (Tang et al., 2024), reaches 0.605 and 0.310 with one resource against 0.475 and 0.290 for a vote over temperature samples, below the 0.775 and 0.540 of keeping the first sample whose totals meet the limits.

Qwen3-4B. Table 13 gives the pass rates of Qwen3-4B, served from the official weights on vLLM with thinking disabled. Qwen3-4B stays low in every method, at most 0.135 with one resource and 0.080 with two, except the checklist on sorted options, which reaches 0.225 against 0.125 for the checklist ($+10.0$ points [3.0, 17.0]); it misjudges from text whether an option meets a requirement too often for sorting alone to help.

A pipeline with a solver. A review also asked how sorting compares with architectures built to handle constraints, in which a model reads the task and a solver enforces the constraints (Hao et al., 2024; Shi et al., 2025; Chen et al., 2026a). In this environment only the model can judge a requirement, so we let the model list, for every slot with a requirement, the options that meet it, and a program take the cheapest listed option of every slot or, with a duration limit, the cheapest plan within both limits (one call per task, default order, the seeds of Table 2, three predictions fixed before the first paid call). With the true labels the program passes every task. With the models' labels it

Table 13: Qwen3-4B in the second environment at $c = 1$, pass rate over 200 paired tasks per cell.

Method	Qwen3-4B, one resource	Qwen3-4B, two resources
Default order	0.095	0.060
Ascending price	0.120	0.030
Descending price	0.040	–
Weighted sum	–	0.070
Calculator	0.100	0.045
Checklist	0.125	0.050
Checklist, sorted options	0.225	0.075
Best-of-4, temperature 1.0	0.125	0.065
Best-of-4, shuffled orders	0.135	0.080

passes 0.885 and 0.955 of the tasks at $c = 1$ with one resource and 0.880 and 0.960 with two (Table 14). With one resource that is $+31.5$ [23.5, 39.5] and $+21.5$ [15.0, 28.0] points above ascending price and level with the checklist (-5.5 [-10.5, -0.5] and -1.5 [-5.0, 2.0]); with two it is above every method of Table 2, by $+44.0$ [36.5, 51.5] and $+44.5$ [37.0, 52.0] points over the weighted sum and $+17.0$ [10.0, 24.0] and $+13.0$ [7.5, 19.0] over the checklist. Its failures come from the labels, and coupling decides what a label error costs. The models wrongly list an infeasible option in 0.14% to 0.49% of cases, at both levels alike, but at $c = 0$ the program takes such an option in 8 of the 52 slots that contain one, and at $c = 1$ in 40 of 40, because under coupling every infeasible option is cheaper than every feasible one and the program takes the cheapest. The pipeline therefore passes less often at $c = 1$ than at $c = 0$, by $+7.5$ [2.5, 13.0] and $+3.5$ [0.5, 7.0] points with one resource and $+9.0$ [4.0, 14.0] and $+4.0$ [1.5, 7.0] with two, although the labelling task is the same at both levels. The prediction that at least 90% of its failures at $c = 1$ come from a wrongly listed option fails with one resource (9 of 23 and 7 of 9 failures); feasible options that the model does not list (10.8% and 3.9% of them) push the cheapest listed plan over the budget in 9 tasks of gpt-4.1-mini. These runs cost 0.79 USD on gpt-4.1-mini and 3.97 USD on gpt-4.1.

Program-aided planning. The review also asked for a baseline in the manner of PAL (Gao et al., 2022), in which the model writes the program. On the cells of the pipeline with a solver, the model writes a Python program that holds every option as data with its price, with two resources its duration, and, for slots with a requirement, the model's judgment of whether the option meets it, and that picks the plan; the program runs in an isolated interpreter with the standard library and its last line is the plan (`coupling_review4_runs_2026-09-22.py`, three predictions fixed before the first paid call; 200 tasks per cell on gpt-4.1-mini and 104 to 105 on gpt-4.1, whose run was stopped at the approved budget with the cells interleaved). It passes 0.800 and 0.981 of the tasks at $c = 1$ with one resource and 0.815 and 0.904 with two (Table 2), $+23.0$ [14.0, 32.0] and $+22.9$ [14.3, 32.4] points against ascending price with one resource, -14.0 [-20.5, -7.5] and $+1.9$ [-2.9, 6.7] against the checklist, and -8.5 [-15.0, -2.5] and $+1.9$ [-2.9, 6.7] against the pipeline with an exact solver; with two resources it stands -6.5 [-12.5, -0.5] and -3.8 [-9.6, 1.9] points from that pipeline. From $c = 0$ to $c = 1$ its pass rate with one resource changes by -8.5 [-15.5, -1.5] and -1.9 [-4.8, 0.0] points. The program's choice fails a requirement in 4.34% and 0.63% of the constrained slots at $c = 1$, and 90% and 100% of its failures there include a requirement failure; 4 and 0 programs return no plan. All three predictions hold. A program has 802 and 941 tokens, so a task costs 2.14 and 11.82 thousandths of a USD (Table 2). These runs cost 1.96 USD on gpt-4.1-mini and 5.45 USD on gpt-4.1; a first attempt (3.89 USD), whose prompt let the models copy the option descriptions into the program so that 28% of gpt-4.1's replies hit the token limit, was discarded before any analysis.

Plan-and-solve prompting in the second environment. The self-review asked how the sorting methods compare with a prompt that reasons before it answers, in the same table and at the same cost unit. We therefore ran plan-and-solve prompting (Wang et al., 2023) as the single generation on the two cells at $c = 1$ of Table 2 (`coupling_review5_runs_2026-09-23.py`, three predictions fixed before the first paid call). It passes 0.910 and 0.940 of the tasks with one resource and 0.755 and 0.885 with two, $+34.0$ [26.5, 41.5] and $+20.0$ [13.0, 27.0] points against ascending price, $+31.5$ [23.5, 39.5] and $+37.0$ [29.0, 45.0] against the weighted sum, -3.0 [-7.5, 2.0] and -3.0 [-7.0, 1.0] against the checklist with one resource, and $+2.5$ [-3.0, 8.0] and -1.5 [-5.5, 2.5] against the

pipeline with an exact solver. A reply has 1,595 and 1,217 tokens, so a task costs 3.8 and 3.1 times a single call; 47 and 8 replies return no plan. All three predictions hold. These runs cost 1.63 USD on gpt-4.1-mini and 7.12 USD on gpt-4.1.

Table 14: The pipeline in which the model lists the options that meet each requirement and a program picks the plan, second environment, 200 tasks per row, with paired differences in points to ascending price and to the checklist of Table 2 (the checklist was run at $c = 1$ only). Wrongly listed is the share of infeasible options that the model lists, missed the share of feasible options that it does not list. The last column counts the constrained slots that contain a wrongly listed option and those in which the program took it.

