DEFAULT-PRESERVING CLARIFICATION: REPAIR, REGRESSION, AND INTERACTION COST

Anonymous authors
Paper under double-blind review

ABSTRACT

With the increasing deployment of language-model agents in complex task environments, proactive clarification has become a standard paradigm to resolve instruction ambiguities. However, an informative clarification can still make the resulting solution worse. When a language-model agent incorporates a reply by regenerating its output, it may repair a failed solution or mistakenly replace a correct one with an incorrect revision. We study this trade-off relative to an explicit default draft. A transition-based account expresses the value of asking as expected repair minus regression and interaction cost. It motivates Default-Preserving Clarification (DPC), a training-free method combining context- and cost-aware selection with a retain-or-revise instruction. In three paired runs on 100 ClarifyCodeBench variants, preservation reaches 49.0% pass@1 versus 47.3% for draft-aware rewriting, while reducing estimated incremental revision API cost by 26.9%. It repairs seven failed defaults and causes no observed regression among the 61 matched-answer pairs whose defaults already pass. With delivered questions and execution pathways held fixed, the cascade and a one-turn SAGE adaptation each solve 83 of 160 tasks, asking 29 and 34 questions, respectively. Ranking and cost studies on coding and translation panels characterize the selection-computation trade-off, with mixed transfer results. These findings support treating answer incorporation as part of the clarification policy, with benefits that depend on both the default and the executor.

1 INTRODUCTION

Language-model agents increasingly operate on underspecified requests, where one instruction admits several plausible interpretations. Clarification has become a standard remedy: rather than guessing, the agent asks a focused question and proceeds with the answer in hand. This practice rests on a rarely stated assumption, namely that a well-chosen question and a correct answer together produce a better solution. We show that the assumption can fail.

A recorded ClarifyCodeBench task makes the failure concrete (Fang et al., 2026). The request allows binary grid flips, and the default program already implements the correct minimization algorithm. Clarification then causes the executor to replace that algorithm with an invalid per-column shortcut that fails the tests; under a retain-or-revise instruction the program keeps the algorithm and passes. The reply is identical; only its incorporation mechanism differs (Appendix H).

Prior work improves clarification timing and question informativeness (Zhang & Choi, 2025; Suri et al., 2026; Kobalczyk et al., 2025; Wei et al., 2026). We examine what follows: how the executor incorporates a reply into a solution that may already be correct. We call a correct-to-incorrect transition *clarification-induced regression*. Aggregate success rates obscure this transition, because they merge repaired and broken outputs into a single number and so cannot separate a policy that fixes many failures from one that also breaks earlier successes.

Nor is preservation automatically the safer choice. Revision is not generally reliable: intrinsic self-correction can degrade reasoning, and code repair depends on feedback quality and computational budget (Huang et al., 2024; Olausson et al., 2024). Our setting supplies external information about intent, but even a truthful reply need not improve a fixed generative executor. Classical nonnegative information value assumes an optimal decision rule that can ignore evidence; it does not guarantee that regeneration implements such a rule. An operator that never revises avoids regressions but

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.

forfeits repairs, and one that always revises gains repairs at the cost of regressions, so the trade-off is genuinely two-sided.

We make this trade-off explicit and then act on it. Expressing the value of asking as expected repair minus regression and interaction cost yields a transition identity that separates the two error directions. It motivates Default-Preserving Clarification (DPC): a context- and cost-aware selector followed by a retain-or-revise executor that keeps the draft when it satisfies the reply and otherwise revises only the conflicting behavior (Figure 1).

Our contributions are an execution-level repair–regression account of clarification value for a fixed executor, including when preservation dominates global rewriting (Section 3); DPC, a training-free selector and retain-or-revise executor that needs no correctness oracle or rollback (Section 4); and paired evaluations that separate answer-incorporation gains from selection gains, with ranking and decision-cost studies across domains (Section 5).

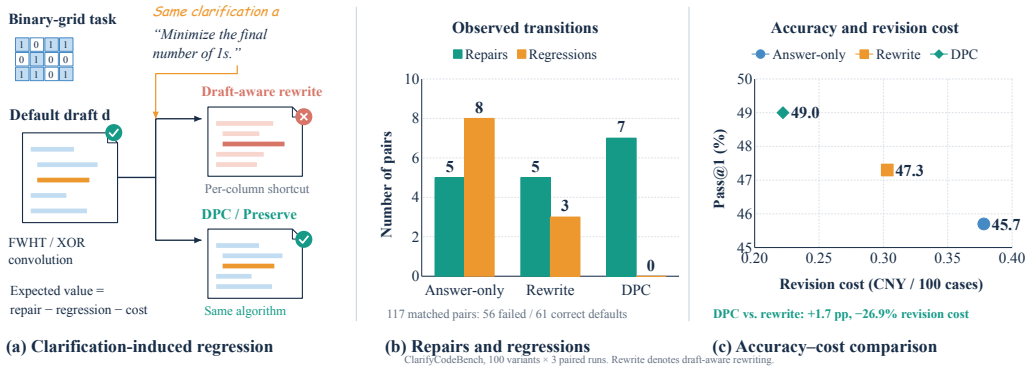

Figure 1: A clarification can repair or break a default solution. Left: schematic of the recorded grid-task failure. Center: transitions among 117 matched-answer pairs (56 failed and 61 successful defaults). Right: pass@1 over all 100×3 pairs and incremental revision API cost. Here DPC denotes its Preserve executor, with defaults and replies held fixed, not the full selection policy. The cost reduction is relative to draft-aware rewriting.

2 Related Work

When and what to ask. IntentSim separates ambiguity from model limitations when deciding whether interaction is worthwhile (Zhang & Choi, 2025), and SAGE combines structured uncertainty, information value, question cost, and a stopping rule (Suri et al., 2026); other work learns interaction policies or studies timing and question choice jointly (Chen et al., 2025a; Wang et al., 2025; Zhao et al., 2026; Deng et al., 2026; Gulati et al., 2026). Question selection has been driven by expected answer utility (Rao & Daumé III, 2018), answer-based utility discriminators (Rao & Daumé III, 2019), dialogue-history conditioning (Aliannejadi et al., 2019), weak supervision (Zamani et al., 2020), and enumeration of plausible answers and rewrites (Min et al., 2020). For task-oriented agents, Active Task Disambiguation applies Bayesian experimental design (BED) over sampled solutions (Kobalczyk et al., 2025), TAI (topological active inference) organizes solutions into persistent intent clusters (Wei et al., 2026), and BED-LLM gathers information over probabilistic hypotheses (Choudhury et al., 2026). We do not claim context or cost as new criteria; our focus is the repair–regression balance of the executor through which a reply acquires value, with the delivered question fixed to isolate selection.

Confidence and incorporation. Self-evaluation and verbalized confidence can predict answer correctness (Kadavath et al., 2022; Tian et al., 2023); semantic uncertainty groups generations by meaning (Kuhn et al., 2023), and semantic entropy is also used to detect confabulation (Farquhar et al., 2024). Such signals do not determine whether a user can supply useful information or whether incorporating it will help, so our selection factors are uncalibrated judgments of those conditions. For incorporation, Self-Refine iterates on model-generated feedback (Madaan et al., 2023), Reflexion

carries verbal feedback across trials (Shinn et al., 2023), SELF-REDRAFT compares refinement with restarting (Chen et al., 2025b), and In-Place Feedback lets users edit a prior output (Choi et al., 2026). DPC instead incorporates a natural-language constraint on task intent, coupling draft-preserving revision to the decision to ask; the contribution is this clarification setting and its paired repair–regression evaluation, not conservative editing in isolation.

3 Clarification Value for a Fixed Executor

Let x include the visible request and context, z denote the intended specification, and d be the default draft produced without clarification. Task success is $u(d, z) \in \{0, 1\}$. A question q elicits a reply a, incorporated by a possibly stochastic executor $R(x, d, q, a)$; below we write $R(d, a)$ with x and q implicit. Our reference action is to execute d. We ask whether the change in task success compensates for the interaction cost $C(q)$.

From information value to execution value. Classical expected value of sample information compares the best posterior action with the best prior action (Poh & Horvitz, 1996). It is nonnegative before observation cost because the prior action remains available. Replacing that posterior optimization with a fixed generation rule removes the guarantee. A truthful reply can redirect the model toward a worse program, even when the original draft already satisfies the intended specification. This does not contradict classical information value; it describes an imperfect implementation of the decision following information acquisition.

