Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Instruction fragility under hidden operational requirements

Chanho Park
Gwangju Institute of Science and Technology
Jinbae Gong
Gwangju Institute of Science and Technology
Minsu Kim
Gwangju Institute of Science and Technology
Gyeongho Gong
Gwangju Institute of Science and Technology
Woochan Lee
Gwangju Institute of Science and Technology
Yeachan Kwak
Gwangju Institute of Science and Technology
Seongim Choi
schoi1@gist.ac.kr

Gwangju Institute of Science and Technology

Research Article

Keywords: large language models, instruction following, prompt underspecification, hidden requirements, requirement elicitation, AI safety

Posted Date: July 7th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10014579/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Instruction fragility under hidden operational requirements

Chanho Park[1], Jinbae Gong[1], Minsu Kim[1], Gyeongho Gong[1], Woochan Lee[1], Yeachan Kwak[1], and Seongim Choi[1,*]

[1]Gwangju Institute of Science and Technology, Republic of Korea

*Correspondence: schoi1@gist.ac.kr

Abstract

Natural-language instructions to AI systems often omit operational requirements that are known to the user or evaluator but absent from the prompt. We formalize this problem by treating an instruction as denoting a set of feasible executions rather than a single intended outcome. Safety under underspecified execution then requires every satisfying execution to lie inside the user's acceptable set. We derive elementary information-theoretic, cardinality and complexity consequences of this view and test them with a 100-task hidden-requirements benchmark across GPT-5.4, Claude Opus 4.7 and Gemini 3.1 Pro. One-shot prompting strictly satisfied all hidden requirements in 2.7% of 300 trials, whereas full-oracle clarification, which reveals all hidden requirements, reached 64.7%. A negative-control clarification condition achieved 1.7%, indicating that interaction without task-relevant information does not improve strict success. These results identify a controlled failure mode for strict operational compliance: generic prompting and generic interaction do not reliably recover omitted constraints, whereas correctly eliciting and executing those constraints substantially improves performance.

Keywords: large language models; instruction following; prompt underspecification; hidden requirements; requirement elicitation; AI safety;

1 Introduction

Large language models (LLMs) and interactive AI systems are increasingly asked to act on compact natural-language instructions[1–3]. Such instructions are rarely complete. A user asking for a summary may silently expect caveats, a risk label, a maximum length, a source-limit statement, or a particular output format. A developer writing a system prompt may assume that unstated policy requirements are preserved. A safety engineer may expect an objective to be pursued only through acceptable causal pathways. These missing constraints are not incidental: they can determine whether the resulting behavior is acceptable.

A simple example illustrates the issue. The instruction "make me wealthy" does not identify a unique future. It may be satisfied by lawful earnings, a bank error, a fraudulent transfer, a change in currency value, or a change in what the user counts as wealth. The problem is not the folklore of malicious wish-granting. It is that a finite instruction denotes

many feasible executions. If an executor is allowed to choose among those executions without sharing the user's unstated preferences, describing one good execution is not enough. The instruction must rule out unacceptable executions.

This paper studies that problem as *instruction fragility under hidden operational requirements*. We use a minimal denotational framework in which an instruction denotes a set of admissible executions rather than a single intended execution. The framework yields a direct safety condition: an instruction is robust only when its entire denotation lies within the user's acceptable set. This formulation is intentionally simple, but it clarifies why one-shot prompting, generic templates, and generic clarification can fail when task-critical requirements are not transmitted.

We then test this idea using a controlled hidden-requirements benchmark. Each benchmark task contains a visible user prompt and a finite set of hidden requirements known to the evaluator but omitted from the prompt. The hidden requirements include conventional constraints, such as uncertainty statements and next steps; operational constraints, such as word limits and source-limit statements; and private formatting constraints, such as JSON output and exact bullet counts. A strict task pass requires satisfying every hidden requirement.

The benchmark is synthetic, text-only, and largely surface-checkable by design. This makes the omitted requirements explicit and the grading auditable, but it also means the results should be interpreted as evidence for a specific failure mode rather than as a complete estimate of real-world assistant reliability. Within that scope, the empirical pattern is strong. One-shot and generic template prompts rarely satisfy all omitted requirements. Forcing questions helps only modestly. Dialogue turns without task-relevant information do not help. Revealing the correct requirements produces a much larger gain, although full revelation still leaves execution failures.

The paper makes four contributions. First, it presents a compact formal account of hidden operational requirements as set-valued instruction semantics. Second, it derives simple information, exactness, and safe-synthesis consequences that explain why omitted requirements cannot generally be recovered by one-shot execution alone. Third, it introduces a 100-task benchmark that separates generic prompting, negative-control clarification, forced elicitation, budgeted oracle revelation, full oracle revelation, and verification. Fourth, it reports requirement-level and paired-condition analyses across three contemporary LLMs, showing that the largest gains come from correctly eliciting missing requirements rather than from generic interaction.

2 Related work

2.1 Objective misspecification and AI safety

A large body of AI-safety work studies failures that occur when a system optimizes a proxy objective that differs from the user's intended objective. Reward hacking, negative side effects, scalable oversight, and robustness to distributional shift are central examples[4]. Goodhart-style failures explain why a useful measure can cease to track the intended target once optimized[5]. Recent work on reward misspecification and goal misgeneralization shows that systems can learn or pursue objectives that diverge from the intended task even when training performance appears strong[6, 7]. Broader AI-control arguments similarly emphasize the difficulty of specifying objectives completely and correctly in real-world settings[8]. Our focus is complementary. Before optimization or execution begins, a

natural-language instruction may already admit multiple incompatible executions. Hidden operational requirements are one concrete form of this pre-execution underspecification.

2.2 Preference learning, corrigibility, and interaction

Preference-learning and human-feedback methods soften the one-shot specification problem by adding feedback channels[3, 9, 10]. Cooperative inverse reinforcement learning models alignment as a cooperative problem under reward uncertainty[11]; inverse reward design treats a designed reward as evidence about the true objective[12]; and the off-switch game shows why preserving corrective intervention can be instrumentally rational under objective uncertainty[13]. The present study examines a narrower and more testable question: when hidden requirements are omitted from a prompt, how much does performance improve when models ask for them, receive them, or verify against them?

2.3 Partial preferences and preference elicitation

The problem is also related to work on partial preferences and preference elicitation. Planning with partial preference models studies how to act when a user's preference trade-offs are incompletely specified, and preference-elicitation work studies how interactive queries can recover information that is not available in the initial specification[14, 15]. Hidden operational requirements are narrower than full preference orderings: they are finite, task-local constraints on an answer. That restriction makes it possible to compare elicitation and execution protocols with deterministic graders.

2.4 Formal requirements and specification languages

Formal methods and requirements engineering have long emphasized that informal requirements are often incomplete, inconsistent, or written at the wrong level of abstraction[16]. Controlled natural-language specifications, temporal and structured specification patterns, and reward machines illustrate ways of using stronger languages or formalisms to constrain admissible behavior[17, 18]. Our denotational model abstracts over the particular language. The question is not which formalism is best, but what any one-shot specification must achieve to be robust: it must exclude all unacceptable executions, not merely allow one acceptable execution.

2.5 Prompt underspecification, benchmarking, and clarification

Semantic underspecification is a general property of language and has become practically important in LLM prompting[19, 20]. Behavioral testing and dynamic benchmarking emphasize that aggregate accuracy can hide systematic failures[21, 22]. InFoBench decomposes instruction following into requirement-level criteria[23], and clarification benchmarks study whether models ask useful questions under ambiguous requests[24, 25]. Our benchmark differs by fixing the omitted requirements in advance, making them explicit to the evaluator, and comparing protocols that reveal no hidden information, reveal model-selected hidden information, reveal a budgeted oracle subset, or reveal the full hidden requirement set. This design estimates the value of correctly elicited operational constraints rather than only measuring whether an answer is generally plausible.

2.6 Incomplete contracts and hidden contingencies

The problem also resembles incomplete contracting. A compact contract or instruction cannot enumerate all future contingencies, leaving residual discretion to the executor[26, 27]. Prompt templates play a similar role to reusable contract clauses: they compress common requirements, but they do not automatically transmit task-specific private constraints. Our empirical results test this analogy in the limited setting of text-generation tasks with hidden operational requirements.