Model	Resources	c	Pass	Difference to ascending price	Difference to the checklist	Wrongly listed / missed (%)	Wrongly listed option taken
gpt-4.1-mini	one	0	0.960	+10.0 [5.0, 15.0]	–	0.32 / 10.4	3 of 16
		1	0.885	+31.5 [23.5, 39.5]	−5.5 [−10.5, −0.5]	0.14 / 10.8	9 of 9
	two	0	0.970	+16.5 [10.5, 22.5]	–	0.28 / 10.0	3 of 16
		1	0.880	+66.5 [60.0, 73.0]	+17.0 [10.0, 24.0]	0.49 / 10.7	19 of 19
gpt-4.1	one	0	0.990	−1.0 [−2.5, 0.0]	–	0.21 / 4.6	2 of 9
		1	0.955	+21.5 [15.0, 28.0]	−1.5 [−5.0, 2.0]	0.20 / 3.9	7 of 7
	two	0	1.000	+2.5 [0.5, 5.0]	–	0.16 / 5.9	0 of 11
		1	0.960	+61.0 [53.5, 68.0]	+13.0 [7.5, 19.0]	0.16 / 4.5	5 of 5

C.4 Dispersion

Definition. If a coupled task forces each attempt to give up one requirement for another, the failure count varies less than it would under independence. The variance ratio is $\mathrm{VR} = S^2(Y)/\mathbb{E}_0[S^2(Y)]$, the sample variance of the failure count Y over its expectation under a null model. The permutation null permutes the outcomes of each check across attempts within strata, holding not-applicable cells fixed, which keeps every per-constraint failure rate and removes dependence between checks; its expectation has the closed form $\mathbb{E}_0[S^2] = \big(\sum_i \sum_j p_j(1-p_j) + \sum_i \mu_i^2 - N\bar{Y}^2\big)/(N-1)$, where p_j is the failure rate of check j in the stratum of attempt i and μ_i is the expected failure count of attempt i. The null model that keeps task difficulty uses a logistic model $\mathrm{logit}\, P(F_{tj} = 1) = \theta_t + \beta_j$, $\theta_t \sim \mathcal{N}(0, \sigma^2)$, whose task effects inflate the expected variance. $\mathrm{VR} < 1$ indicates that failures of different checks covary negatively on average. When failure indicators are negatively associated, the probability that all checks pass is at most the product of the per-constraint pass probabilities (Joag-Dev & Proschan, 1983), so a product-law projection overestimates the pass rate.

Coupling does not consistently concentrate the failures of a single attempt. If coupling forced each attempt to trade one requirement for the budget, the variance ratio of single attempts would fall as c rises (Figure 4, left). Single attempts of gpt-4.1 and gpt-4.1-mini fail almost only on the budget (99.1% and 97.4% of failed attempts), which holds their variance ratio near 1 by construction. The ratio at $c = 1$ minus that at $c = 0$ is -0.017 [-0.046, 0.000] for gpt-4.1 and -0.057 [-0.094, -0.023] for gpt-4.1-mini, whose attempts under coupling fail either a requirement or the budget and never both. The Qwen models violate requirements in single attempts (Qwen3-8B in 41% of attempts at $c = 1$, $K = 6$) and show no monotone change, with differences of $+0.061$ [-0.095, 0.247] and -0.026 [-0.089, 0.039]. A preregistered fallback regresses the variance ratio on coupling and pass rate across the pilot settings. Its coupling coefficient is -0.002 [-0.039, 0.048] and -0.009 [-0.031, 0.015] for the GPT models and -0.315 [-0.409, -0.218] and -0.346 [-0.478, -0.069] for the Qwen models. The two analyses disagree for the Qwen models; the pilot regression spans all four slack values, including the tightest (0.02), which no Qwen main cell uses.

TravelPlanner failures are close to independent across plans and negatively associated within a query. The 900 plans have a mean of 1.66 failed checks, and 10.9% pass. The variance of the failure count is 1.166 against 1.197 expected under the permutation null, a variance ratio of 0.974 [0.838, 1.125] ($p = 0.2557$), and the ratio under the null model that keeps task difficulty is 0.660 [0.590, 0.736] (Figure 4, right). The two nulls answer different questions: across all plans the checks fail nearly as if independent, but once the difficulty of each query is held fixed the failure count varies less than independence predicts, which is the trade-off between checks that Section 4.3

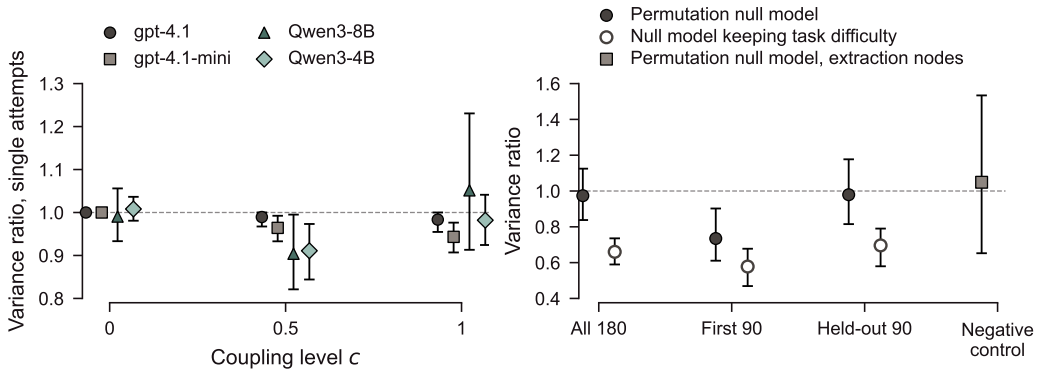

Figure 4: Variance ratio of the failure count. Left, single attempts in the synthetic environment under the permutation null, by coupling level, with 95% task-bootstrap intervals. Right, 900 TravelPlanner plans from gpt-4.1 under the permutation null (filled) and the null model that keeps task difficulty (open), for all queries and for each half, and the negative control of 60 extraction tasks with 40 independent nodes. Values below the dashed line mean failures are spread more evenly than independence predicts.

describes. The first 90 queries were used in an earlier comparison of agent architectures and the other 90 are held out. The halves give 0.735 and 0.979 under the permutation null (0.735 on the first half and 0.979 on the held-out half), and the negative control does not depart from 1 (1.049 [0.652, 1.534]). The product of per-constraint pass rates predicts a pass rate of 0.143 against the observed 0.109, a difference of -0.034 [-0.073, 0.009]. The share of failing plans with exactly one failed check exceeds its null value on the first half (0.612 against 0.527) and not on the held-out half (0.259 against 0.256). Earlier drafts reported a clearly lower ratio and an analysis of 87 architectures; both came from an evaluator that misread some restaurant names (Appendix C.6), and the counts of those architectures cannot be re-graded because their plans were not logged, so they are withdrawn.

Language-model forecasts expected clustering. Following forecast elicitation in experimental economics (DellaVigna et al., 2019), three frontier models answered a questionnaire that described the design without results, five times each. All 15 forecasts expected TravelPlanner failures to cluster, with a median variance ratio of 1.5, and all 15 named the prediction from the theory of constraints as the most accurate. The median forecast of the calculator regression rate for gpt-4.1 was 0.06 at $c = 0$ and 0.40 at $c = 1$, and both measured values are 0. Language models have predicted neuroscience results better than human experts (Luo et al., 2024), but they remain a weak proxy for the expectations of agent researchers. We wrote the questionnaire after seeing the pilot and the pooled TravelPlanner ratio, although the forecasters saw neither (Appendix B.4).