Repair and regression. Let H be the event that a usable reply reaches the executor and $\pi_q = \Pr(H \mid x, d, q)$. Outside H, the output remains d. Conditional on H, define

$$\begin{aligned} \alpha_H &= \Pr\big(u(d, z) = 1 \mid H, x, d, q\big), \\ r_R &= \Pr\big(u(R(d, a), z) = 1 \mid u(d, z) = 0, H, x, d, q\big), \\ \ell_R &= \Pr\big(u(R(d, a), z) = 0 \mid u(d, z) = 1, H, x, d, q\big). \end{aligned} \tag{1}$$

The terms r_R and ℓ_R are the repair and regression rates among initially failed and successful defaults, respectively.

Proposition 1 (Repair–regression identity)**.** *If the system retains d whenever no usable reply arrives, its expected incremental utility is*

$$\Delta U(q, R) = \pi_q\big[(1 - \alpha_H) r_R - \alpha_H \ell_R\big] - C(q). \tag{2}$$

This identity partitions the four default-to-revision transitions: repair contributes $+1$, regression contributes -1, and unchanged correctness contributes zero. Appendix A gives the derivation. Writing $\rho = \Pr(\text{response} \mid x, d, q)$ and $p_{\text{ans}} = \Pr(H \mid \text{response}, x, d, q)$ yields $\pi_q = \rho p_{\text{ans}}$. Conditioning α_H on H allows answer availability and default correctness to be correlated.

When does preservation help? Compare a preservation operator S with a global rewrite G, holding the default, reply distribution, and interaction cost fixed. Then

$$\Delta U(q, S) - \Delta U(q, G) = \pi_q\big[\alpha_H(\ell_G - \ell_S) - (1 - \alpha_H)(r_G - r_S)\big]. \tag{3}$$

For $\pi_q > 0$, preservation improves value precisely when avoided regressions outweigh any lost repair opportunities. The condition gives a practical reason to retain a plausible draft, but also a reason to permit revision: a fundamentally wrong algorithm may require extensive changes. Restricting edit scope alone does not make ℓ_S zero, since a local change can alter global behavior.

4 Default-Preserving Clarification

Equation 2 identifies two ways to improve clarification value: avoid interactions with little repair opportunity, and incorporate replies without introducing unnecessary regressions. DPC implements these as separate operations using frozen prompts. A context- and cost-aware selector decides whether to ask; a retain-or-revise instruction determines how the reply changes the default. The transition probabilities describe the resulting behavior; they are not estimated at inference time. Selection precedes code generation, so default adequacy concerns a proposed interpretation, not the correctness of an executable draft.

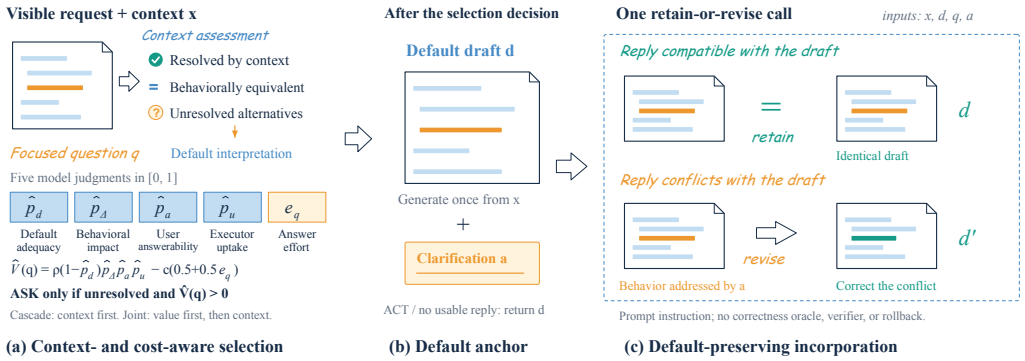

Figure 2: DPC separates selection, default generation, and answer incorporation. The five abbreviated scores correspond in order to Equation 4. Joint scores before context checking; cascade checks context first. Full-panel ranking adds an explicit-conflict check to joint (not drawn). All executable policies share the delivered question and Preserve operator; retain-or-revise is one generation call, not a verifier.

4.1 Deciding Whether to Ask

Context sufficiency. The controller distinguishes three cases. Resolved means the visible input settles the candidate question. Equivalent means the remaining interpretations would produce the same observable behavior. Unresolved means behavior-changing alternatives remain. The first two cases lead to ACT, executing the default without asking. The judgment concerns the need for clarification, not whether the model can implement the task correctly. Equivalence is assessed by the model rather than established by a program proof.

Estimated interaction value. For each unresolved candidate, the model proposes a focused question and a default interpretation, then returns five scores in $[0, 1]$: default adequacy, behavioral impact, user answerability, executor uptake, and answer effort e_q. The selection score is

$$\widehat{V}(q) = \rho(1 - \hat{p}_{\text{default}})\hat{p}_{\text{change}}\hat{p}_{\text{answerable}}\hat{p}_{\text{uptake}} - c(0.5 + 0.5e_q). \tag{4}$$

The product suppresses questions when any required condition is judged weak: the default interpretation is already adequate, alternatives do not change behavior, an answer is difficult to obtain, or the executor is unlikely to use it. The effort term penalizes costly interactions. Native selection retains $\rho = 0.5$, $c = 0.02$, and the affine effort penalty from preliminary runs as operating assumptions, not fitted estimates of user response or effort. It asks only on unresolved candidates with positive score. These factors are uncalibrated model judgments; their product is a selection heuristic, not an estimator of Equation 2 or a probabilistic independence assumption. Regression risk is addressed by the answer-incorporation instruction and measured in the paired execution study.

Two controller implementations. The *joint* controller proposes and scores candidate issues in one call, then invokes a separate context critic to override unnecessary questions. The *cascade* reverses this order: it assesses context first and requests a question and value scores only if an unresolved issue remains. It therefore saves calls on resolved tasks, at the cost of making an early context judgment without a proposed question. We denote them DPC (joint) and DPC (cascade). They share the scoring rule but differ in prompts and call order. Native operation uses the context gate and zero threshold; fixed-quota analyses rank tasks and fill the stated ASK budget. Appendix A explains why these operating rules need not select the same tasks.

The full ranking panels use joint with an additional *conflict check*. A separate call identifies pairs of explicit, incompatible requirements in the visible input. A detected conflict reopens the issue as unresolved and deterministically recomputes its score from existing policy fields, without another Value-model call. Missing information alone does not trigger this override, nor does a conflict force ASK if its score remains nonpositive. The matched 300-case study reports joint with and without this stage, pairing each configuration's quality with its own decision cost.

4.2 Using the Answer Without Discarding the Default

After the selection decision, the backend generates a default draft d from the original request. Under ACT, this is the final output. Under ASK, the draft remains available if no usable answer is obtained. With a usable answer, one revision call receives the original request, the draft, the question, and the reply. The model is instructed to check whether the reply conflicts with the draft, return the draft unchanged if it does not, and otherwise make the smallest necessary correction while preserving unrelated requirements. We label this operator *Preserve* in the experiments.

The operator emits a complete program. It does not use hidden tests, verify semantic equivalence, or mechanically restrict the edit span. Returning the same program is an available action, not a guaranteed outcome. In the grid example from the introduction, the preserved output keeps the already-correct algorithm because the answer adds no conflicting requirement. Appendix H gives the recorded trajectory, and Appendix M provides the actual revision prompts.

Question generation is an interchangeable interface. To isolate task selection, the executable policy study uses the same one-shot question across selectors, even when their internal reasoning proposes different questions. The reported policy comparison consequently measures which requests receive that shared intervention; it does not evaluate each selector's own question generator or optimize a multi-turn dialogue.

5 Experiments

The experiments address two questions: does default preservation improve answer incorporation, and can selective clarification retain those gains with fewer interruptions? We then examine ranking quality and computation across domains. Table 1 lists the panels and their evaluation units.

The comparisons vary one component at a time. The revision study shares defaults and replies across operators, distinguishing draft-free generation, draft-aware rewriting, and preservation. The policy study instead shares the default and revised outcomes, changing only which tasks receive clarification. This design distinguishes a better use of the same information from a better choice of when to request it.