3 Methodology

3.1 Overview

The study combines a formal specification model with a controlled LLM benchmark. The formal component defines what it would mean for an instruction to be robust under hidden requirements. The benchmark component operationalizes that model with visible prompts, hidden requirement sets, deterministic graders, and protocol comparisons. The statistical component compares protocols at the task level, requirement level, and paired task–model level.

3.2 Formal problem formulation

Let $\mathcal{H}$ be a nonempty set of feasible post-instruction histories and let $h_0 \in \mathcal{H}$ be a baseline history. Let $\mathcal{L}$ be an instruction language and let B be a background theory containing shared facts, conventions, legal assumptions, system instructions, templates, and interpretive norms. A one-shot instruction $s \in \mathcal{L}$ has set-valued semantics

$$[\![s]\!]_B \subseteq \mathcal{H},$$

where each $h \in [\![s]\!]_B$ is a feasible execution consistent with the instruction under B. Unless otherwise stated, we restrict infimum and supremum quantities to satisfiable instructions, so $[\![s]\!]_B \neq \emptyset$.

The user's evaluation is represented by a utility function $u : \mathcal{H} \to \mathbb{R}$ and a side-effect pseudometric $d : \mathcal{H} \times \mathcal{H} \to \mathbb{R}_{\geq 0}$ relative to h_0. For a utility threshold τ and side-effect budget η, the acceptable set is

$$A_{\tau,\eta} = \{h \in \mathcal{H} : u(h) \geq \tau \text{ and } d(h, h_0) \leq \eta\}.$$

We define robust value, best-case value, semantic spread, and side-effect radius as

$$V_B^-(s) = \inf_{h \in [\![s]\!]_B} u(h), \qquad V_B^+(s) = \sup_{h \in [\![s]\!]_B} u(h),$$

$$\Delta_B(s) = V_B^+(s) - V_B^-(s), \qquad S_B(s) = \sup_{h \in [\![s]\!]_B} d(h, h_0).$$

The safe-specification complexity for threshold τ and budget η is

$$C_B(\tau, \eta) = \min\{|s| : [\![s]\!]_B \subseteq A_{\tau,\eta}\},$$

with value ∞ if no safe instruction exists.

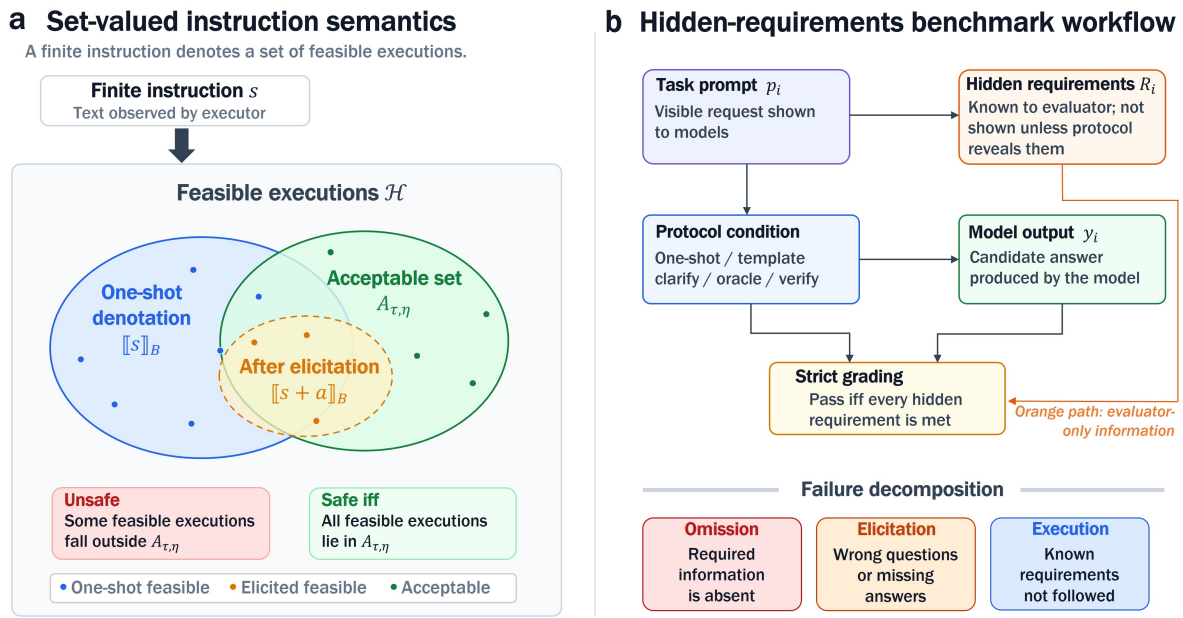


Fig. 1 **Set-valued semantics and benchmark workflow. a**, A one-shot instruction s denotes a set of feasible executions $[\![s]\!]_B$ under background assumptions B. The acceptable set $A_{\tau,\eta}$ contains histories with sufficient utility and bounded side effects. Safety requires the whole denotation to be contained in the acceptable set. Clarification can shrink the denotation by eliciting missing constraints. **b**, The hidden-requirements benchmark instantiates this idea with visible task prompts, omitted operational requirements, prompting protocols, model outputs, and rule-based grading.

Proposition 1 (Adversarial safety criterion)**.** *If an executor may choose any history in* $[\![s]\!]_B$*, then every admissible execution of* s *is acceptable if and only if*

$$[\![s]\!]_B \subseteq A_{\tau,\eta}.$$

Equivalently, $V_B^-(s) \geq \tau$ *and* $S_B(s) \leq \eta$*.*

Thus a prompt is not robust because it has a good interpretation; it is robust only when it lacks unacceptable interpretations. Fig. 1 summarizes this view and the benchmark workflow. The same structure applies to prompts, reward descriptions, contracts, and natural-language requirements.

3.3 Theoretical analyses

The set-valued view clarifies why hidden requirements are not solved by one-shot prompting alone. If a finite alphabet has size q, then fewer than $q^{\ell+1}$ instructions of length at most ℓ exist. Distinguishing among N live candidate intentions therefore requires $\Omega(\log_q N)$ symbols in the worst case. The same counting argument implies that any binary clarification protocol needs at least $\lceil \log_2 N \rceil$ yes/no questions to identify one target among N distinguishable candidates[28, 29].

Exact one-shot specification is also limited by cardinality. If $\mathcal{L}$ is countable and $\mathcal{H}$ is infinite, only countably many subsets of $\mathcal{H}$ can be exact denotations of finite instructions, whereas $\mathcal{P}(\mathcal{H})$ is uncountable. Arbitrary intention sets therefore cannot be expected to have exact one-shot descriptions without strong shared structure, approximation, or

interaction. Finally, safe synthesis can be computationally hard even in finite settings. Suppose a desiderata set D must be protected and each available clause c covers a subset $G(c) \subseteq D$. The shortest safe specification asks for at most k clauses whose union covers D, which is exactly Set Cover and is NP-complete[30].

These elementary results have one concrete empirical prediction: strict success should fall as hidden requirements accumulate, because passing a task requires satisfying every omitted requirement. Templates can help by adding shared structure, but targeted elicitation should help most when it reveals the missing operational constraints.

3.4 Benchmark construction and requirement taxonomy

We constructed a 100-task benchmark. Each task contains a visible user prompt and a set of hidden requirements known to the evaluator but omitted from the prompt. The task set is balanced across hidden-requirement counts: 25 tasks each contain 2, 3, 4, and 5 hidden requirements. The benchmark uses 20 topical domains, including product updates, policy memos, research summaries, decision analysis, incident response, student feedback, travel planning, meeting notes, email replies, comparisons, debugging, hiring, financial notes, health administration, data interpretation, education planning, legal plain-language explanation, operations, security awareness, and market analysis.

The ten requirement types fall into three groups. *Conventional implicit requirements* include assumptions, uncertainty or caveats, trade-offs, risk level, and next steps. These may plausibly be inferred from context or professional writing norms. *Operational requirements* include word limits and source or verification limits. These are common in deployed settings but often private to the evaluator unless stated. *Private formatting requirements* include valid JSON and exact bullet count. These are deliberately difficult to infer without disclosure and illustrate the information-theoretic edge case: a private constraint omitted from the prompt cannot be guaranteed by a one-shot executor. The distribution of hidden requirement types across the benchmark is reported in Table 1.