C.5 Failure structure and predictions of the gain

Proposition 2 (Accounting identity for the gain)**.** *Let p_0 be the baseline pass rate and s_T the share of baseline attempts with $\emptyset \neq \Phi_a(t) \subseteq T_b$. The gain in pass rate G_b equals $s_T(1 - m_b) - p_0 r_b$ whenever (i) b repairs every attempt whose baseline failures lie in T_b, and (ii) no attempt with a baseline failure outside T_b passes under b.*

Proof. $G_b = P(\Phi_a \neq \emptyset, \Phi_b = \emptyset) - P(\Phi_a = \emptyset, \Phi_b \neq \emptyset)$, and the second term is $p_0 r_b$. Under (ii) the first term is $P(\emptyset \neq \Phi_a \subseteq T_b, \Phi_b = \emptyset)$. Under (i) every attempt counted there belongs to S_b, and a fraction $1 - m_b$ of S_b passes. □

Predictions. The prediction from the theory of constraints is $\hat{G} = s_T(1 - \hat{m}) - p_0 \hat{r}$, with $\hat{m}$ and $\hat{r}$ pooled over the other methods of the same model and cell. The prediction from the share of violations is $v_T \hat{q}$, with v_T the share of baseline attempts violating an addressed check and $\hat{q}$ the repair rate of the other methods. The prediction from the product of per-constraint pass rates lowers each addressed check's failure rate by $\hat{q}$ and recomputes the product of per-constraint pass

rates. Additive main effects predict a method's gain in a cell from its gain in the other cells and the other methods' gains in that cell; for calculator plus checklist it adds the two single gains, following Liu (2026). The two-way additive prediction, added after the results, is the method's mean gain in the other cells plus the mean gain of the other methods in the cell minus the mean gain of the other methods in the other cells. Each family of two models yields 60 predictions (five methods, six cells); feedback, whose addressed checks vary by task, is reported separately.

The prediction from the theory of constraints substitutes estimates of m and r from other methods into this identity. It therefore needs conditions (i) and (ii) and also needs m and r to transfer across methods. Both conditions are testable, and this appendix shows which one fails for each method.

Table 15: Predictions of the gain on 60 held-out cases per model family (five methods, six cells, two models). MAE is the mean absolute error of the predicted gain in pass rate; Spearman is the rank correlation with the observed gain; Sign is the share of predictions with the observed sign.

	gpt-4.1 and gpt-4.1-mini			Qwen3-8B and Qwen3-4B		
Prediction	MAE	Spearman	Sign	MAE	Spearman	Sign
Theory of constraints	0.184	0.109	0.767	0.260	−0.260	0.550
Share of violations	0.226	0.012	0.800	0.113	−0.188	0.800
Product of per-constraint pass rates	0.227	−0.001	0.800	0.112	−0.195	0.800
Additive main effects	0.116	0.889	0.900	0.105	0.481	0.767
Two-way additive (exploratory)	0.067	0.903	0.900	0.092	0.471	0.733

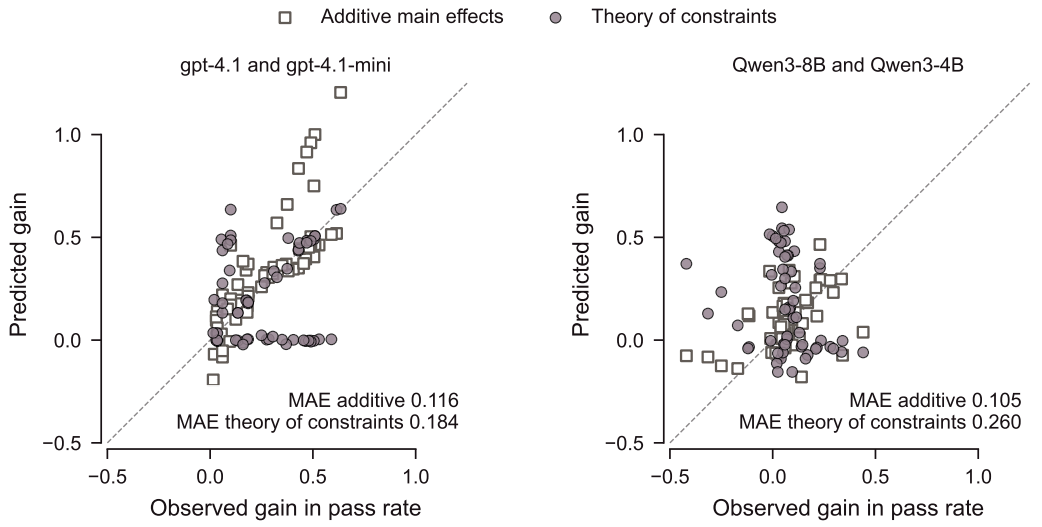

Figure 5: Predicted against observed gains in pass rate for the prediction from the theory of constraints (filled circles) and additive main effects (open squares). The dashed line marks perfect prediction. The prediction from the theory of constraints assigns near-zero gains to checklists and sub-agents, whose observed gains reach 0.590, and large gains to sampling, which repairs few attempts.

The prediction from the theory of constraints loses to an additive model. On the GPT models this prediction has a mean absolute error of 0.184, below the prediction from the share of violations (0.226) and the product of per-constraint pass rates (0.227) and above additive main effects (0.116), a difference of $+0.068$ [0.002, 0.135] (Table 15). On the Qwen models it is the least accurate predictor, 0.260 against 0.105 for the additive model, a difference of $+0.155$ [0.114, 0.197]. Its advantage over the share of violations on the GPT models grows with coupling as preregistered (-0.031, -0.042 and -0.051 at the three levels), without reaching significance ($p = 0.079$).

Both conditions of the accounting identity fail. Checklists and sub-agents address the requirements yet remove budget failures, which breaks condition (ii) of Proposition 2. For gpt-4.1 at $c = 1$ and $K = 3$ the checklist raises the pass rate from 0.360 to 0.950, and sub-agents raise it to 0.890, while only 0.5% of baseline attempts fail inside their addressed checks (Figure 5). Best-of-four sampling with a budget verifier addresses the budget and fixes few budget failures, which breaks condition (i); in the same cell it gains 0.100 against a prediction of 0.635. The four samples of a task agree on the budget verdict in 85.5% of gpt-4.1 tasks at $c = 1$, against 32.2% expected if samples were independent. The five TravelPlanner samples of a query agree on the budget verdict in 73.0% of 174 queries against 22.6% under the permutation null, and the best of five samples passes on 16.7% of queries against 13.3% for a language-model selector. A selector cannot repair a failure that every candidate shares. The point estimate of the calculator and checklist interaction is negative for every model and coupling level (differences in differences from -0.378 to -0.020), mostly because each alone lifts the GPT pass rates to 0.915 or more; on Qwen3-4B the two together pass less often than the checklist alone (0.383 against 0.580 at $c = 0.5$), an interference between the two methods rather than a ceiling (Appendix C.8).