Table 1: Evaluation panels. Variant counts include related examples from the same source. Revision and policy panels overlap in task IDs but use separate generated outputs. Necessity labels describe ambiguity conditions, not measured execution gain.

Study	Data	Cases	Main outcomes
Revision	ClarifyCodeBench, 100 variants	100×3	pass@1; repair; regression
Selection	ClarifyCodeBench, expanded panel	160	solved tasks; ASK; utility
Ranking	ClarifyCodeBench	499	necessity AUROC
	DiscourseMT	267	necessity AUROC
	HumanEvalComm	926	necessity AUROC
Decision cost	100 cases from each ranking panel	300	AUROC; tokens; API cost
Transfer	ClariQ, two disjoint topic sets	111 / 80	necessity AUROC; ASK

5.1 Setup

Data and splits. ClarifyCodeBench provides annotated ambiguities in programming requirements (Fang et al., 2026). The repeated revision study uses 100 variants from 54 programming sources. Its task IDs and ambiguity pairs are disjoint from the development panel, but 44 source problems overlap. The policy panel combines these 100 variants with a 60-variant extension and uses separately generated default and revised outputs; 29 of its variants, from 16 sources, are source-unseen relative to development. The two studies are therefore complementary evaluations, not independent replications. The necessity panels contain ambiguous and complete requirements from ClarifyCodeBench and HumanEvalComm, and context-withheld and context-supplied cases from DiscourseMT (Wu & Fard, 2025; Bawden et al., 2018). ClariQ provides clarification-need labels for search topics (Aliannejadi et al., 2020). Appendix C documents source versions, label balance, and overlap. We abbreviate ClarifyCodeBench as CCB in compact tables and the appendix.

Answers and shared execution. One-shot questions are matched to annotated ClarifyCodeBench ambiguity units by three judge samples and majority vote. A match returns the corresponding gold answer; a miss returns an explicit irrelevant-question response. In every revision arm, that response copies the same default without another model call. In the repeated study, 117 of 300 task–replicate pairs invoke answer incorporation; the other 183 follow the shared copy rule. No real users participate. Controllers and question generators receive only visible task inputs; their outputs are fixed before answer matching and execution evaluation.

Baselines. *Direct* executes the default without asking. *Answer-only* generates from the request and answer without the draft. *Draft-aware rewrite* receives the draft but may freely revise the whole algorithm. For task selection, we compare with Always Ask and a one-turn SAGE adaptation (Suri et al., 2026). The latter enumerates behavior interpretations, partitions them by candidate answers, and applies its structured value and stopping rule. On the full ranking panels we compare with IntentSim and Sample Entropy adaptations (Zhang & Choi, 2025): ten simulated user answers or task outputs are clustered, and the cluster entropy supplies the score. These are local adaptations with a common backend, not results copied from the original papers. In particular, our sampling baselines use model-generated batches and model clustering instead of IntentSim's original sampling and NLI setup. Appendix B.1 specifies the interfaces and shared-question controls.

Metrics and inference. The primary backend is the provider alias `deepseek-flash-guan`. Controllers use temperature 0; the repeated revision study uses 0.2, with a fresh default and three paired answer-incorporation arms in each replicate. Hidden tests evaluate the first program from each arm, so the reported metric is pass@1 rather than best-of-three. Revision differences are averaged within task and bootstrapped over 54 source-problem clusters with 10,000 draws. These intervals are descriptive: the third replicate was added after inspecting the first two. AUROC measures ranking against the necessity proxy; it does not estimate the probability that asking improves execution. Configuration, prompt, and cost details are given in Appendix G.

For a fixed policy with ASK indicator B_i, default outcome u_i^0, and answer-pathway outcome u_i^R, we report

$$\widehat{U}(\rho, c) = \frac{1}{N} \sum_i \left[u_i^0 + \rho B_i (u_i^R - u_i^0) - c B_i \right]. \tag{5}$$

Here ρ is a sensitivity parameter for user response and c is a normalized cost per question. The mixture retains the default on non-response. It reuses fixed outcomes and fixed ASK sets; it is not a user-response estimate. Monetary API cost is reported separately because no empirical conversion between currency and task utility is available.

5.2 Preservation Changes the Repair–Regression Balance

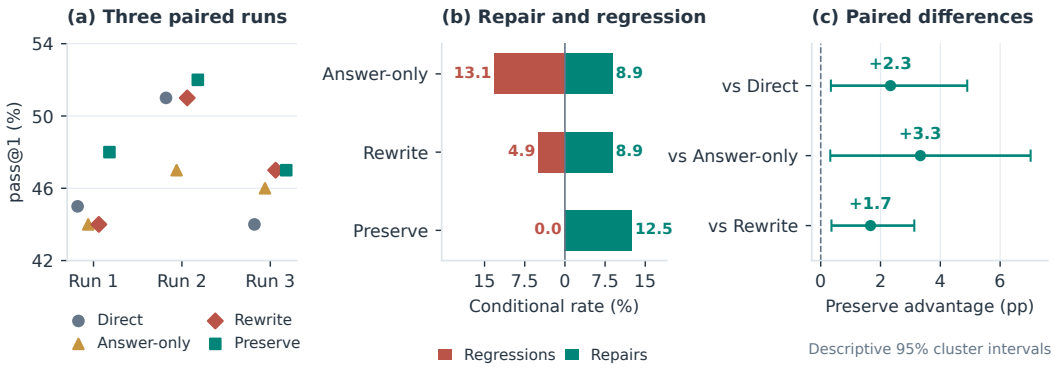

Figure 3: Paired answer-incorporation results on ClarifyCodeBench. (a) Each point is one 100-variant generation replicate. (b) Conditional repair and regression rates, with 56 failed and 61 successful defaults eligible, respectively. (c) Preserve's pass@1 differences with descriptive 95% source-clustered intervals. "Rewrite" denotes draft-aware rewriting throughout the figures.

Table 2: Revision outcomes in three paired runs on 100 ClarifyCodeBench variants. Among 117 matched-answer pairs, 56 defaults fail and 61 pass; the other 183 pairs copy the shared default. Cost is estimated incremental revision API cost per 100 evaluated cases, excluding shared drafting, question generation, matching, and testing.

Operator	Passed / 300	pass@1 (%)	Repair / 56	Regression / 61	CNY/100
Direct	140	46.7	–	–	–
Answer-only	137	45.7	5	8	0.378
Draft-aware rewrite	142	47.3	5	3	0.303
Preserve	**147**	**49.0**	**7**	**0**	**0.222**

Preserve improves pass@1 over draft-aware rewriting by 1.7 percentage points, with a descriptive source-clustered interval of [0.4, 3.1] (Table 2). Both operators see the same draft and answer, but differ in revision instructions. The bootstrap interval excludes zero, but the post-hoc source sign-flip test does not reject at the 0.05 level ($p = 0.0625$; Appendix E). Five sources favor Preserve, none favors rewriting, and 49 tie. We treat this as directional evidence concentrated in five sources, not a confirmatory significance claim.

The transition counts explain the aggregate difference. Preserve repairs 7/56 failed defaults (12.5%) and regresses on 0/61 successful ones. Draft-aware rewriting repairs 5/56 (8.9%) and regresses on 3/61 (4.9%); answer-only generation repairs 5/56 and regresses on 8/61 (13.1%). Zero observed regressions do not establish zero risk. Preserve returns exactly the same code in 45 matched cases and changes it in 72, so the result is not explained by copying every draft.

In aggregate, the five additional passes over draft-aware rewriting comprise two more repairs and three fewer regressions. Thus the observed advantage is not simply greater conservatism at the expense of repair. Across the three generation replicates, Preserve solves 48, 52, and 47 tasks versus 44, 51, and 47 for rewriting, leading in two runs and tying in the third (Appendix Table 21).

Preserve also uses 26.9% less incremental revision API cost than draft-aware rewriting (0.222 versus 0.303 CNY per 100 cases), with the same 117 revision calls. An earlier ablation shows a larger gap when regeneration has no draft anchor: naive regeneration loses six points relative to Direct, whereas Preserve gains three (Appendix I). That comparison changes both draft access and instructions; the repeated draft-aware comparison is the more specific test of preservation. In both studies, answer availability alone does not determine the sign of the execution gain.