Table 1 Distribution of hidden requirement types across the benchmark. Counts are pooled across the 300 task–model pairs per condition. The 100 tasks contain 350 unique task-level hidden requirements; evaluating three models yields 1,050 requirement occurrences per condition.

Requirement type	Occurrences
Assumptions	111
Avoid absolutes	111
Bullet count	66
JSON	96
Max words	111
Next steps	108
Risk level	126
Source limits	105
Trade-offs	96
Uncertainty	120

3.5 Prompting protocols

The conditions were designed to separate longer prompting, generic interaction, elicitation quality, information budget, and execution quality. The *one-shot* condition gives only the visible task prompt. The *template* condition adds a generic structured system prompt but reveals no task-specific hidden requirements. The *negative-control clarification* condition forces menu-based questions but responds that no special hidden constraints apply. This is not a neutral interaction control: it supplies negative evidence and may suppress inference. We use it only to test whether dialogue turns without task-relevant information improve strict success.

The *forced clarify* condition forces up to four menu questions and truthfully reveals matching hidden requirements. *Verify visible* uses the same truthful forced-clarification question phase and then adds a visible checklist and self-check before the final answer. *Verify clean* uses the same truthful forced-clarification question phase but performs verification internally and outputs only the final answer. *Budgeted oracle* reveals up to four true hidden requirements. *Full oracle* reveals all hidden requirements. *Full oracle + verify* reveals all hidden requirements and adds clean internal verification. Clarification questions were restricted to a fixed menu containing FORMAT_JSON, MAX_WORDS, BULLET_COUNT, ASSUMPTIONS, UNCERTAINTY, RISK_LEVEL, AVOID_ABSOLUTES, NEXT_STEPS, TRADEOFFS, and SOURCE_LIMITS.

3.6 Model suite and API settings

We evaluated the API model identifiers `gpt-5.4`, `claude-opus-4-7`, and `gemini-3.1-pro-preview`. Runs were conducted between 26 April and 12 May 2026 UTC. Each model was run once on all 100 tasks and nine conditions, producing 2,700 task–model–condition observations. The maximum output token budget was 8,192. For GPT-5.4 we used high reasoning effort when supported. We omitted the temperature parameter from all API calls because some current endpoints reject explicit temperature settings; all other sampling parameters were left at provider defaults. All clarification protocols used a budget of $k = 4$ questions except full-oracle protocols, which revealed all hidden requirements. API evaluations are time-sensitive and may not be exactly reproducible if providers update model weights or serving infrastructure.

3.7 Evaluation metrics and statistical analysis

A task is a tuple

$$T_i = (p_i, R_i, \{G_{ir} : r \in R_i\}),$$

where p_i is the visible prompt, R_i is the hidden requirement set, and G_{ir} is a deterministic grader for requirement r. A model m under condition c produces an output $Y_{i,m,c}$. The strict pass indicator is

$$\mathrm{Pass}(i, m, c) = \prod_{r \in R_i} G_{ir}(Y_{i,m,c}),$$

where each grader returns 0 or 1. Requirement-level success for a requirement type r under condition c is computed over all task–model pairs containing that requirement type:

$$\mathrm{ReqSucc}(r, c) = \frac{1}{N_{r,c}} \sum_{(i,m):r \in R_i} G_{ir}(Y_{i,m,c}).$$

Strict task pass is therefore the empirical analogue of the formal safety condition $[\![s]\!]_B \subseteq A_{\tau,\eta}$: if even one hidden requirement is violated, the task fails.

The rule-based grader checks JSON validity and required keys, maximum word count, exact bullet count, explicit assumption language, uncertainty or caveat language, explicit low/medium/high risk level, banned absolute terms, next-step indicators, trade-off indicators, and source-limit statements. The benchmark was designed around mechanically checkable constraints to make the grading auditable.

We report binomial proportions with descriptive 95% Wilson intervals. Because each task–model pair appears under every condition, protocol comparisons use exact two-sided McNemar tests on discordant pairs, with Holm correction for the reported contrasts and $\alpha = 0.05$. We also fit a logistic regression with condition, model, and hidden-requirement count as predictors and cluster-robust standard errors by task. Requirement-level analyses use the same raw outputs but treat each hidden requirement occurrence as one row. A stratified manual audit of 120 outputs compared human strict-pass labels against the automatic grader. All audited labels agreed with the automatic labels; the audit should be interpreted as validation of the rule implementation for these objective hidden requirements, not as evidence that all forms of semantic usefulness are automatically gradable.

4 Results

4.1 Correctly elicited requirements drive strict success

Strict pass equals 1 only if every hidden requirement in a task is satisfied. Table 2 and Fig. 2 show the pooled results. One-shot prompting passed 8 of 300 trials (2.7%, 95% Wilson interval 1.4–5.2%). A generic template passed 22 of 300 trials (7.3%, 4.9–10.9%). Negative-control clarification performed similarly to one-shot (5 of 300; 1.7%, 0.7–3.8%), showing that interaction without task-relevant information does not improve strict success. Forced clarification passed 24 of 300 trials (8.0%, 5.4–11.6%). Visible verification passed 21 of 300 trials (7.0%, 4.6–10.5%), and clean verification passed 22 of 300 trials (7.3%, 4.9–10.9%).

Revealing true hidden requirements produced a qualitatively larger gain. Budgeted oracle reached 162 of 300 trials (54.0%, 48.3–59.6%). Full oracle reached 194 of 300 trials (64.7%, 59.1–69.9%), and full oracle with clean verification reached 184 of 300 trials (61.3%, 55.7–66.7%). The gap between forced clarification and budgeted oracle indicates that models often fail to ask the right questions. The budgeted–full oracle contrast is most directly interpretable for five-requirement tasks, where the four-requirement disclosure budget necessarily omits at least one constraint. For tasks with four or fewer hidden requirements, both oracle protocols can in principle reveal all requirements, so residual differences should be interpreted as prompt- and generation-level variation rather than a pure information-budget effect. The gap between full oracle and 100% shows that even known requirements can be executed incorrectly.

Model-level results show the same structure with different absolute levels (Table 3 and Fig. 3). Under full oracle, GPT-5.4 passed 78 of 100 tasks, Gemini 3.1 Pro passed 68 of 100 tasks and Claude Opus 4.7 passed 48 of 100 tasks. Because oracle performance still varied across models, the pooled estimates should be read as protocol-level effects across the evaluated systems rather than as a single estimate of frontier-model capability.

Table 2 Strict pass rate by condition. Intervals are descriptive 95% Wilson binomial intervals.

Condition	Trials	Passes	Pass rate	95% CI
One-shot	300	8	2.7%	1.4–5.2%
Template	300	22	7.3%	4.9–10.9%
Negative-control clarify	300	5	1.7%	0.7–3.8%
Forced clarify	300	24	8.0%	5.4–11.6%
Verify visible	300	21	7.0%	4.6–10.5%
Verify clean	300	22	7.3%	4.9–10.9%
Budgeted oracle	300	162	54.0%	48.3–59.6%
Full oracle	300	194	64.7%	59.1–69.9%
Full oracle + verify	300	184	61.3%	55.7–66.7%

Table 3 Model-level strict pass rates.