C.6 Fixes on TravelPlanner and a correction to the evaluator

Evaluator correction. The sole-planning reference tables are fixed-width text. The first version of our evaluator took the first number in a restaurant row as its cost, which cut names that contain a number (197 of 11,460 validation rows; "Castle 9" was read as "Castle" costing 9) and failed plans that used the full name. The corrected parser reads columns by the positions of the header; the 11,263 rows that the first version read correctly are unchanged. Re-grading the 900 plans raises the passing plans from 71 to 98, and every TravelPlanner number in this paper comes from the corrected evaluator.

Runs. All arms use gpt-4.1, the sole-planning prompt and the corrected evaluator; single-call arms sample at temperature 1.0 and repair calls at temperature 0. Table 16 gives the pass rates. The removal step and the repair messages read the constraint fields that the dataset annotates for every query (room type, house rule, days and number of cities) and the columns of the reference tables, never a reference plan. An agent in use has only the text of the query, so we let gpt-4.1 at temperature 0 extract the nine fields the programs read from the query text alone (`tp_extract_runs_2026-09-21.py`, two predictions fixed before the first paid call). All nine agree with the annotation on 174 of the 180 queries (96.7%); four of the six differences concern the number of cities, one the number of travellers and one the room type. On these six queries the removal and sort arm and the repair were run again with the extracted fields, and as many of them pass as with the annotated fields (1 and 2 of the 6), so the pass rates of 23.3% and 41.7% are unchanged and both predictions hold. The evaluator still grades with the annotation. The extraction and the re-runs cost 0.34 USD. The repair messages list the failed checks; the alternatives variant adds, for each stay that breaks minimum nights, room type or house rule, the five cheapest accommodations in that city that meet them, and, when the plan is over budget, the six largest savings that keep every requirement (a cheaper feasible stay, restaurant or flight). An earlier version of that message could point two meals at the same restaurant and was run before the evaluator correction; its results are superseded. The runs cost 33.16 USD in API calls; the shuffled arm stopped after 162 queries when the account ran out of credits and its last 18 queries were run on 2026-09-20. These pass rates are not comparable with the highest reported on TravelPlanner: Choi et al. (2025) reach 44.4% on the validation split with Gemini-2.5-Pro and a multi-agent search over tools, Qiu et al. (2026) 80.56% with a GPT-5.6 backbone and a repository of compiled artifacts, and Zhang et al. (2025) 82.7% on the test split with a Qwen3-8B fine-tuned on multi-agent traces; our arms hold the model fixed at gpt-4.1 in sole-planning mode and change only what it is shown.

C.7 TravelPlanner checks and additional measurements

Checks. The 12 analysis checks are completeness (delivery merged with the absent-information check), city route, distinct restaurants, distinct attractions, minimum nights, transportation (commonsense rule merged with the requested mode), current-city consistency, sandbox existence, budget, and, when requested, room rule, cuisine and room type. Minimum nights, budget, room rule, cuisine and room type are not applicable when the sandbox check fails, because the evaluator passes

Table 16: TravelPlanner validation, gpt-4.1, pass rate over 180 queries, with paired differences and 95% bootstrap intervals over queries.

Arm	Pass	Difference
Default order	0.106	
Ascending price	0.100	-0.6 [-5.6, 4.4] vs default order
Remove infeasible stays, then sort	0.233	$+12.8$ [6.7, 18.9] vs default order
Checklist on sorted options	0.222	$+11.7$ [6.1, 17.8] vs default order
First sample, no repair	0.128	
Repair listing failed checks	0.167	$+3.9$ [1.1, 6.7] vs no repair
Repair with feasible alternatives	0.361	$+19.4$ [13.9, 25.6] vs listing failed checks
Remove, sort, then repair with alternatives	0.417	$+31.1$ [23.9, 38.3] vs default order
Any of four temperature samples	0.167	
Any of four shuffled samples	0.278	$+11.1$ [5.6, 16.7] vs temperature

items it cannot find. A plan that cannot be parsed fails completeness and has every other check not applicable. The pass criterion is the official one.

Additional measurements. Table 17 gives the dispersion results per half and Table 18 the negative control with its sensitivity logs. The mean Mantel–Haenszel log odds ratio (Mantel & Haenszel, 1959) between the budget check and spending-related checks is 0.264 lower than between the budget and other checks ($p = 0.238$ over 462 label assignments). In a same-run comparison on 90 queries, a method that gives the planner a cost estimate computed by code and no sub-agent reports passes 0.244 of queries against 0.189 for single generation after the evaluator correction, a difference of $+0.056$ [-0.011, 0.122] with McNemar (McNemar, 1947) exact $p = 0.227$; before the correction the values were 0.200, 0.122, $+0.078$ [0.011, 0.144] and $p = 0.065$.

How strict is the coupling of a real task? For each validation query with a requirement on the accommodation we take every city table with at least four accommodations and ask where the cheapest accommodation that meets the requirement, the minimum nights and the room rule sits in the price order. Over 103 such slots in 51 queries, with a median of 12 accommodations, the cheapest accommodation meets the requirement in 36.9% of slots, the median rank of the cheapest one that does is 2, on average 12.0% of the options are cheaper than it, and only 1.9% of the slots separate the two kinds of option completely, as the synthetic environment does by construction.

Regressions by the coupling of the query. A review asked for the regressions of a real task stratified by its coupling. We call a query with a requirement on the accommodation coupled when, in at least one city it visits, the cheapest accommodation fails the requirement, so that the budget and the requirement trade against each other (42 of the 180 validation queries), uncoupled otherwise (56), and put the 82 queries without such a requirement in a third group. Over the four one-round repair messages (Appendices C.2 and C.6), the plans that a repair fixes fail another check in 10 of 24 cases on coupled queries, 9 of 36 on uncoupled ones and 19 of 39 on queries without an accommodation requirement; with feasible alternatives the counts are 3 of 15, 1 of 15 and 4 of 20 (`coupling_review6_fixes_2026-09-23.py`). The direction between the two groups with a requirement matches the synthetic result, but 24 and 36 fixed plans do not separate the two shares, nor do the 12 and 53 fixed by the message with the failed checks over all five samples (8 and 32 fail another check), and the third group shows that a measure on the accommodation misses the coupling of the city route, the budget and the minimum nights, through which most of its regressions move.

Table 17: TravelPlanner dispersion (gpt-4.1, five samples per query at temperature 1.0). P_1 is the share of failing plans with exactly one failed check; p is its one-sided permutation p value under the permutation null.