5.3 Selective Clarification Retains Part of the Gain

Table 3: Policy comparison on 160 ClarifyCodeBench variants. All ASK policies share the delivered question, matched answer, default, and Preserve outcome; Hits counts matched questions. Pass@1 assumes full response; utility uses $\rho = 1$, $c = 0.02$. The 30% quota is applied within each constituent panel (30+18 questions). Our selector is the cascade.

		ASK			pass@1	
Policy	Asked	(%)	Hits	Solved	(%)	Utility
Direct	0	0.0	–	81	50.63	0.5063
Always Ask	160	100.0	49	85	53.13	0.5113
SAGE-adapted, 30%	48	30.0	21	83	51.88	0.5128
DPC (cascade), 30%	48	30.0	20	83	51.88	0.5128
SAGE-adapted, native	34	21.3	18	83	51.88	0.5145
DPC (cascade), native	29	18.1	18	83	51.88	0.5151

Always Ask solves four additional tasks over Direct but interrupts every request. Both native selectors recover two of those gains (Table 3). DPC asks 29 questions versus 34 for SAGE-adapted, a 14.7% reduction, with the same 18 question matches and 83 solved tasks. At a matched 30% quota, their success counts also tie. The observed native-policy advantage is therefore fewer interruptions at equal success, not an accuracy improvement over SAGE. Its utility difference is 0.000625 and comes entirely from question cost.

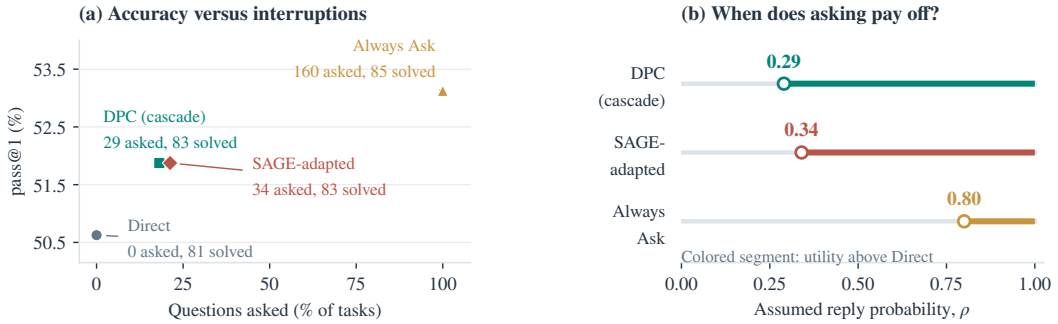

Figure 4: Execution–interaction trade-off on 160 variants. (a) Native operating points: cascade and SAGE-adapted both solve 83 tasks, with 29 and 34 questions. The vertical axis is restricted to show the small accuracy differences. (b) Empirical reply-probability thresholds above which each policy exceeds Direct's utility, with fixed outcomes and $c = 0.02$. Colored intervals denote positive net gain; these are sensitivity calculations, not observed response rates.

Figure 4 shows how the comparison changes when users may not respond. At $c = 0.02$, the empirical break-even probabilities against Direct are 0.80 for Always Ask, 0.34 for SAGE-adapted, and 0.29 for DPC. Below those values, the observed execution gains no longer offset interruptions. These thresholds describe the fixed sampled outcomes rather than a learned model of user availability. Appendix E reports the exact arithmetic and its statistical limits.

5.4 Ranking Quality and the Cost of Selection

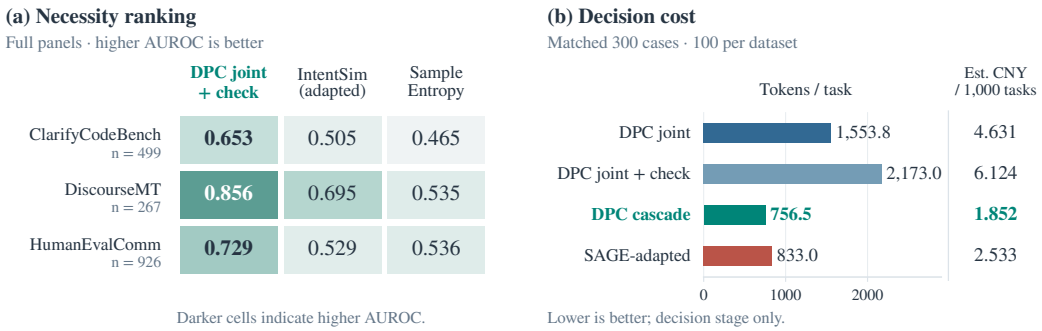

Figure 5: Necessity ranking and decision computation. (a) Full-panel AUROC; the DPC entry uses joint plus conflict checking. (b) Mean decision tokens on a separate matched subset of 100 cases per dataset; the right column rescales estimated fees to CNY per 1,000 tasks. Costs exclude drafting, question delivery, and revision. Table 4 pairs each configuration with its matched-subset AUROCs and per-task fees.

On the complete panels, joint with conflict checking reaches AUROCs of 0.653 on ClarifyCodeBench, 0.856 on DiscourseMT, and 0.729 on HumanEvalComm. IntentSim-adapted scores 0.505, 0.695, and 0.529; Sample Entropy scores 0.465, 0.535, and 0.536. These are necessity-label comparisons, not measured execution gains. Appendix D gives direct-score reconstruction and class-sensitive average precision; Appendix G details conflict-check cost. Full-panel SAGE scores are unavailable; its matched-subset comparison is reported separately.

The cascade invokes Context on every task but skips 213 of 300 Value calls. Relative to joint with conflict checking, it uses 65.2% fewer decision tokens and reduces mean summed API time from 6.17 to 2.39 seconds. AUROC falls by 0.112 on ClarifyCodeBench and 0.016 on DiscourseMT, but rises by 0.020 on HumanEvalComm. Conflict checking itself improves the two-call joint controller's HumanEvalComm AUROC from 0.687 to 0.781, with no consistent benefit on the other two subsets.

Table 4: Quality and decision cost on the matched 300-case subset. AUROC columns each use 100 cases. Tokens and estimated API cost are per task, averaged over all 300 cases; costs exclude question delivery, drafting, revision, and test execution. Bold marks the best value in each column.

Controller	CCB	DiscourseMT	HumanEvalComm	Tokens	CNY/task
DPC (joint)	**0.694**	0.820	0.687	1553.8	0.004631
Joint + conflict check	**0.694**	0.813	0.781	2173.0	0.006124
DPC (cascade)	0.582	0.797	**0.801**	**756.5**	**0.001852**
SAGE-adapted	0.579	**0.917**	0.780	833.0	0.002533

SAGE-adapted remains substantially stronger on DiscourseMT. Different prompts and call orders make this a controller comparison, not an isolated early-exit ablation.

5.5 Transfer and Boundary Conditions

The gains do not hold uniformly. On DiscourseMT contrastive choice, the joint controller with conflict checking has higher cost-adjusted utility than Always Ask (0.772 versus 0.766, at $\rho = 0.5$ and $c = 0.02$) but lower full-response accuracy (83.6% versus 84.3%). This is an A/B choice evaluation, not translation sequence likelihood. On 100 HumanEvalComm tasks supplied with complete hidden requirements, Preserve and draft-aware rewriting both solve 93, while answer-only generation solves 94. Preservation retains all 51 initially successful outputs but gives no accuracy advantage over the other answer-conditioned operators.

The cascade also trails SAGE-adapted on 111 ClariQ dev/test topics (0.586 versus 0.714 AUROC), then leads on a disjoint 80-topic sample from the training split (0.728 versus 0.618). Both are zero-shot comparisons without threshold tuning. These changes argue against a domain-independent ranking advantage. Appendix F includes operating points and complete-information reference conditions; Appendix K records inconclusive cross-executor probes.

6 Discussion and Limitations

The paired experiments distinguish two effects of clarification: acquiring a useful constraint and incorporating it without disrupting a successful draft. Their distinction matters when evaluating a selective policy. Fewer questions can improve utility even with unchanged success, while a better revision operator can improve success without changing which questions are asked. Neither effect guarantees stronger necessity ranking across domains.