Model	Condition	Passes/trials	Pass rate
GPT-5.4	One-shot	4/100	4.0%
GPT-5.4	Template	4/100	4.0%
GPT-5.4	Negative-control clarify	1/100	1.0%
GPT-5.4	Forced clarify	10/100	10.0%
GPT-5.4	Verify visible	11/100	11.0%
GPT-5.4	Verify clean	10/100	10.0%
GPT-5.4	Budgeted oracle	59/100	59.0%
GPT-5.4	Full oracle	78/100	78.0%
GPT-5.4	Full oracle + verify	70/100	70.0%
Claude Opus 4.7	One-shot	3/100	3.0%
Claude Opus 4.7	Template	8/100	8.0%
Claude Opus 4.7	Negative-control clarify	2/100	2.0%
Claude Opus 4.7	Forced clarify	7/100	7.0%
Claude Opus 4.7	Verify visible	6/100	6.0%
Claude Opus 4.7	Verify clean	6/100	6.0%
Claude Opus 4.7	Budgeted oracle	44/100	44.0%
Claude Opus 4.7	Full oracle	48/100	48.0%
Claude Opus 4.7	Full oracle + verify	51/100	51.0%
Gemini 3.1 Pro	One-shot	1/100	1.0%
Gemini 3.1 Pro	Template	10/100	10.0%
Gemini 3.1 Pro	Negative-control clarify	2/100	2.0%
Gemini 3.1 Pro	Forced clarify	7/100	7.0%
Gemini 3.1 Pro	Verify visible	4/100	4.0%
Gemini 3.1 Pro	Verify clean	6/100	6.0%
Gemini 3.1 Pro	Budgeted oracle	59/100	59.0%
Gemini 3.1 Pro	Full oracle	68/100	68.0%
Gemini 3.1 Pro	Full oracle + verify	63/100	63.0%

The result is not driven only by private formatting constraints. When valid-JSON and exact-bullet requirements are excluded from the strict-pass calculation, one-shot increases from 2.7% to 6.3%, template to 14.0%, forced clarification to 13.7%, budgeted

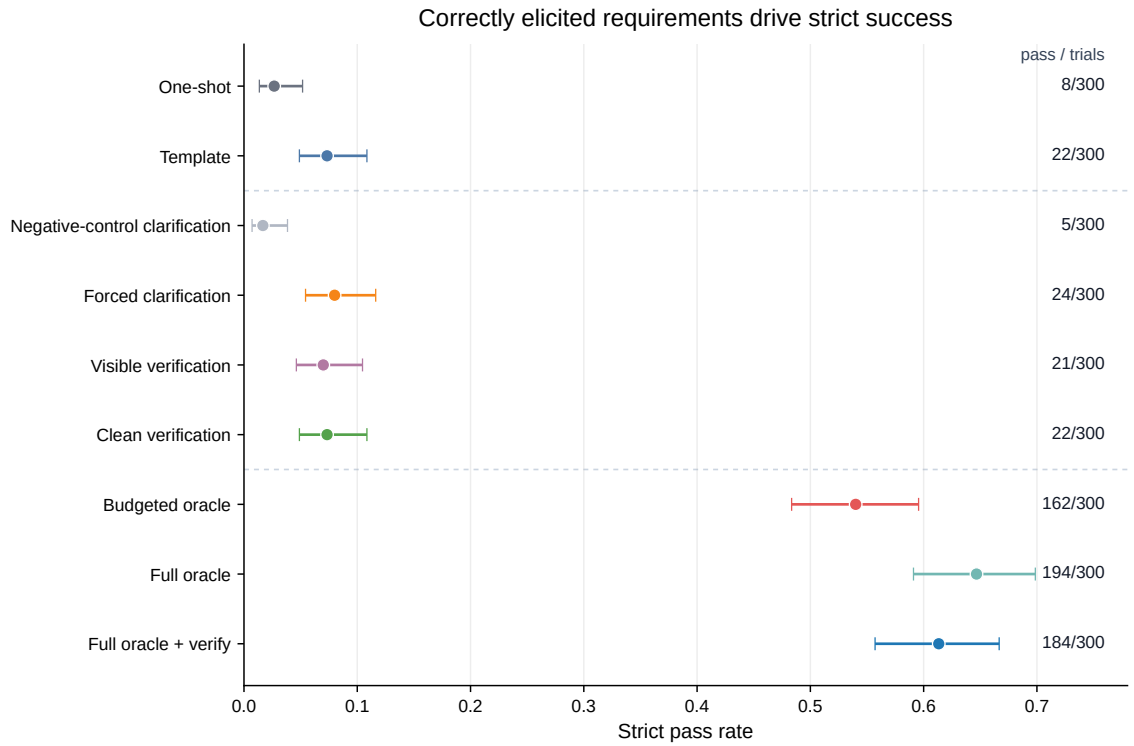


Fig. 2 **Correctly elicited requirements drive strict success.** Points show pooled strict pass rates across 300 trials per condition. Horizontal error bars are descriptive 95% Wilson binomial intervals. Numbers indicate pass counts. Low-information conditions remain near zero or in the single digits, whereas budgeted and full oracle conditions improve substantially.

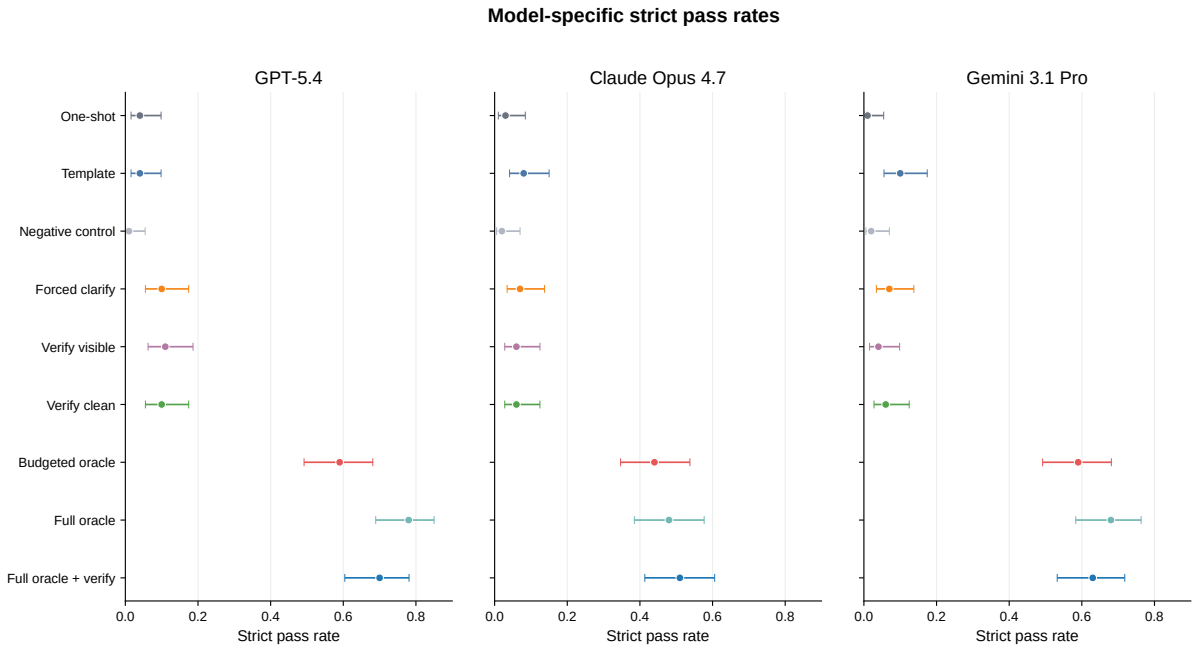


Fig. 3 Model-specific strict pass rates with 95% Wilson intervals

oracle to 56.3% and full oracle to 67.0% (Table 4). Thus the qualitative gap between low-information protocols and correctly elicited requirements persists even after removing

the most obviously private formatting requirements.

Table 4 Sensitivity analyses by requirement subset. Private formatting means valid JSON and exact bullet count. Operational non-private means word limits and source or verification limits.