Plans	Units	Mean Y	Pass	VR, permutation null	VR, null keeping difficulty	P_1	P_1 under permutation null	p
All 180 queries	900	1.66	0.109	0.974 [0.838, 1.125]	0.660 [0.590, 0.736]	0.424	0.381	0.0020
First 90	450	1.22	0.169	0.735 [0.611, 0.902]	0.578 [0.469, 0.678]	0.612	0.527	0.0002
Held-out 90	450	2.10	0.049	0.979 [0.815, 1.177]	0.696 [0.580, 0.790]	0.259	0.256	0.4205

Table 18: Negative control, 60 extraction tasks with 40 independent nodes each. The primary log and four of the five sensitivity logs use gpt-4.1-mini; the last uses gpt-4.1.

Extraction log	Node error rate	VR, permutation null [CI]
primary	0.113	1.049 [0.652, 1.534]
replicate 2	0.115	1.011 [0.670, 1.476]
replicate 3	0.115	1.168 [0.760, 1.681]
mixed thresholds	0.120	0.946 [0.682, 1.234]
word-count subtask	0.926	1.175 [0.693, 1.687]
gpt-4.1, fixed parser	0.094	0.731 [0.489, 1.014]

C.8 Interaction of calculator and checklist

Table 19 gives the preregistered interaction analysis. For the GPT models, calculator or checklist alone brings the pass rate close to 1 at every coupling level, so the combination cannot add the sum of the two gains and the interaction is negative by ceiling.

Table 19: Pass rates pooled over K, the interaction coefficient β_3 of a logistic model with task random effects, and the difference in differences on the probability scale with 95% task-bootstrap interval.

Model	c	single	calculator	checklist	both	β_3 [Wald CI]	Difference in differences [CI]
gpt-4.1	0	0.880	0.985	0.975	0.985	−3.60 [−5.64, −1.56]	−0.095 [−0.130, −0.062]
	0.5	0.710	0.993	0.980	0.993	−6.00 [−8.92, −3.08]	−0.270 [−0.315, −0.225]
	1	0.588	0.988	0.975	0.998	−5.82 [−10.12, −1.52]	−0.378 [−0.425, −0.330]
gpt-4.1-mini	0	0.485	0.930	0.915	0.993	−0.58 [−1.90, 0.73]	−0.367 [−0.422, −0.312]
	0.5	0.562	0.953	0.983	0.995	−1.61 [−3.30, 0.08]	−0.378 [−0.432, −0.323]
	1	0.593	0.943	0.985	0.990	−2.29 [−3.70, −0.89]	−0.345 [−0.400, −0.290]
Qwen3-8B	0	0.688	0.843	0.833	0.845	−1.43 [−2.11, −0.74]	−0.142 [−0.198, −0.087]
	0.5	0.713	0.767	0.777	0.812	−0.16 [−0.84, 0.51]	−0.020 [−0.070, 0.030]
	1	0.662	0.735	0.748	0.775	−0.45 [−1.11, 0.22]	−0.045 [−0.095, 0.005]
Qwen3-4B	0	0.490	0.505	0.797	0.535	−2.30 [−2.90, −1.71]	−0.278 [−0.343, −0.210]
	0.5	0.328	0.372	0.580	0.383	−2.15 [−2.78, −1.52]	−0.242 [−0.310, −0.177]
	1	0.472	0.477	0.637	0.532	−1.00 [−1.60, −0.40]	−0.110 [−0.177, −0.043]

D Extended studies and ablations

D.1 Difficulty controls, a dependence-preserving reference and failure transitions

These analyses answer points raised in review and were defined after the results were in. They are computed by `binding_review_fixes_2026-09-18.py` on the saved logs, with the addressed checks, failure sets and fixed attempts of Section 3.1.

Difficulty controls. Each task carries the number of plans that meet every requirement and the budget, which the exhaustive solver counts. Attempts are placed in strata formed by K and the quintile of that count, and the attempts at $c = 1$ are reweighted onto the stratum distribution at $c = 0$. The second control keeps only attempts whose baseline failure was the budget alone, which leaves fixed attempts for gpt-4.1 of $S = 22$ at $c = 0$ and 55 at $c = 1$, for gpt-4.1-mini of 114 and 116, for Qwen3-8B of 78 and 14 and for Qwen3-4B of 152 and 144. Reweighting gives values of m at $c = 1$ of 18.0%, 19.2% and 41.7% for gpt-4.1, Qwen3-8B and Qwen3-4B, and gpt-4.1-mini has no stratum present at both levels. For the calculator m stays at 0 on both GPT models under either control.

Difficulty by pass probability. A third control uses the model itself as the measure of difficulty. Every task has four samples at temperature 1.0 from best-of-four sampling, and the number of them that pass estimates how hard the task is for that model. Fixed attempts at $c = 1$ are reweighted

Table 20: Checks of the main findings, detailed in the appendix of the last column. Four values are in the order gpt-4.1, gpt-4.1-mini, Qwen3-8B, Qwen3-4B.

Check	Result	Where
Two difficulty controls	m after feedback rises from $c = 0$ to $c = 1$ by $+18.2$, $+11.2$, $+30.6$ and $+53.0$ points among budget-only failures and by $+18.5$, $+14.2$, -1.3 and $+51.7$ when reweighted on pass probability	App. D.1
Without conditioning on the repair	Among all attempts that failed the budget alone, the share that then violates a requirement rises by 6.8, 9.7, 25.2 and 42.8 points	App. D.1
Price relation reversed	With every feasible option cheaper than every infeasible one, none of the 71 and 26 fixed attempts fails another check, against 12.9% and 17.5% at $c = 1$	App. D.1
Milder coupling, more options, paraphrases	With the price ranges overlapping by two ranks, m after feedback is 7.5% and 11.4% on gpt-4.1-mini and gpt-4.1, against 2.7% and 2.4% under independence; the gain of ascending price is $+23.0$ to $+39.0$ points with overlap, $+32.0$ to $+63.0$ with 20 and 50 options per slot and $+20.5$ to $+36.5$ under two paraphrases	App. D.1, D.3
Exact solver on the model's labels	Wrongly listed options 0.14% to 0.49% at both levels; the program takes one in 8 of 52 such slots at $c = 0$ and 40 of 40 at $c = 1$	App. C.3
TravelPlanner, fields read from the query text	All nine fields correct on 174 of 180 queries; pass rate 41.7% unchanged	App. C.6
Models of other families, on tasks not matched to them	On Phi-4-mini and Mistral-7B, m after feedback rises from 8.3% and 5.6% at $c = 0$ to 75.4% and 72.3% at $c = 1$ (42.9% and 58.2% under independence)	App. D.2
TravelPlanner, one repair round on all five samples	With the failed checks alone, 63 of 108 fixed plans fail another check, 58.3% [48.9, 67.2]	App. C.2
Training on permuted option orders	A LoRA adapter on Qwen3-4B trained to answer consistently across orders raises single pass at $c = 1$ from 0.472 to 0.598, leaves the first listed feasible option at 55.7% against 57.6%, raises m after feedback from 56.7% to 69.9% (36.4% under independence) and leaves the gain of ascending price at $+10.0$ points against $+2.5$	App. D.4

onto the distribution of K and that number among the fixed attempts at $c = 0$, which covers 87.7%, 95.2%, 99.2% and 98.0% of the fixed attempts at $c = 1$ for gpt-4.1, gpt-4.1-mini, Qwen3-8B and Qwen3-4B. On this common distribution m rises from $c = 0$ to $c = 1$ by $+18.5$ [8.5, 29.1] points, $+14.2$ [7.9, 21.2], -1.3 [-7.8, 4.8] and $+51.7$ [42.2, 58.9], the same pattern as without the control (`coupling_review3_fixes_2026-09-21.py`).