The main execution evidence uses simulated, matcher-gated replies and one provider-routed API alias. It does not establish generalization to natural conversations or multiple fixed model checkpoints. Shared sources across development and evaluation, and the adaptively added third replicate, constrain the interpretation of the reported intervals. Source-clustered resampling addresses related variants but cannot replace a source-disjoint replication. The selector uses uncalibrated scores, and preservation is a prompt instruction without a verifier. Finally, the policy study evaluates a shared delivered question and fixed execution outcomes; it does not measure each method's full native dialogue. Larger source-disjoint studies and matched cross-executor controls are needed to determine the reliability of the observed gains.

7 Conclusion

The value of clarification depends on how the answer changes execution. Relative to a shared default, a reply can repair a failed output or induce a regression. DPC makes retaining the default explicit and combines this operation with context- and cost-aware selection. Paired code experiments show a better observed repair–regression balance than draft-aware rewriting. In the policy panel, the cascade and SAGE-adapted each solve 83 tasks, with five fewer questions for the cascade. Uneven transfer results qualify this finding: preservation and selectivity should be evaluated together, reporting execution gains, interruptions, and computation separately.

AI USE STATEMENT

The authors employed AI tools to support various stages of this research, specifically for routine code development, bug identification, and manuscript formatting assistance. We emphasize that all core experimental setups, algorithmic logic, and final outcomes were independently verified by the authors through rigorous manual testing. The authors maintain complete ownership of the research findings and take full responsibility for the integrity of the published results.

ETHICS STATEMENT

The study uses public benchmarks and constructed development cases. No new human participants were recruited; replies were simulated or obtained from benchmark annotations. Such answers and proxy necessity labels do not capture users' full preferences, effort, or willingness to respond. Benchmark contamination and provider-side model changes may affect the results. In deployment, unnecessary questions can burden users, while an incorrect retained draft can remain harmful; the observed absence of regressions is not a safety guarantee. Any redistribution of benchmark-derived data must respect upstream licenses and restrictions.

REPRODUCIBILITY STATEMENT

Appendix A states the theoretical assumptions and derivations. Appendices C, E, and G specify data provenance, overlap, metrics, inference settings, and statistical analyses; Appendix M gives the core prompts. The companion package contains editable figures and offline scripts auditing revision outcomes, ranking scores, and stage-aligned decision costs against archived records. Per-example logs remain in the research archive. Exact checkpoint replication is limited by the provider's unversioned API alias. The ClariQ snapshot is identified by file hashes and frozen topic IDs. Release of code, derived identifiers, and logs remains subject to upstream permissions.

REFERENCES

Mohammad Aliannejadi, Hamed Zamani, Fabio Crestani, and W. Bruce Croft. Asking clarifying questions in open-domain information-seeking conversations. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2019. doi: 10.1145/3331184.3331265. URL https://arxiv.org/abs/1907.06554.

Mohammad Aliannejadi, Julia Kiseleva, Aleksandr Chuklin, Jeff Dalton, and Mikhail Burtsev. ConvAI3: Generating clarifying questions for open-domain dialogue systems (ClariQ), 2020. URL https://arxiv.org/abs/2009.11352. arXiv:2009.11352.

Rachel Bawden, Rico Sennrich, Alexandra Birch, and Barry Haddow. Evaluating discourse phenomena in neural machine translation. In *Proceedings of NAACL-HLT 2018*, pp. 1304–1313, 2018. URL https://aclanthology.org/N18-1118/.

Maximillian Chen, Ruoxi Sun, Tomas Pfister, and Sercan Arik. Learning to clarify: Multi-turn conversations with action-based contrastive self-training. In *International Conference on Learning Representations*, 2025a. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/4ffd05ca3cf3985f4572af015b4cfc1e-Abstract-Conference.html.

Yixiang Chen, Tianshi Zheng, Shijue Huang, Zhitao He, and Yi R. Fung. SELF-REDRAFT: Eliciting intrinsic exploration-exploitation balance in test-time scaling for code generation, 2025b. URL https://arxiv.org/abs/2511.02854. arXiv:2511.02854.

Youngbin Choi, Minjong Lee, Saemi Moon, Seunghyuk Cho, Chaehyeon Chung, MoonJeong Park, and Dongwoo Kim. In-place feedback: Reliable refinement for multi-turn expert-LLM collaboration, 2026. URL https://arxiv.org/abs/2510.00777. arXiv:2510.00777, version 2.

Deepro Choudhury, Sinead Williamson, Adam Goliński, Ning Miao, Freddie Bickford Smith, Michael Kirchhof, Yizhe Zhang, and Tom Rainforth. BED-LLM: Intelligent information gathering with

LLMs and bayesian experimental design. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=qyylZMLYT8.

Mengyi Deng, Zhiwei Li, Xin Li, Tingyu Zhu, Ying Zhao, Zhijiang Guo, and Wei Wang. Uncertainty-aware clarification in LLM agents with information gain, 2026. URL https://arxiv.org/abs/2606.03135. arXiv:2606.03135.

Yang Deng, Wenqiang Lei, Wenxuan Zhang, Wai Lam, and Tat-Seng Chua. PACIFIC: Towards proactive conversational question answering over tabular and textual data in finance. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 6970–6984, 2022. doi: 10.18653/v1/2022.emnlp-main.469. URL https://aclanthology.org/2022.emnlp-main.469/.

Zheng Fang, Dongming Jin, Yihong Dong, Yongmin Li, Kechi Zhang, Zhi Jin, and Ge Li. ClarifyCodeBench: Evaluating LLMs on clarifying ambiguous requirements for code generation, 2026. URL https://arxiv.org/abs/2607.00711. arXiv:2607.00711.

Sebastian Farquhar, Jannik Kossen, Lorenz Kuhn, and Yarin Gal. Detecting hallucinations in large language models using semantic entropy. *Nature*, 630:625–630, 2024. doi: 10.1038/s41586-024-07421-0. URL https://www.nature.com/articles/s41586-024-07421-0.

Anmol Gulati, Hariom Gupta, Elias Lumer, Sahil Sen, and Vamse Kumar Subbiah. Ask early, ask late, ask right: When does clarification timing matter for long-horizon agents?, 2026. URL https://arxiv.org/abs/2605.07937. arXiv:2605.07937.

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. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.01798.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El-Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, Jackson Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom Brown, Jack Clark, Nicholas Joseph, Ben Mann, Sam McCandlish, Chris Olah, and Jared Kaplan. Language models (mostly) know what they know, 2022. URL https://arxiv.org/abs/2207.05221. arXiv:2207.05221.

Katarzyna Kobalczyk, Nicolás Astorga, Tennison Liu, and Mihaela van der Schaar. Active task disambiguation with LLMs. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=JAMxRSXLFz.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2302.09664.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Sewon Min, Julian Michael, Hannaneh Hajishirzi, and Luke Zettlemoyer. AmbigQA: Answering ambiguous open-domain questions. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 5783–5797, 2020. doi: 10.18653/v1/2020.emnlp-main.466. URL https://aclanthology.org/2020.emnlp-main.466/.

Theo X. Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. Is self-repair a silver bullet for code generation? In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/9ddc141bdbf9d1db510cefff56c586ad-Abstract-Conference.html.

Kim-Leng Poh and Eric J. Horvitz. A graph-theoretic analysis of information value. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, pp. 427–435, 1996. URL https://arxiv.org/abs/1302.3596.

Sudha Rao and Hal Daumé III. Learning to ask good questions: Ranking clarification questions using neural expected value of perfect information. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2737–2746, 2018. doi: 10.18653/v1/P18-1255. URL https://aclanthology.org/P18-1255/.

Sudha Rao and Hal Daumé III. Answer-based adversarial training for generating clarification questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 143–155, 2019. doi: 10.18653/v1/N19-1013. URL https://aclanthology.org/N19-1013/.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Manan Suri, Puneet Mathur, Nedim Lipka, Franck Dernoncourt, Ryan A. Rossi, and Dinesh Manocha. Structured uncertainty guided clarification for LLM agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 40811–40838, 2026. doi: 10.18653/v1/2026.findings-acl.2028. URL https://aclanthology.org/2026.findings-acl.2028/.