Condition	Trials	Passes	Pass rate
Private formatting excluded			
One-shot	300	19	6.3%
Template	300	42	14.0%
Negative-control clarify	300	17	5.7%
Forced clarify	300	41	13.7%
Verify visible	300	32	10.7%
Verify clean	300	35	11.7%
Budgeted oracle	300	169	56.3%
Full oracle	300	201	67.0%
Full oracle + verify	300	190	63.3%
Conventional requirements only			
One-shot	291	34	11.7%
Template	291	93	32.0%
Negative-control clarify	291	35	12.0%
Forced clarify	291	71	24.4%
Verify visible	291	71	24.4%
Verify clean	291	60	20.6%
Budgeted oracle	291	203	69.8%
Full oracle	291	225	77.3%
Full oracle + verify	291	208	71.5%
Operational non-private only			
One-shot	168	15	8.9%
Template	168	11	6.5%
Negative-control clarify	168	13	7.7%
Forced clarify	168	36	21.4%
Verify visible	168	17	10.1%
Verify clean	168	47	28.0%
Budgeted oracle	168	100	59.5%
Full oracle	168	126	75.0%
Full oracle + verify	168	128	76.2%
Private formatting only			
One-shot	162	5	3.1%
Template	162	4	2.5%
Negative-control clarify	162	5	3.1%
Forced clarify	162	75	46.3%
Verify visible	162	56	34.6%
Verify clean	162	95	58.6%
Budgeted oracle	162	146	90.1%
Full oracle	162	152	93.8%
Full oracle + verify	162	151	93.2%

4.2 Paired contrasts decompose the failure modes

Because each task-model pair appears under every condition, we can compare protocols pairwise. Exact two-sided McNemar tests were used on paired strict-pass outcomes, with Holm correction across the reported contrasts. We used $\alpha = 0.05$ for statistical tests. Template improved over one-shot by 4.7 percentage points (15 discordant wins versus 1 loss; Holm-adjusted $P = 0.002$). Forced clarification improved over the negative-control condition by 6.3 percentage points (20 versus 1; Holm-adjusted $P = 1.3 \times 10^{-4}$), showing that truthful information gain matters beyond the act of asking questions.

The largest contrast was between budgeted oracle and forced clarification: +46.0 percentage points (141 wins versus 3 losses; Holm-adjusted $P = 3.1 \times 10^{-37}$). Full oracle further improved over budgeted oracle by +10.7 percentage points (54 versus 22; Holm-adjusted $P = 0.002$). Full oracle improved over one-shot by +62.0 percentage points (188 versus 2; Holm-adjusted $P = 2.1 \times 10^{-52}$). Clean verification did not produce a reliable gain after forced clarification or full oracle; it was slightly below forced clarification by 0.7 percentage points and below full oracle by 3.3 percentage points, with both contrasts non-significant after correction. The full set of paired strict-pass contrasts is reported in Table 5.

Table 5 Paired strict-pass contrasts. Discordant pairs report A-passed/B-failed versus A-failed/B-passed. Holm-adjusted P values are from exact two-sided McNemar tests.

Contrast	Delta	Discordant	Holm P
Template - one-shot	+4.7%	15 vs 1	0.002
Forced clarify - negative control	+6.3%	20 vs 1	1.3×10^{-4}
Budgeted oracle - forced clarify	+46.0%	141 vs 3	3.1×10^{-37}
Full oracle - budgeted oracle	+10.7%	54 vs 22	0.002
Full oracle - forced clarify	+56.7%	172 vs 2	1.0×10^{-47}
Full oracle - one-shot	+62.0%	188 vs 2	2.1×10^{-52}
Full oracle + verify - full oracle	-3.3%	31 vs 41	0.866
Verify clean - forced clarify	-0.7%	10 vs 12	1.000
Verify clean - verify visible	+0.3%	10 vs 9	1.000

Fig. 4 summarizes these contrasts. We interpret them as a decomposition into omitted-information failure, elicitation failure, information-budget failure and execution failure. The remaining failure under full oracle is especially important: even when all hidden requirements are known, models still do not always execute the full specification.

The same pattern appears at the question-selection level. In forced clarification, models asked almost exactly four menu questions per task, but the questions recalled only 37.4% of the hidden requirements and had 32.8% precision. Recall was high for next steps (78.7%) and word limits (65.8%), but very low for source-limit statements (1.0%) and risk labels (22.2%). Clean verification did not solve this elicitation problem: its question recall was 39.0% and precision 34.1% (Tables 6 and 7). The budgeted-oracle gap therefore reflects a concrete failure to identify the relevant latent constraints, not merely a failure to answer after asking.

A clustered logistic regression with condition, model and hidden-requirement count as predictors gave the same qualitative pattern. Relative to one-shot, full oracle had an odds ratio of 107.2 for strict pass (95% CI 41.2–279.0), full oracle plus verification 90.5 (35.9–228.0), and budgeted oracle 63.3 (25.6–156.7). Each additional hidden requirement

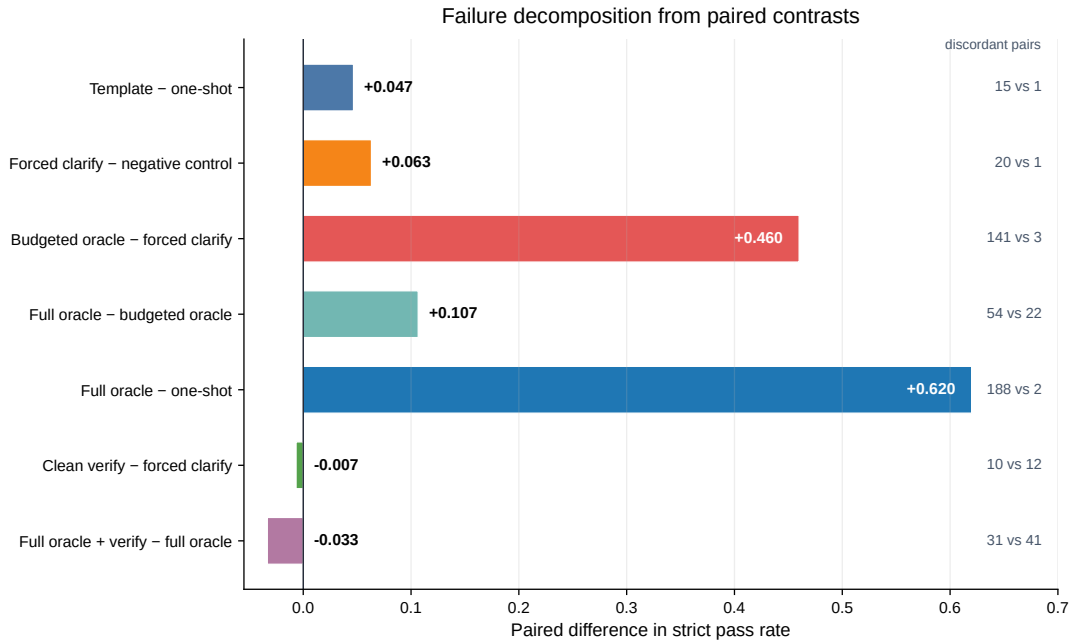


Fig. 4 **Failure decomposition from paired contrasts.** Bars show paired differences in strict pass rate across the same task-model pairs. Parentheses give the number of discordant pairs in which condition A passed and condition B failed versus the reverse. The largest gains come from revealing the correct requirements, not from generic prompting or verification.

Table 6 Question-selection performance in menu-based clarification conditions. Recall is the fraction of hidden requirements selected by the model's questions; precision is the fraction of asked questions that matched a true hidden requirement.

Condition	Avg. questions	Hidden-requirement recall	Question precision
Negative-control clarify	4.00	39.8%	34.8%
Forced clarify	3.99	37.4%	32.8%
Verify visible	3.98	38.9%	34.2%
Verify clean	4.00	39.0%	34.1%

Table 7 Forced-clarification question recall by requirement type.

Requirement type	Hidden occurrences	Asked when hidden	Recall
Next steps	108	85	78.7%
Max words	111	73	65.8%
Bullet count	66	38	57.6%
JSON	96	41	42.7%
Uncertainty	120	44	36.7%
Trade-offs	96	30	31.2%
Assumptions	111	31	27.9%
Risk level	126	28	22.2%
Avoid absolutes	111	22	19.8%
Source limits	105	1	1.0%

reduced the odds of strict pass by about half (odds ratio 0.47, 95% CI 0.37–0.60). Because baseline strict pass is very low, the paired risk differences in Fig. 4 are the primary effect-size summary; odds ratios are reported as a complementary model-based analysis (Table 8).

Table 8 Clustered logistic regression summary. Odds ratios are from a logistic regression with condition, model and hidden-requirement count as predictors, with standard errors clustered by task.