Milder coupling. In a coupled slot of the main study each of the five infeasible options is cheaper than each of the three feasible ones. A review asked whether m stays above its reference when the two price ranges overlap. We rebuilt the 400 tasks at $c = 1$ of each GPT model with the prices of the feasible options moved down by $g = 1$ and $g = 2$ ranks, so that one and two infeasible options cost more than every feasible one, recomputed the budget with the same slack, and ran single generation and feedback as in the main study, whose records are $g = 0$ on the same seeds (`coupling_review3_runs_2026-09-21.py`, two predictions fixed before the first paid call). On gpt-4.1-mini m is 12.9% at $g = 0$, 9.5% [4.7, 14.8] at $g = 1$ and 7.5% [3.3, 12.3] at $g = 2$ ($S = 124$, 126 and 133), against 4.8%, 3.5% and 2.7% expected under independence. On gpt-4.1 it is 17.5%, 13.0% [4.1, 23.5] and 11.4% [2.5, 23.3] ($S = 57$, 46 and 35), against 4.0%, 2.5% and 2.4%. Both predictions hold: m stays above its reference at every overlap and does not rise with the overlap. From $g = 0$ to $g = 2$ it falls by 5.4 points [-12.8, 1.9] and 6.1 [-20.4, 8.9], so the intervals include no change, and with $S = 35$ the interval of gpt-4.1 at $g = 2$ only just excludes its reference. The regressions therefore weaken as the coupling gets milder and do not vanish. The run cost 2.42 USD.

Price relation reversed. A direct test of the cause named in Section 4.1. We rebuilt the 400 tasks at $c = 1$ of each GPT model with the price blocks of every coupled slot swapped, so that every feasible option is cheaper than every infeasible one, recomputed the budget with the same slack and ran single generation and feedback as in the main study (`coupling_review5_runs_2026-09-23.py`, two predictions fixed before the first paid call). Single generation passes 0.760 and 0.825 of these tasks against 0.593 and 0.588, because the cheapest option of a coupled slot now meets its requirement. After feedback none of the 71 and 26 fixed attempts fails another check, against 12.9% and 17.5% at $c = 1$ of the main study on the same seeds, a difference of -12.9 [-18.9, -7.2] and -17.5 [-28.3, -8.2] points, and the independence reference is 0.0% and 0.0%. The first prediction holds and the second fails narrowly: feedback fixes 74.0% and 37.1% of the failing attempts against 76.1% and 34.5% at $c = 1$. With the same tasks, requirements and slack, the regressions after feedback follow the price relation of the coupled slots and nothing else in the task. The run cost 1.07 USD.

More tasks without coupling. The 22 fixed gpt-4.1 attempts at $c = 0$ were the smallest cell of Section 4.1. We generated 1,000 further tasks at $c = 0$ for gpt-4.1 with the slack and number of feasible options per slot of the main run, 500 each at $K = 3$ and $K = 6$, and ran single generation and feedback (`coupling_review4_runs_2026-09-22.py`, one prediction fixed before the first paid call). Single generation passes 0.872 of them, feedback fixes 53 of the 128 failing attempts, and none of the 53 fails another check. Pooled with the main run, 0 of 75 fixed attempts at $c = 0$ fail another check (interval [0, 4.9%]) against 17.5% of 57 at $c = 1$, a difference of 17.5 points [8.3, 28.3]. The prediction holds. The run cost 2.10 USD.

Without conditioning on the repair. The set of fixed attempts is defined by the outcome, so we also take every attempt whose baseline failure was the budget alone, whatever happened to the budget after feedback, and count those that violate a requirement afterwards. From $c = 0$ to $c = 1$ this share rises by 6.8 points [3.1, 11.0] on gpt-4.1, 9.7 [5.2, 14.9] on gpt-4.1-mini, 25.2 [2.8, 48.5] on Qwen3-8B and 42.8 [35.3, 50.2] on Qwen3-4B.

A reference that keeps within-attempt dependence. The independence-based value multiplies per-constraint rates and so ignores that some attempts fail more checks than others. A second reference pairs each fixed attempt with a matched attempt of the same model, condition, coupling level and K, drawn at random from all tasks, and asks whether the matched attempt fails any check outside the addressed checks. Over 2,000 draws at $c = 1$ with feedback it gives 4.3% [0.0, 8.8] for gpt-4.1, 4.8% [1.6, 8.9] for gpt-4.1-mini and 25.3% [18.7, 32.7] for Qwen3-4B, against observed 17.5%, 12.9% and 56.7%.

Failure transitions. Counting what each fixed attempt failed before and after the method locates the move. With feedback at $c = 1$, gpt-4.1 moves 45 attempts from a budget failure to no failure and 10 from the budget to a requirement, gpt-4.1-mini moves 103 and 13 with 3 going from a requirement to the budget, Qwen3-4B moves 61 and 83, and Qwen3-8B moves 97 of its 128 fixed attempts from a requirement to no failure and only 9 from the budget to no failure, 14 having failed the budget alone, which is why its m stays low.

Conditions of the accounting identity. The two conditions are checked per method. At $c = 1$ condition (i) holds for 34.5% of the 165 gpt-4.1 attempts whose baseline failures lie in the checks feedback addresses and for 20.2% of the 163 under sampling, while condition (ii) is violated by 93.9% of the 163 checklist attempts and 85.9% of the 163 sub-agent attempts that carry a baseline failure outside the addressed checks. On Qwen3-4B at $c = 1$ condition (i) holds for 71.1% of the 211 attempts inside the checks feedback addresses and 12.9% of the 201 under sampling, and condition (ii) is violated by 42.9% of the 205 checklist attempts and 22.0% of the 205 sub-agent attempts outside the addressed checks.

Residuals of the additive model. Over the 144 cells of model, coupling level, K and method, an additive fit of the pass-rate gain reaches $R^2 = 0.396$ and adding a coupling by method interaction reaches 0.423, with a mean absolute residual of 0.109 in both. The mean residual of the additive fit is 0.003, -0.006 and 0.003 at the three coupling levels, so the residuals carry no coupling pattern.