Katherine Tian, Eric Mitchell, Allan Zhou, Archit Sharma, Rafael Rafailov, Huaxiu Yao, Chelsea Finn, and Christopher Manning. Just ask for calibration: Strategies for eliciting calibrated confidence scores from language models fine-tuned with human feedback. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5433–5442, 2023. doi: 10.18653/v1/2023.emnlp-main.330. URL https://aclanthology.org/2023.emnlp-main.330/.

Wenxuan Wang, Shi Juluan, Zixuan Ling, Yuk-Kit Chan, Chaozheng Wang, Cheryl Lee, Youliang Yuan, Jen-tse Huang, Wenxiang Jiao, and Michael R. Lyu. Learning to ask: When LLM agents meet unclear instruction. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 21773–21784, 2025. doi: 10.18653/v1/2025.emnlp-main.1104. URL https://aclanthology.org/2025.emnlp-main.1104/.

Yangbo Wei, Zhen Huang, Shaoqiang Lu, Junhong Qian, Chen Wu, and Lei He. Topological active inference for task disambiguation. In *International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=d8jyUp19OE.

Jie JW Wu and Fatemeh H. Fard. HumanEvalComm: Benchmarking the communication competence of code generation for LLMs and LLM agent. *ACM Transactions on Software Engineering and Methodology*, 2025. doi: 10.1145/3715109. URL https://arxiv.org/abs/2406.00215. arXiv:2406.00215.

Hamed Zamani, Susan Dumais, Nick Craswell, Paul Bennett, and Gord Lueck. Generating clarifying questions for information retrieval. In *Proceedings of The Web Conference 2020*, pp. 418–428, 2020. doi: 10.1145/3366423.3380126. URL https://www.microsoft.com/en-us/research/publication/generating-clarifying-questions-for-information-retrieval/.

Linghao Zhang, Shilin He, Chaoyun Zhang, Yu Kang, Bowen Li, Chengxing Xie, Junhao Wang, Maoquan Wang, Yufan Huang, Shengyu Fu, Elsie Nallipogu, Qingwei Lin, Yingnong Dang, Saravan Rajmohan, and Dongmei Zhang. SWE-bench goes live!, 2025. URL https://arxiv.org/abs/2505.23419. arXiv:2505.23419.

Michael JQ Zhang and Eunsol Choi. Clarify when necessary: Resolving ambiguity through interaction with LMs. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 5541–5558, 2025. doi: 10.18653/v1/2025.findings-naacl.306. URL https://aclanthology.org/2025.findings-naacl.306/.

Jiale Zhao, Ke Fang, and Lu Cheng. When and what to ask: AskBench and rubric-guided RLVR for LLM clarification. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17120–17140, 2026. doi: 10.18653/v1/2026.findings-acl.845. URL https://aclanthology.org/2026.findings-acl.845/.

A Derivations and the Boundary of the Preservation Argument

Nonnegative EVSI is an expectation statement. Classical expected value of sample information (Poh & Horvitz, 1996) is

$$\mathrm{EVSI} = \mathbb{E}_a\left[\max_f \mathbb{E}[u(f,z) \mid x,a]\right] - \max_f \mathbb{E}[u(f,z) \mid x] \geq 0. \tag{6}$$

Let $f^\star$ maximize prior expected utility. For every evidence realization a, the posterior optimum has value at least $\mathbb{E}[u(f^\star,z) \mid x,a]$. Taking expectations over a and applying the tower property proves Equation 6. This argument requires that $f^\star$ remain feasible after information arrives. It does not require that an optimal posterior policy preserve correctness in every hidden state, and a fixed generative executor need not implement the posterior maximization.

A binary example makes the distinction explicit. Let states A, B have prior probabilities 0.6 and 0.4, and actions guess the state for reward one. The prior-optimal action is A. Let a signal E occur with probability 0.2 in state A and 0.8 in state B. On E, the optimal action is B because its posterior probability is $0.32/0.44$; outside E, the optimal action remains A. The evidence-conditioned policy succeeds with probability $0.32 + 0.48 = 0.80$, improving on 0.60. Nevertheless, state-A cases with signal E regress, with joint probability 0.12. Positive EVSI therefore coexists with positive regression probability relative to the prior action. Classical information value is not equivalent to setting $\ell_R = 0$.

Derivation of execution value. Outside H, the output equals d and contributes zero to incremental reward. Conditional on H, partition outcomes into the four binary transitions $(u(d,z), u(R(d,a),z))$. Transitions $(0,0)$ and $(1,1)$ contribute zero, $(0,1)$ contributes $+1$, and $(1,0)$ contributes -1. Thus the conditional expectation is $(1-\alpha_H)r_R - \alpha_H \ell_R$. Multiplying by $\Pr(H \mid x,d,q)$ and subtracting cost proves Equation 2. Correlation between answer availability and default correctness is absorbed into α_H.

Expectations are over the intended specification, reply availability and content, and executor randomness, conditional on the visible input and fixed draft. A transition rate conditioned on a zero-probability branch is immaterial: its weighted contribution is zero. If $\pi_q = 0$, no revision occurs and the incremental utility is simply $-C(q)$.

Subtracting this identity for two executors on the same distribution proves Equation 3. The analysis does not certify preservation: a local code edit can change global behavior, and zero observed regressions in a finite matcher-gated sample is not a bound on future risk.

Quotas versus thresholds. For independent decisions with additive known rewards v_i, identical unit ASK costs, and exactly k required questions, maximizing $\sum_i B_i v_i$ selects the top k values. A threshold can reproduce that solution with appropriate tie-breaking. With at most k questions, negative-valued cases need not be selected. Heterogeneous costs, interactions, mandatory gates, estimation error, and native score scaling break a blanket equivalence. Our offline quota fills all $\lceil bN \rceil$ slots, even when native context gating or a zero threshold would ask fewer questions. We therefore report them as different operating rules.

B Evaluation Inventory

Table 5 distinguishes the main execution studies from ranking evaluations, transfer checks, and development probes. Groups identify the evaluation protocol; their sample counts and utilities are not pooled across protocols. Model aliases are listed in Table 18.

B.1 Baseline and Controller Interfaces

SAGE adaptation. The implementation enumerates one to six mutually exclusive behavioral interpretations with normalized weights p_j, and up to three questions partitioning those interpretations into answer groups $A_k(q)$. For each question, the code computes

$$v(q) = \sum_k \max_{j \in A_k(q)} p_j - \max_j p_j, \qquad m(q) = v(q) - 0.1 \max_j p_j. \tag{7}$$

Table 5: Experiment inventory. Groups 1–3 contain the main and supporting evaluations; groups 4–6 document earlier development and diagnostic studies.

Group	Experiment family	Role in this paper
1	CCB paired revision, 100 variants × 3 replicates	Main execution evidence (Section 5.2).
1	CCB policy panel, 160 variants	Main selection evidence (Section 5.3).
2	CCB/DiscourseMT/HumanEvalComm/ClariQ necessity proxies	Proxy ranking and operating points (Section 5.4).
2	Context→Value cascade, 300 cases	Decision-cost comparison (Section 5.4).
3	DiscourseMT contrastive downstream; HumanEvalComm complete-requirement injection; question-match audit	Boundary conditions and protocol audits.
4	Controlled development panels (App. J)	Template-labeled development checks; never pooled with groups 1–3.
5	Cross-executor and placebo-controlled probes (App. K)	Small-sample checks of transfer and answer specificity.
6	Development diagnostics, negative results, and infrastructure-blocked runs (App. L)	Earlier exploratory results and feasibility checks.

Table 6: Selection interfaces. No selector sees generated code. Internal candidate questions support scoring; they need not be the question delivered in the shared-intervention evaluation.

Method	Score or decision	Question interface
Direct / Always Ask	Never / always intervene	Uses the common delivered question if asking
IntentSim-adapted	Entropy of ten simulated user answers	Scores an available candidate question
Sample Entropy	Entropy of ten simulated output contracts	Samples predictions from the visible request
Ask-or-Act	Prompted binary necessity judgment	Receives an available candidate as context
SAGE-adapted	Structured answer-partition value and stopping margin	Proposes up to three internal candidates
DPC (joint)	Context-filtered value score	Proposes and scores candidate issues, then checks context
DPC (cascade)	Value score after context check	Proposes a candidate only if context is unresolved

It asks if context is unresolved and the best margin is nonnegative. The history-dependent redundancy penalty is zero in this one-turn setting. This preserves the structured belief and stopping components of SAGE while replacing tool schemas with behavior interpretations. It is not a reproduction of SAGE's full multi-turn tool agent.