Term	Odds ratio	95% CI	P
Template vs one-shot	3.0	1.7–5.3	2.2×10^{-4}
Negative-control clarify vs one-shot	0.6	0.2–1.6	0.302
Forced clarify vs one-shot	3.3	1.5–7.4	0.004
Verify visible vs one-shot	2.8	1.3–6.2	0.010
Verify clean vs one-shot	3.0	1.5–6.1	0.002
Budgeted oracle vs one-shot	63.3	25.6–156.7	2.9×10^{-19}
Full oracle vs one-shot	107.2	41.2–279.0	1.0×10^{-21}
Full oracle + verify vs one-shot	90.5	35.9–228.0	1.2×10^{-21}
Each additional hidden requirement	0.47	0.37–0.60	2.8×10^{-9}

4.3 Strict success compounds across hidden requirements

Fig. 5 shows strict pass rate by hidden-requirement count. Low-information protocols collapse rapidly. One-shot passes 8.0% of two-requirement tasks, 0% of three- and five-requirement tasks and 2.7% of four-requirement tasks. Template passes 20.0% of two-requirement tasks but only 1.3% of five-requirement tasks. Forced clarification passes 22.7% of two-requirement tasks and 0% of five-requirement tasks. By contrast, full oracle remains above 50% even at five hidden requirements, indicating that revealing all requirements prevents the multiplicative collapse from being complete, but does not eliminate execution failures (Table 9).

The multiplicative structure is expected. If each hidden requirement has a condition-specific satisfaction probability $p_{c,r}$, then a task with requirement set R_i has approximate strict pass probability

$$\widehat{P}(\text{strict pass} \mid i, c) = \prod_{r \in R_i} \widehat{p}_{c,r}.$$

Using empirical requirement-level success rates, this multiplicative baseline captures the qualitative collapse of low-information protocols, but it deviates for some oracle conditions where requirement composition and cross-requirement dependencies matter (Fig. 6). The baseline should therefore be read as a descriptive check rather than an independent predictive model. It links the formal view to the benchmark: strict pass is a conjunction over hidden requirements, so small per-requirement failure probabilities compound.

4.4 Requirement types differ sharply

Requirement-level success varies by constraint type (Fig. 7). Some requirements are often satisfied without disclosure: avoiding absolute language and acknowledging uncertainty

Table 9 Oracle-style strict pass rates by hidden-requirement count.

Hidden requirement count	Trials	Passes	Pass rate
Budgeted oracle			
2	75	60	80.0%
3	75	49	65.3%
4	75	39	52.0%
5	75	14	18.7%
Full oracle			
2	75	59	78.7%
3	75	47	62.7%
4	75	50	66.7%
5	75	38	50.7%
Full oracle + verify			
2	75	61	81.3%
3	75	46	61.3%
4	75	42	56.0%
5	75	35	46.7%

have high success even in one-shot settings. Others are rarely recovered unless revealed. Valid JSON is essentially unrecovered by one-shot and template prompting but reaches 100% under full oracle. Source-limit statements and explicit risk labels remain difficult even after full disclosure, with full-oracle success rates of 63.8% and 63.5%, respectively (Table 10). Word limits, next steps, assumptions, uncertainty and trade-offs improve strongly under full-information protocols. These differences support the taxonomy above: conventional requirements may be partially inferred from writing norms, whereas private formatting and operational constraints generally require explicit communication.

Table 10 Full-oracle failure counts by requirement type. A failure occurrence means that the requirement was revealed but not satisfied in the final answer.

Requirement type	Occurrences	Failure occurrences	Failure rate
Risk level	126	46	36.5%
Source limits	105	38	36.2%
Bullet count	66	10	15.2%
Assumptions	111	10	9.0%
Trade-offs	96	6	6.2%
Max words	111	5	4.5%
Avoid absolutes	111	3	2.7%
Next steps	108	1	0.9%
Uncertainty	120	1	0.8%
JSON	96	0	0.0%

We manually audited a stratified sample of 120 outputs across models and conditions. The manual strict-pass labels agreed with the rule-based grader on all audited examples. This high agreement is expected because the benchmark intentionally uses mechanically checkable requirements such as JSON validity, word limits, bullet counts and explicit

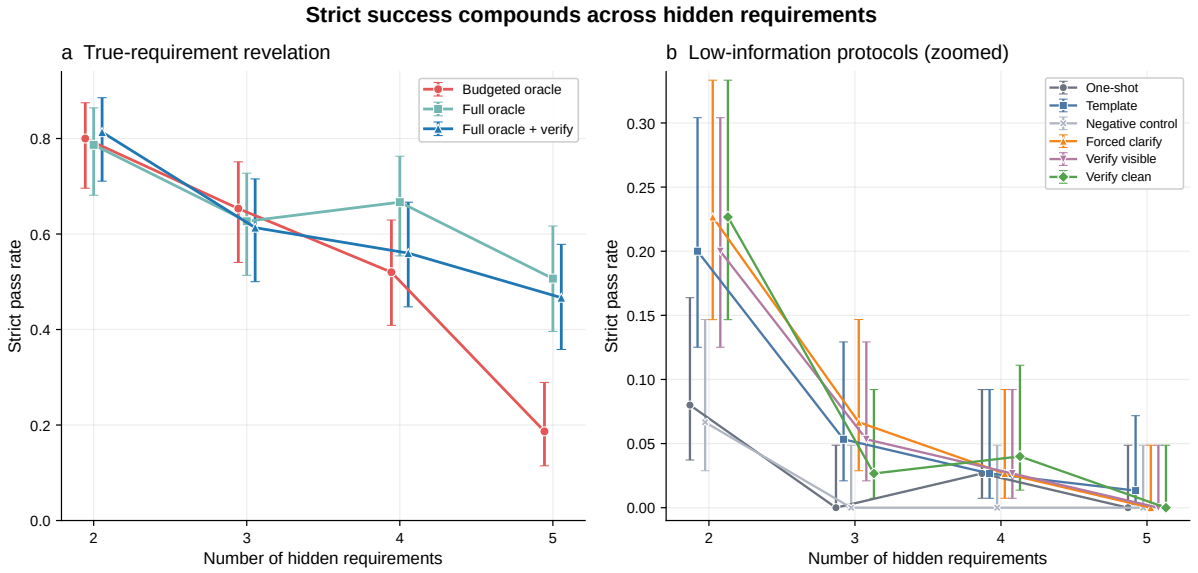


Fig. 5 **Strict success compounds across hidden requirements.** **a**, Oracle-style protocols that reveal true requirements. **b**, Low-information protocols, plotted on a zoomed vertical axis. Points show strict pass rate by hidden-requirement count, with 95% Wilson intervals. Each count has 75 trials per condition.

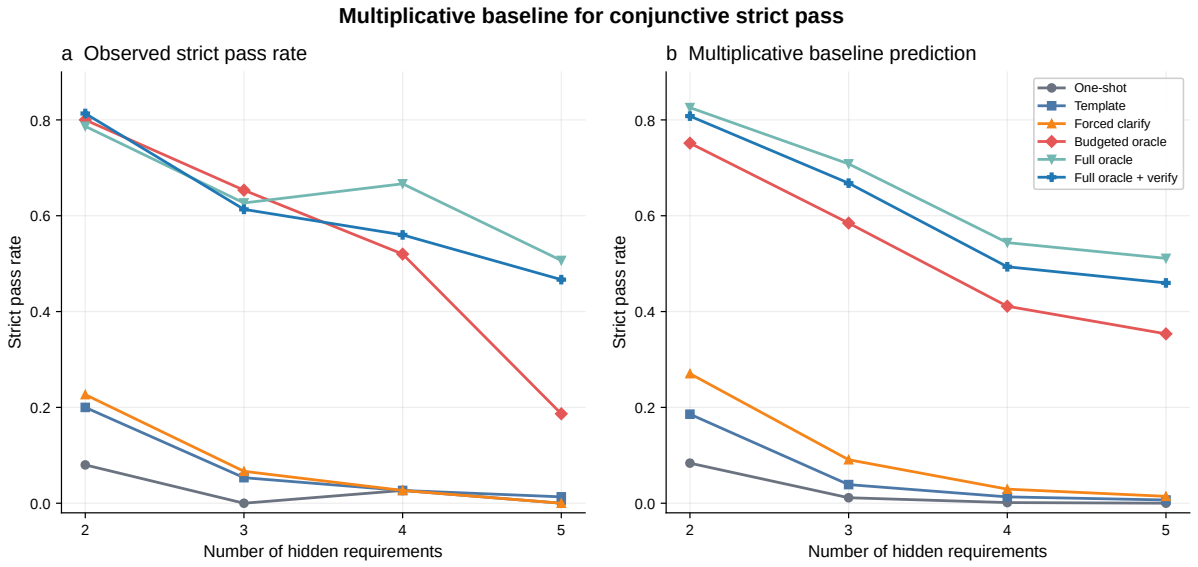


Fig. 6 Observed strict pass rates and multiplicative baseline predictions. The baseline captures the qualitative collapse of low-information protocols but deviates for some oracle conditions, suggesting requirement composition and cross-requirement dependencies

labels. The audit should therefore be interpreted as validation of the implemented grading rules for this benchmark, not as evidence that all forms of semantic usefulness can be automatically graded.