D.2 Models of other families

A review asked whether the findings hold on other model families. After the results were in, we ran single generation and feedback with two models of other families, Phi-4-mini and Mistral-7B, on the 1,200 main tasks of gpt-4.1-mini, served from the official weights on vLLM with one rented card and no API calls (`coupling_review4_runs_2026-09-22.py`, three predictions per model fixed before the first call). The tasks were not difficulty matched to these models, so the comparison is within model across coupling levels (Table 21). On Phi-4-mini (Abouelenin et al., 2025), single generation passes 0.052, 0.235 and 0.278 of the tasks at $c = 0$, 0.5 and 1; m after feedback is 8.3% [1.8, 16.2], 69.4% [62.7, 76.0] and 75.4% [69.2, 81.2] ($S = 60$, 183 and 195) against 21.9%, 39.6% and 42.9% under independence, and the first listed feasible option is taken in 33.2%, 43.7% and 58.7% of the constrained slots where it is not the cheapest feasible one, against 33.3% under random choice. All three predictions hold. On Mistral-7B (Jiang et al., 2023a), single generation passes 0.040, 0.172 and 0.172 of the tasks at $c = 0$, 0.5 and 1; m after feedback is 5.6% [0.0, 18.8], 80.3% [73.8, 86.5] and 72.3% [66.2, 78.2] ($S = 18$, 147 and 220) against 26.0%, 58.4% and 58.2% under independence, and the first listed feasible option is taken in 13.7%, 20.2% and 25.8% of the constrained slots where it is not the cheapest feasible one, against 33.3% under random choice. The

predictions on m hold and the prediction on the first listed option fails: Mistral-7B takes it in 25.8% of these slots, below the third of random choice, and takes an infeasible option in 15.2% of them, so its regressions come from misjudged requirements rather than from the position.

Table 21: Two models of other families on the main tasks of gpt-4.1-mini, single generation and one round of feedback. Single pass is the pass rate of single generation over the 400 tasks of the level; fixed and failing another check count attempts, and the share carries a 95% task-bootstrap interval; independence is the reference of Section 3.1; first listed is the share of constrained slots, among those whose first listed feasible option is not the cheapest feasible one, in which single generation takes it, with a Wilson interval.

Model	c	Single pass	Fixed	Failing another check	Share	Independence	First listed
Phi-4-mini	0	0.052	60	5	8.3% [1.8, 16.2]	21.9%	0.332 [30.6, 35.9]
	0.5	0.235	183	127	69.4% [62.7, 76.0]	39.6%	0.437 [40.9, 46.6]
	1	0.278	195	147	75.4% [69.2, 81.2]	42.9%	0.587 [55.9, 61.4]
Mistral-7B	0	0.040	18	1	5.6% [0.0, 18.8]	26.0%	0.137 [11.9, 15.8]
	0.5	0.172	147	118	80.3% [73.8, 86.5]	58.4%	0.202 [18.0, 22.6]
	1	0.172	220	159	72.3% [66.2, 78.2]	58.2%	0.258 [23.5, 28.4]

D.3 Coupling strength, option count and paraphrases in the sorting study

Coupling strength, option count and other orders. Three questions about the design need no model call, so we answer them by simulating the reading rule on fresh tasks (`coupling_review_fixes_2026-09-20.py`, 200 tasks per row, seeds from 970000). Shifting the price block of the options that meet a requirement down by g ranks, so that the two price ranges overlap, leaves the rule's pass rate at 1.000 under ascending price for $g = 0$ to 4, against 0.470, 0.495, 0.475, 0.455 and 0.480 under the default order. Giving each slot 8, 12 or 20 options gives 1.000 against 0.465, 0.470 and 0.440, and 50 options on widened price ranges gives 1.000 against 0.480, with the budget of the runs with 20 or more options set from 20,000 sampled plans instead of by enumeration. With two resources at $c = 1$ the rule reaches 0.385 under the weighted sum, 0.140 under the default order, 0.050 under ascending price, 0.050 under a Pareto-layer order and 0.080 under an epsilon-constraint order that keeps the options below a per-slot duration cut before sorting by price. A model departs from the rule, so these are predictions for the rule rather than measurements of a model.

The same questions put to the models. Because a simulation fixes the rule and not the model, the first three questions were also run on models (`coupling_review_runs_2026-09-20.py`): 14 arms of 200 paired tasks, the default order against ascending price at $c = 1$ with one resource, on task seeds drawn apart from those of the registered study. Five predictions were fixed in the runner before its first paid call, and Table 22 gives the outcome. On gpt-4.1-mini the gain of ascending price stays above half of the 25.5 points measured as registered in every setting and does not fall as the two price ranges overlap more, at $+34.5$, $+27.5$ and $+39.0$ points for $g = 1$, 2 and 4; it grows with the number of options, $+50.0$ at 20 and $+63.0$ at 50; and it moves by 0.5 points between the two paraphrases. On gpt-4.1 the gain also stays above half of its registered 21.5 points, at $+29.0$, $+23.0$ and $+24.0$ points for $g = 1$, 2 and 4, and grows with the number of options, $+32.0$ at 20 and $+44.5$ at 50; it is $+20.5$ points under both paraphrases. All five predictions hold on gpt-4.1, and four of them on gpt-4.1-mini. The prediction that the default order falls as options are added holds on gpt-4.1 by a small margin (0.545 at 20 options against 0.510 at 50) and fails on gpt-4.1-mini (0.295 against 0.325), where the share of choices that take the first listed option meeting the requirement falls from 0.580 to 0.515; on gpt-4.1 that share is 0.420 and 0.414. On Qwen3-4B the gain reaches at most $+8.5$ points and its interval covers zero in six of the seven settings, as in the registered study. The runs cost 3.58 USD on gpt-4.1-mini and 18.00 USD on gpt-4.1, whose run was completed in two sessions, resuming by arm and seed, because the account ran out of credits; Qwen3-4B was served from the official weights on a rented GPU and made no paid call.

Table 22: Graded coupling, option count and paraphrases on the GPT models: pass rate in the default order and in ascending price at $c = 1$ with one resource, and their paired difference in points with a 95% task-bootstrap interval (200 paired tasks per cell). The first row repeats the registered setting, 12 options with the two price ranges apart and the original wording. Overlap g moves the price block of the options that meet a requirement down by g ranks. Qwen3-4B is in Table 23.

Setting	gpt-4.1-mini		gpt-4.1	
	Default / ascending	Difference	Default / ascending	Difference
As registered	0.315 / 0.570	+25.5 [18.5, 33.0]	0.525 / 0.740	+21.5 [14.0, 28.5]
Overlap $g = 1$	0.360 / 0.705	+34.5 [27.0, 42.0]	0.510 / 0.800	+29.0 [22.5, 36.0]
Overlap $g = 2$	0.305 / 0.580	+27.5 [20.0, 35.0]	0.535 / 0.765	+23.0 [16.5, 29.5]
Overlap $g = 4$	0.330 / 0.720	+39.0 [32.0, 46.0]	0.590 / 0.830	+24.0 [17.5, 30.5]
20 options per slot	0.295 / 0.795	+50.0 [43.0, 57.0]	0.545 / 0.865	+32.0 [25.0, 39.0]
50 options per slot	0.325 / 0.955	+63.0 [56.0, 69.5]	0.510 / 0.955	+44.5 [37.5, 51.5]
Paraphrase b	0.315 / 0.680	+36.5 [29.5, 43.5]	0.470 / 0.675	+20.5 [14.0, 27.5]
Paraphrase c	0.320 / 0.680	+36.0 [29.0, 43.0]	0.545 / 0.750	+20.5 [13.5, 27.5]

Table 23: The settings of Table 22 on Qwen3-4B, served from the official weights on vLLM with thinking disabled.