Sampling and binary baselines. IntentSim-adapted generates a list of ten plausible user answers in one call and groups them by intended behavior in a second call. Sample Entropy analogously generates ten task predictions; for code, these are behavioral contracts rather than complete programs. Both use cluster entropy. The original IntentSim uses its own sampling and NLI clustering, so our results concern the stated adaptations. Ask-or-Act is a descriptive name for the local binary intent-agent prompt; it is not presented as a separately reproduced published system.

Shared-question evaluation. In the full proxy comparison, sampling baselines receive a candidate derived from the joint controller, and question-usefulness judgments are reused across selectors. In the executable CCB policy study, selector decisions are recorded first. A separate one-shot generator then produces one shared question per task from the visible requirement alone, without controller outputs or hidden annotations in its payload. The question is fixed before annotation matching supplies a reply, and its answer pathway is reused across policies. Both constituent plans use the original `QUESTION_SYSTEM` prompt reproduced in Appendix M. This isolates task selection under a common intervention, while allowing a mismatch between a controller's internal candidate and the delivered question.

C Data Construction, Splits, and Chronology

Table 7: Constructed public-input panels. Positive means a clarification-necessity proxy, not positive net downstream value. Counts include related variants and must not be interpreted as independent source problems.

Panel	Positive	Negative	Total	Construction
CCB	419	80	499	Underspecified variants and complete original controls
DiscourseMT	89	178	267	Context withheld versus two disambiguating-context versions
HumanEvalComm	762	164	926	Modified HumanEval descriptions and complete originals
ClariQ	47	64	111	Need level ≥ 3 versus lower levels

The CCB, DiscourseMT, and HumanEvalComm source versions reported in the implementation audit are, respectively, `5e2d5b5ce6259daa034cebb69f65e5e4c6dec3e9`, `13d88cee7cfca92d3d5ae2b471b04c62795baac3`, and `794a91f3c520e3939b8361e9f4ae927c25756a6e`. The local ClariQ snapshot is not a version-controlled checkout, so we identify it by the SHA-256 hashes below. ClariQ-80 is a label-stratified sample of the train file; ClariQ-111 uses dev and test topics only. Their run plans preserve every topic ID. The local DiscourseMT parser yields 89 blocks, each expanded into three context conditions. The resulting 267-case panel differs from an earlier 88-block extraction reported during development.

Table 8: ClariQ source and frozen-panel provenance.

Artifact	SHA-256
`train.tsv`	`65d3da13b2d6ea77e7eaa45290894ffc162a5bd000e7640decd1b0a272a6e9d1`
`dev.tsv`	`68d2a5f87eab73721979b5f45f64099a9b2f080db1d0ce4b979d9daa4249906e`
`test_with_labels.tsv`	`792fb25a3f258e96b6957490faae13dc2ab9926a0d23345b12ab4b925eb02b58`
ClariQ-80 public/private	`85badc11cffb82179ee18072b0fc5ac8e1a8bcc44d7574a034492e81a52b0b20/df2e1e52e84f730a969c190ee04583e2eedf7a6e5a2e2e1d62cc36a4d2962aac`
ClariQ-111 public/private	`900340683afe2928ff913fc6ebabebf3f847c2e7982ae40aee277c7fc76eeb8c/446a51ccc75d0a441b95f3a1c9e6ba18a87cb1cb3143179a2ead4283f77cd18f`

The development, held-out task/pair, and extension CCB sets contain 100, 100, and 60 variants, covering 59, 54, and 39 source problems. Development/held-out, development/extension, and held-out/extension share 44, 28, and 25 sources, respectively. A source can have one to seven ambiguity variants. Only 29 evaluation variants from 16 sources are unseen at source level relative to development. Neither the 160-variant aggregate nor the 100-variant repeated study is a strict source-generalization evaluation.

Chronology. The full proxy panels and joint-controller comparisons were assembled around September 16, 2026. Development on the original 100 CCB variants exposed harmful regeneration, after which the preservation pathway was evaluated on a new set of 100 variants and a 60-variant extension. Local selection plans preceded the model calls, but the studies were not externally preregistered. Two revision replicates were run first; the third was added after examining them, without replacing their outputs. The DiscourseMT downstream study, HumanEvalComm revision check, and cascade controller were evaluated subsequently.

Other panels. The project also contains the controlled context/cost and executable-world panels of Appendix J. Their labels arise from templates, rules, and hidden worlds rather than human annotation. They are reported separately and their sample counts are not added to the natural-panel aggregates of groups 1–3.

D Exact Metric Definitions and Fixed-Budget Selection Results

Let Y_i be the binary necessity label, B_i the selected ASK action, and H_i the question-usefulness judgment. Ordinary WHEN Accuracy is

$$\mathrm{Acc}_{\mathrm{when}} = N^{-1} \sum_i \mathbf{1}[B_i = Y_i]. \tag{8}$$

The archival effective-question score uses

$$T_e = \sum_i Y_i B_i H_i, \qquad F_e = \sum_i (1 - Y_i) B_i, \qquad M_e = \sum_i Y_i - T_e, \tag{9}$$

$$F_{1,e} = \frac{2T_e}{2T_e + F_e + M_e}. \tag{10}$$

Here T_e is an effective hit, F_e counts ASK on necessity-negative cases, and M_e includes necessity-positive cases with ineffective questions even when the action ASK itself was correct. Ineffective questions on positive cases are not included in F_e. Consequently, the precision implicit in Equation 10, $T_e/(T_e+F_e)$, is not the fraction of all asked questions that are useful. We retain the archival definition for transparency, without presenting it as conventional binary F_1 or as a complete interaction-efficiency metric. Question-hit/ASK uses all asked questions as its denominator.

The controlled-panel metric of Appendix J is defined differently and must not be compared with it. There, *beneficial-ask precision* is the count of ASK actions whose net value is positive under the panel's cost model divided by the count of ASK actions, and the denominator is the number of cases the method chose to ask on, not the number it was entitled to ask on. The two metrics answer different questions: the archival effective-question F_1 asks whether a useful question was generated on a case that needed one, whereas beneficial-ask precision asks whether the interruption paid for itself.

For CCB at the 10% quota, both methods ask 50 questions. Joint with conflict checking has ordinary TP/FP/TN/FN of 50/0/80/369; IntentSim-adapted has 43/7/73/376. Their ordinary accuracies are 26.1% and 23.2%. The effective counts (T_e, F_e, M_e) are 39/0/380 and 19/7/400, giving effective F_1 values of 17.0% and 8.5%. These values are reconstructed from the frozen scores and shared question judgments.

Table 9: Full-panel proxy selection results. DPC uses joint plus conflict checking. Cells show WHEN Accuracy / effective-question F_1 in percent. Candidate questions and their judgments are shared. These numbers do not describe the later cascade.

Panel	ASK quota	DPC joint + check	IntentSim-adapted	Sample Entropy
CCB	10%	26.1 / 17.0	23.2 / 8.5	23.2 / 8.5
	20%	34.5 / 29.4	29.7 / 14.5	29.3 / 13.3
	30%	42.9 / 40.7	35.7 / 19.1	34.9 / 19.0
DiscourseMT	10%	73.0 / 37.9	72.3 / 36.2	63.3 / 15.5
	20%	77.2 / 57.3	71.9 / 46.5	60.7 / 26.6
	30%	77.5 / 64.7	70.8 / 53.3	61.0 / 37.9
HumanEvalComm	10%	27.8 / 21.5	25.2 / 11.2	27.5 / 16.8
	20%	36.1 / 36.2	31.1 / 20.0	35.4 / 29.0
	30%	44.3 / 45.2	38.9 / 30.1	40.0 / 34.8

The quota is $\lceil bN \rceil$ over all cases: 50/100/150 for CCB, 27/54/81 for DiscourseMT, and 93/186/278 for HumanEvalComm. Joint ranks each task by the maximum unresolved-issue score, using -1 if none remains; cascade uses its single Value score or -1 after early exit. Ties are broken by task identifier. This is neither a quota within unresolved cases nor a simulated arrival-time online budget. A native controller may ACT where offline quota filling still selects ASK. This distinction is important on CCB and HumanEvalComm, whose high positive prevalence also makes accuracy small under restrictive ASK budgets.