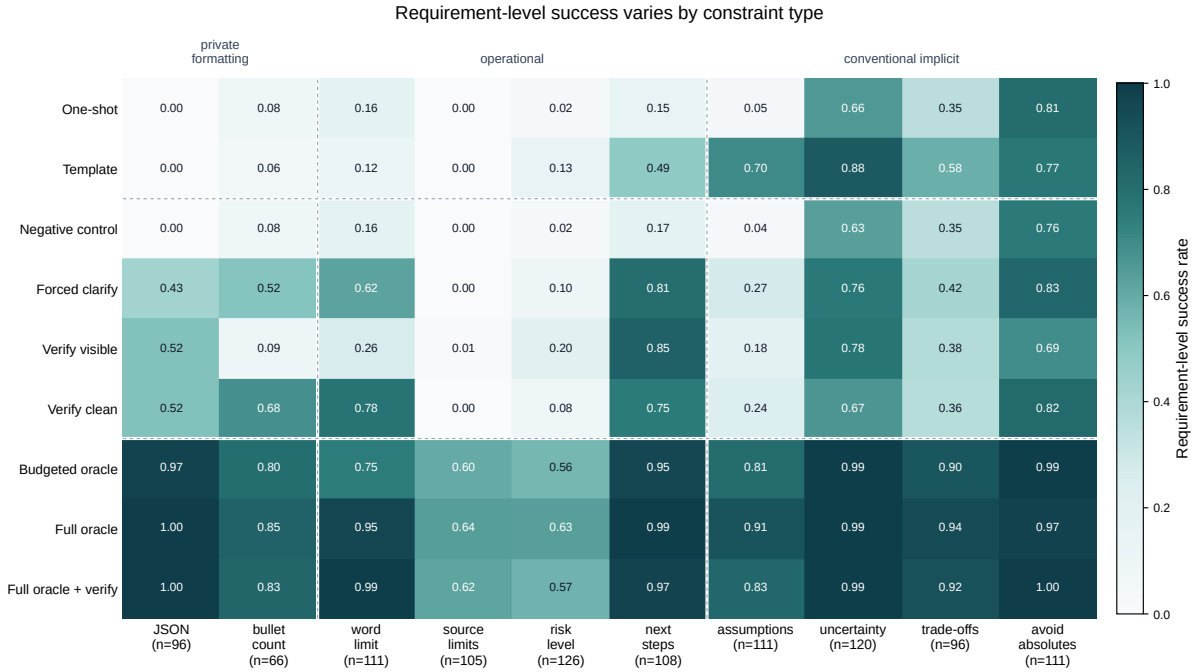


Fig. 7 **Requirement-level success varies by constraint type.** Cells show the fraction of requirement occurrences satisfied by each condition; n values denote pooled requirement occurrences across the 300 task–model pairs per condition. Conventional requirements such as uncertainty and avoidance of absolutes are comparatively easy; private formatting and source-limit constraints are difficult unless revealed and still imperfectly executed.

5 Discussion

5.1 Main interpretation

The main result is not that current models are generally poor instruction followers, and the benchmark is not a test of model clairvoyance. Rather, it is a diagnostic of communication failure: omitted operational requirements create a distinct specification failure mode. Generic structure and generic questioning provide only small gains. Correctly eliciting the missing constraints produces a much larger improvement. But even full elicitation does not yield perfect execution, revealing a second problem: known-constraint compliance.

This distinction matters for the design of AI systems. Many deployed workflows treat clarification as optional: the model may ask questions if it thinks they are necessary. Our results suggest that this is insufficient. The key missing capability is targeted requirement elicitation: identifying which operational constraints are likely to be latent, asking high-value questions, storing the answers as explicit requirement state, and checking the final output against that state. In our benchmark, the negative-control clarification condition performed no better than one-shot prompting, and forced clarification remained far below the budgeted oracle. The bottleneck is not the existence of a dialogue turn but the quality of the information elicited and integrated. This is consistent with the broader ML-systems lesson that untracked dependencies and assumptions can become hidden technical debt in deployed systems[31].

5.2 Implications for prompt templates and verification

The study also reframes prompt templates. Templates are useful because they compress shared requirements into a reusable structure, much as contract clauses compress contingencies. But templates are not a substitute for task-specific requirements. They improved strict pass only slightly over one-shot prompting in our controlled setting. This aligns with prompt underspecification work showing that models may infer omitted requirements but do so inconsistently across prompts and model changes[20]. Our benchmark isolates a stricter version of the same problem: private operational requirements cannot be guaranteed unless communicated.

The formal results should be read accordingly. The adversarial safety criterion is almost definitional, but it is useful because it changes the object of analysis. The question is not whether a prompt has a good completion, but whether every completion consistent with the prompt is acceptable. The counting, cardinality, and set-cover observations are likewise elementary. Their role is to explain why finite one-shot descriptions, templates, and clarification protocols face information and search costs. The empirical contribution is to show that these costs appear in a controlled hidden-requirements benchmark for contemporary LLMs.

5.3 Limitations and future work

The scope of our benchmark is bounded by its synthetic, text-only, and surface-checkable nature, precluding the measurement of long-horizon agent behavior, non-textual environments, subjective quality, or the social negotiation of preferences. Notably, we intentionally incorporated hidden requirements—such as strict JSON formatting and exact bullet counts—to test whether omitted operational constraints could be guaranteed without prior communication. These results support the design principle that strict operational compliance benefits from explicit requirement-state elicitation, faithful integration of elicited constraints, and verification against the resulting requirement state, rather than relying only on longer prompts or generic clarification. Additionally, while utilizing a fixed menu for clarification guarantees reproducibility, it does not fully capture the dynamics of free-form human dialogue. Finally, as API-based evaluations are inherently time-sensitive, the absolute performance metrics reported here may shift alongside future model updates.

6 Conclusion

We introduced a formal and empirical account of instruction fragility under hidden operational requirements. The formal analysis treats an instruction as a set of admissible executions and identifies robust safety with containment of that set inside an acceptable set. The empirical benchmark translates this account into a controlled instruction-following setting in which visible prompts omit requirement sets known to the evaluator. Across three LLMs and 2,700 task–model–condition observations, strict pass rates remained very low under one-shot prompting, templates, and negative-control clarification, while oracle disclosure of the hidden requirements produced much larger gains. These results suggest that robust AI assistants require explicit requirement-state elicitation, faithful integration of elicited constraints, and verification against the resulting requirement state, not only longer prompts or generic clarification.

Declarations

Funding

The authors received no specific funding for this work.

Clinical trial registration

Not applicable. This study is not a clinical trial.

Consent to Publish declaration:

not applicable.

Consent to Participate declaration:

not applicable.

Ethics declaration:

not applicable.

Use of generative AI

Generative AI tools were used for editorial assistance, code drafting, figure drafting, and manuscript preparation. The authors reviewed, edited, and verified the scientific content, code, data files, citations, analyses, and figures, and take responsibility for the final manuscript.

A Formal observations

A.1 Proof of the adversarial safety criterion

Let $\mathcal{H}$ be a nonempty set of feasible histories, let $[\![s]\!]_B \subseteq \mathcal{H}$ be the denotation of instruction s under background theory B, and let $A_{\tau,\eta} = \{h : u(h) \geq \tau, d(h, h_0) \leq \eta\}$. Unless otherwise stated, instructions are assumed to be satisfiable, so $[\![s]\!]_B$ is nonempty when infimum and supremum quantities are used. An admissible execution is any $h \in [\![s]\!]_B$. Thus every admissible execution is acceptable exactly when every element of $[\![s]\!]_B$ belongs to $A_{\tau,\eta}$, which is the set inclusion $[\![s]\!]_B \subseteq A_{\tau,\eta}$. The equivalent inequalities $V_B^-(s) \geq \tau$ and $S_B(s) \leq \eta$ follow from the definitions of infimum utility and supremum side-effect radius.