Setting	Qwen3-4B	
	Default / ascending	Difference
As registered	0.095 / 0.120	+2.5 [−3.5, 8.0]
Overlap $g = 1$	0.085 / 0.110	+2.5 [−3.0, 8.5]
Overlap $g = 2$	0.090 / 0.125	+3.5 [−2.5, 9.5]
Overlap $g = 4$	0.110 / 0.160	+5.0 [−1.0, 11.0]
20 options per slot	0.085 / 0.135	+5.0 [−1.0, 11.0]
50 options per slot	0.060 / 0.055	−0.5 [−5.0, 4.0]
Paraphrase b	0.070 / 0.125	+5.5 [−0.5, 11.5]
Paraphrase c	0.065 / 0.150	+8.5 [2.5, 14.5]

D.4 Training for consistent answers across option orders

A review asked whether training removes the position bias of Section 4.3, as in reinforcement learning that rewards the same answer under permuted options (Zheng et al., 2026). We trained one LoRA adapter (rank 16) on Qwen3-4B with GRPO (`coupling_pagrpo_train_2026-09-22.py`): 2,000 fresh tasks of the first environment, disjoint from every evaluation task, each shown in 4 random option orders, a reward of one for a plan that passes every check plus 0.5 when the plan equals the plurality plan over the four orders of the task, 500 steps at a learning rate of 10^{-5} with thinking disabled and a completion limit of 128 tokens. A GPU fault interrupted the run, which resumed from the checkpoint at step 200; three predictions were fixed before the first step. Over the 300 batches logged after the resume the mean reward rose from 0.83 to 0.90 while the agreement across orders stayed at 0.33, so the reward was earned by passing more often, not by answering consistently. Table 24 gives the adapter against the base model on the base model's own 1,200 main tasks and in the second environment, both served from the official weights on vLLM. Single generation passes more often at every level (0.490 to 0.800 at $c = 0$, 0.472 to 0.598 at $c = 1$). At $c = 1$ the first listed feasible option is taken in 55.7% [52.9, 58.5] of the slots where it is not the cheapest, against 57.6% [54.8, 60.4] before training, so T1 holds by 1.9 points with overlapping intervals; m after feedback at $c = 1$ is 69.9% [61.9, 77.6] against 36.4% under independence, above the 56.7% of the base model, so T3 holds; and in the second environment ascending price adds 10.0 points [3.5, 16.5] over the default order, against 2.5 [-3.0, 8.0] before training, so T2 fails, as the gain grew instead of shrinking. This is one run with one set of hyperparameters. A stronger recipe may do more; this one raised the pass rate of Qwen3-4B without removing the position bias or the regressions that follow a repair message.

Table 24: Qwen3-4B before and after training on permuted option orders: single generation and feedback on the 1,200 main tasks of Qwen3-4B, and the sorting arms of the second environment with one resource (200 paired tasks per cell). Choices are counted at $c = 1$ in constrained slots whose first listed feasible option is not the cheapest one.

	Base model	Adapter
Single pass, $c = 0$ / 0.5 / 1	0.490 / 0.328 / 0.472	0.800 / 0.610 / 0.598
First listed feasible option at $c = 1$	57.6%	55.7%
Cheapest feasible option at $c = 1$	21.7%	27.2%
m after feedback at $c = 1$ (independence)	56.7% (31.0%)	69.9% (36.4%)
m after feedback at $c = 0$	4.6%	2.9%
Second environment, default / ascending at $c = 0$	0.230 / 0.305	0.330 / 0.510
Second environment, default / ascending at $c = 1$	0.095 / 0.120	0.095 / 0.195
Gain of ascending price at $c = 1$ (points)	$+2.5$ [-3.0, 8.0]	$+10.0$ [3.5, 16.5]

E Limitations

We do not claim that repairs break other constraints in general; with identical prompts m ranges from no trend on Qwen3-8B to 56.7% on Qwen3-4B, and when the revision is written step by step with plan-and-solve prompting no fixed attempt fails another check (Appendix C.2). The first environment has one resource and its requirements are single yes/no columns; the second environment assembles its descriptions from templates, so a program given the phrase lists could still judge whether an option meets a requirement. The GPT models rarely violate such requirements in a single attempt, which holds their single-attempt variance ratio near 1 by construction and limits what the environment can say about concentration. Difficulty matching failed, so coupling and pass rate are partly confounded. On TravelPlanner we have per-constraint outcomes for one model, and the repair round with the failed checks fixes 108 plans over the five samples, so its share still carries an interval of 48.9 to 67.2 per cent. The difficulty controls cover 47.4%, 39.8% and 42.7% of the attempts at $c = 1$ for gpt-4.1, Qwen3-8B and Qwen3-4B and none for gpt-4.1-mini, so they narrow the confound without removing it. The sub-agent mechanism, the difficulty controls, the direct variance-ratio differences and the intervals on m are exploratory, and the preregistered trend test for the variance ratio was invalid. The second environment covers three models, two of them from one family, and Qwen3-4B stays near its floor. Its checklist-on-sorted-options and sampling meth-

ods were added after a pilot, one registered prediction was falsified and another fails on gpt-4.1, and the threshold of the first is stated in two ways (Appendix C.3). Sorting by price alone moves TravelPlanner failures to minimum nights, so sorting helps only on the resources it reads. The TravelPlanner fixes use the benchmark's evaluator as the critic, were run after the main results with one model, and the evaluator's restaurant parser was corrected during these runs (Appendix C.6). The sub-agent result concerns one aggregator, which reads every report and writes the plan without an imposed price rule. The answer to the question about other orders with two resources is a simulation of the reading rule (Appendix D.3), not a model run; graded coupling, 20 and 50 options and the paraphrases were run at $c = 1$ with one resource only. The overlap of price ranges for m stops at two ranks, and it and the message that names the slots were run on the two GPT models only; with 35 to 57 fixed gpt-4.1 attempts the fall of m with the overlap is not separated from no change. On TravelPlanner the coupling of a query is measured on its accommodation only, and the fields extracted from the query text feed the removal step and the repair message, while the evaluator that serves as the critic still reads the annotation. The sorting method needs a resource that a program can read and order; benchmarks whose constraints are physical and temporal (Chen et al., 2026a) or general constraint programs (Shi et al., 2025) have no single scalar to sort by, so the regression rate applies there and the fix does not. We did not run training for permutation robustness (Zheng et al., 2026), iterative selection (Bito et al., 2025), more than two shared resources, or WorldTravel (Wang et al., 2026), whose constraints are tightly coupled. Plan-and-solve prompting, the pipeline with a solver and program-aided planning were run on the two GPT models only, the one model of another family on the tasks of gpt-4.1-mini without difficulty matching, and none of the trained or multi-agent planners named in Appendix C.6 was run. The rewrite of this paper on the structure of another paper changed no number except H3.3, which is re-graded with the corrected evaluator (Appendix C.7).