A.2 Counting lower bound

If an alphabet has size $q \geq 2$, then the number of strings of length at most ℓ is $\sum_{i=0}^{\ell} q^i < q^{\ell+1}$. Therefore representing N pairwise distinct target sets exactly requires length at least $\lceil \log_q N \rceil - 1$ in the worst case. A yes/no clarification protocol is a binary decision tree; distinguishing N possible targets requires at least N leaves and therefore depth at least $\lceil \log_2 N \rceil$.

A.3 Cardinality observation

If $\mathcal{L}$ is countable and $\mathcal{H}$ is infinite, then the image $\{[\![s]\!]_B : s \in \mathcal{L}\}$ is countable, while $\mathcal{P}(\mathcal{H})$ is uncountable by Cantor's theorem. Hence most subsets of $\mathcal{H}$ are not exact denotations of finite instructions.

A.4 Shortest safe synthesis

Let D be a finite desiderata set and let C be a library of clauses, where each clause $c \in C$ guarantees $G(c) \subseteq D$. The decision problem asking whether there are at most k clauses whose union covers D is Set Cover. Membership in NP is immediate, and NP-hardness follows by mapping any Set Cover instance $(U, \mathcal{S}, k)$ to $D = U$ and one clause per set $S \in \mathcal{S}$ with $G(c_S) = S$.

B Additional benchmark details and reproducibility

B.1 Task domains and hidden requirements

The benchmark uses 20 topical domains: product update, policy memo, research summary, decision analysis, incident response, student feedback, travel planning, meeting notes, email reply, comparison, debugging, hiring, financial note, health administration, data interpretation, education plan, legal plain language, operations, security awareness and market analysis. Each domain contributes five task variants, giving 100 total tasks. Hidden-requirement counts are balanced: 25 tasks contain 2 requirements, 25 contain 3, 25 contain 4 and 25 contain 5.

B.2 Clarification menu and protocols

Clarification questions were restricted to a fixed menu to make elicitation behavior auditable. The menu contained: FORMAT_JSON, MAX_WORDS, BULLET_COUNT, ASSUMPTIONS, UNCERTAINTY, RISK_LEVEL, AVOID_ABSOLUTES, NEXT_STEPS, TRADEOFFS and SOURCE_LIMITS. Negative-control clarification used the same question phase as forced clarification, but the simulator answered that no special hidden constraint applied. This should be interpreted as a negative-control condition, not a neutral no-information interaction.

B.3 Rule-based grader details

The grader checks each hidden requirement independently. JSON requirements require a valid JSON object with specified keys. Word-limit requirements count word-boundary tokens. Bullet-count requirements count lines beginning with bullet or numbered-list markers. Assumption requirements require explicit assumption language. Uncertainty requirements require caveat or uncertainty language. Risk requirements require an explicit low, medium or high risk label. Avoid-absolutes requirements fail if banned absolute terms appear. Next-step requirements require a next-step section or key. Trade-off requirements require explicit trade-off signals. Source-limit requirements require both a statement that the response is based on provided information and a statement that it is not externally verified.

B.4 Manual audit

A stratified manual audit of 120 outputs compared human strict-pass labels with the automatic grader. The manual labels agreed with the automatic labels for all audited examples. Because the benchmark uses mechanically checkable requirements, this audit validates implementation fidelity for the grader rules rather than broad semantic usefulness.

References

[1] Brown, T. B. et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems* **33**, 1877–1901 (2020).

[2] OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[3] Ouyang, L. et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems* **35**, 27730–27744 (2022).

[4] Amodei, D. et al. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565* (2016).

[5] Goodhart, C. A. E. Problems of monetary management: the UK experience. In *Monetary Theory and Practice*, 91–121 (Macmillan, 1984).

[6] Pan, A., Bhatia, K. & Steinhardt, J. The effects of reward misspecification: mapping and mitigating misaligned models. *International Conference on Learning Representations* (2022).

[7] Langosco Di Langosco, L., Koch, J., Sharkey, L. D., Pfau, J. & Krueger, D. Goal misgeneralization in deep reinforcement learning. In *Proceedings of the 39th International Conference on Machine Learning*, PMLR **162**, 12004–12019 (2022).

[8] Russell, S. *Human Compatible: Artificial Intelligence and the Problem of Control*. Viking (2019).

[9] Christiano, P. F. et al. Deep reinforcement learning from human preferences. *Advances in Neural Information Processing Systems* **30** (2017).

[10] Bai, Y. et al. Constitutional AI: harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073* (2022).

[11] Hadfield-Menell, D., Russell, S., Abbeel, P. & Dragan, A. Cooperative inverse reinforcement learning. *Advances in Neural Information Processing Systems* **29** (2016).

[12] Hadfield-Menell, D., Milli, S., Abbeel, P., Russell, S. & Dragan, A. Inverse reward design. *Advances in Neural Information Processing Systems* **30** (2017).

[13] Hadfield-Menell, D., Dragan, A., Abbeel, P. & Russell, S. The off-switch game. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, 220–227 (2017).

[14] Nguyen, T. A., Do, M. B., Kambhampati, S. & Srivastava, B. Planning with partial preference models. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 1772–1777 (2009).

[15] Chen, L. & Pu, P. Survey of preference elicitation methods. Technical Report IC/2004/067, Swiss Federal Institute of Technology / EPFL (2004).

[16] Zave, P. & Jackson, M. Four dark corners of requirements engineering. *ACM Transactions on Software Engineering and Methodology* **6**, 1–30 (1997).

[17] Gordon, C. S. & Matskevich, S. Trustworthy formal natural language specifications. In *Proceedings of the 2023 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! '23)*, 50–70 (2023).

[18] Toro Icarte, R., Klassen, T. Q., Valenzano, R. & McIlraith, S. A. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, PMLR **80**, 2107–2116 (2018).

[19] Pezzelle, S. Dealing with semantic underspecification in multimodal NLP. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics* (2023).

[20] Yang, C., Shi, Y., Ma, Q., Liu, M. X., Kästner, C. & Wu, T. What prompts don't say: understanding and managing underspecification in LLM prompts. *arXiv preprint arXiv:2505.13360* (2025).

[21] Ribeiro, M. T., Wu, T., Guestrin, C. & Singh, S. Beyond accuracy: behavioral testing of NLP models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 4902–4912 (2020).

[22] Kiela, D. et al. Dynabench: rethinking benchmarking in NLP. In *Proceedings of NAACL-HLT*, 4110–4124 (2021).

[23] Qin, Y. et al. InFoBench: evaluating instruction following ability in large language models. *arXiv preprint arXiv:2401.03601* (2024).

[24] Gan, Y., Li, C., Xie, J., Wen, L., Purver, M. & Poesio, M. ClarQ-LLM: a benchmark for models clarifying and requesting information in task-oriented dialog. *arXiv preprint arXiv:2409.06097* (2024).

[25] Zhang, M. J. Q. & Choi, E. Clarify when necessary: resolving ambiguity through interaction with LMs. In *Findings of the Association for Computational Linguistics: NAACL 2025*, 5541–5558 (Association for Computational Linguistics, 2025).

[26] Maskin, E. & Tirole, J. Unforeseen contingencies and incomplete contracts. *Review of Economic Studies* **66**, 83–114 (1999).

[27] Al-Najjar, N. I., Anderlini, L. & Felli, L. Unforeseen contingencies. Working paper (2002).

[28] Shannon, C. E. A mathematical theory of communication. *Bell System Technical Journal* **27**, 379–423 and 623–656 (1948).

[29] Kolmogorov, A. N. Three approaches to the quantitative definition of information. *Problems of Information Transmission* **1**, 1–7 (1965).

[30] Karp, R. M. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, 85–103 (Plenum, 1972).

[31] Sculley, D. et al. Hidden technical debt in machine learning systems. In *Advances in Neural Information Processing Systems* **28**, 2503–2511 (2015).

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- Task2.zip
- Task1.zip