Direct-score reconstruction and class balance. AUROC is computed from each baseline's own normalized entropy, aligned by task ID with the same necessity labels as DPC. An earlier evaluator copied DPC issue records into baseline reports and inadvertently retained non-candidate scores

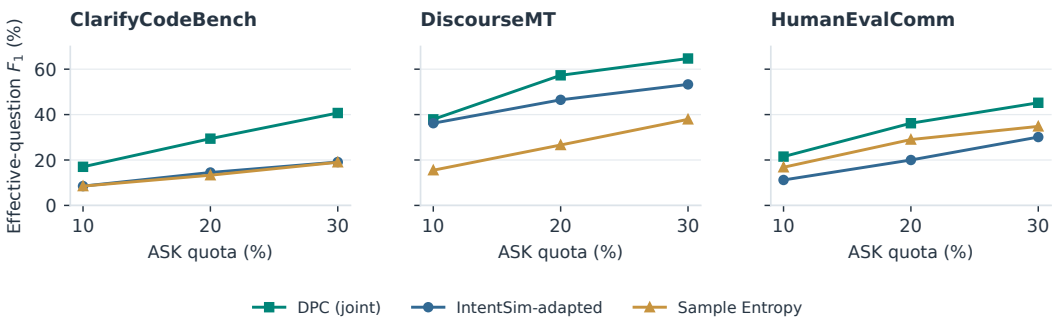

Figure 6: Effective-question F_1 at matched ASK budgets on the complete proxy panels. The metric follows Equation 10; it is not ordinary binary F_1 or downstream utility.

when taking the maximum; direct-score reconstruction removes this contamination. The displayed quota/action metrics already rank by pure entropy and are unchanged. No model output was regenerated. The companion audit script rejects duplicate or missing IDs, incomplete rows, and non-finite scores.

Average precision (AP) uses $\sum_k (R_k - R_{k-1})P_k$ at distinct score thresholds, including ties as groups; it is not trapezoidal PR area. Positive prevalence is 0.840 for CCB, 0.333 for DiscourseMT, and 0.823 for HumanEvalComm. These are also the APs of a constant-score predictor, explaining why high coding-panel AP alone is weak evidence.

Table 10: Full-panel AUROC / average precision, recomputed from frozen scores. DPC includes the explicit-conflict check.

Panel	DPC joint + check	IntentSim-adapted	Sample Entropy
CCB	0.653 / 0.909	0.505 / 0.845	0.465 / 0.836
DiscourseMT	0.856 / 0.737	0.695 / 0.546	0.535 / 0.362
HumanEvalComm	0.729 / 0.923	0.529 / 0.844	0.536 / 0.867

E Response-Probability Sensitivity and Statistical Interpretation

Using the exact policy counts in Table 3, Direct has mean reward $81/160$. Always Ask improves the full-response mean by $4/160$; each native selector improves it by $2/160$. Equation 5 therefore gives the values in Table 11. Minor differences in the last digit from older reports reflect decimal display conventions; this table recomputes only the arithmetic from the reported integer counts, without new model runs.

Table 11: Fixed-policy utility for $c = 0.02$. Reply probability is an offline mixture weight, not observed user behavior.

ρ	Direct	Always Ask + Preserve	SAGE-adapted native	DPC (cascade) native
0.25	0.506250	0.492500	0.505125	0.505750
0.50	0.506250	0.498750	0.508250	0.508875
0.75	0.506250	0.505000	0.511375	0.512000
1.00	0.506250	0.511250	0.514500	0.515125

For an observed full-response improvement $\delta > 0$ and ASK fraction b, the empirical break-even condition is $\rho > cb/\delta$. It yields 0.80, 0.34, and 0.29 for Always Ask, SAGE-adapted native, and DPC (cascade) native. The condition is specific to these observed outcomes and fixed policies. If response probability depends on question difficulty or intended specification, a single uniform mixture weight may be inappropriate.

The original revision-study analysis used 10,000 task-level resamples (seed 2026091702): it first averaged the three paired replicate differences for each task ID, then resampled 100 task IDs. We additionally reanalyzed the raw execution logs by resampling the 54 source-problem clusters (10,000 draws, seed 20260925) and retaining all variants belonging to each sampled source. The point estimates are unchanged; the source-clustered intervals are used in the main text.

Table 12: Paired bootstrap intervals for the repeated revision study. Both analyses are descriptive, unadjusted, and not preregistered.

Comparison	Difference (pp)	Task-level 95% CI	Source-clustered 95% CI
Preserve – Direct	+2.3	[0.3, 5.0]	[0.3, 4.9]
Preserve – Answer-only	+3.3	[0.7, 6.7]	[0.3, 7.0]
Preserve – Draft-aware	+1.7	[0.3, 3.3]	[0.4, 3.1]

Source-level paired sensitivity. We also sum Preserve-minus-comparator binary outcomes within each source problem, retaining all variants and replicates. Under a source-level sign-symmetry null, independently reversing each nonzero source sum gives the exact two-sided sign-flip distribution of the total paired difference. Table 13 reports the results. These post-hoc checks use the same adaptive study and are not confirmatory tests. They show how few sources support the aggregate advantage; unlike a percentile bootstrap over favorable empirical differences, the sign-flip distribution includes differences in both directions.

Table 13: Source-level sensitivity of Preserve's paired advantage. Favorable, unfavorable, and tied counts sum to 54 sources. The statistic is the sum of paired binary differences, not an unweighted count of favorable sources.

Comparator	Favorable	Unfavorable	Tied	Two-sided p
Direct	4	0	50	0.1250
Answer-only	5	1	48	0.1250
Draft-aware rewrite	5	0	49	0.0625

The raw-log join also identifies the denominator behind each transition. All three answer-conditioned pathways are invoked on the same 117 matched-answer pairs. Of these, 56 Direct defaults fail and are eligible for repair, while 61 pass and are eligible for regression.

Table 14: Conditional transitions on matched-answer pairs. Rates use the eligible default state, not all 300 task–replicate pairs, as denominator. Unchanged/revised is defined by exact equality with the Direct code.

Pathway	Matched	Repair	Repair rate	Regression	Regression rate	Unchanged / revised
Answer-only	117	5/56	8.9%	8/61	13.1%	–
Draft-aware	117	5/56	8.9%	3/61	4.9%	–
Preserve	117	7/56	12.5%	0/61	0.0%	45 / 72

The two-sided exact binomial 95% upper bound for zero regressions among 61 initially successful matched pairs is 5.9%. It is descriptive and ignores dependence among related sources and replicates; it is not a bound on deployment risk.

The 160-task policy bootstrap uses seed 20260917 and 10,000 task-level resamples. For the four favorable and zero unfavorable discordant Always-Ask/Direct outcomes, the exact two-sided paired test is $2(1/2)^4 = 0.125$. A percentile bootstrap can nevertheless exclude zero because resampling an empirical distribution with no observed harmful differences does not create such differences. This explains the inferential sensitivity; it does not make the exact test a correction for shared-source dependence or a replacement for a source-clustered analysis. The repeated revision study has a different pairing structure, so this p-value must not be transferred to its comparisons.

F Additional Results and Boundary Conditions

ClariQ and PACIFIC. The 111-topic operating-point comparison is shown in Table 15. Here precision, recall, and F_1 are ordinary binary necessity-label metrics, unlike the effective-question metric above. DPC asks fewer questions and has higher precision, but much lower recall. The ranking comparison also favors SAGE: AUROC is 0.714 versus 0.586, ordinal concordance is 0.706 versus 0.600, and extreme-level AUROC is 0.949 versus 0.808 over the 25 topics labeled 1 or 4. The gap therefore extends beyond the choice of decision threshold.

A disjoint, 80-topic ClariQ sample reverses this ordering: our cascade reaches AUROC 0.728 against 0.618, with ordinal concordance 0.712 against 0.645 and extreme-level AUROC 0.913 against 0.783. On the 100-topic PACIFIC panel (Deng et al., 2022), which uses its own native label, both methods are near chance and SAGE leads (0.570 vs 0.541), with identical precision-recall trade-offs at matched budgets. Neither selector consistently dominates these zero-shot transfer comparisons.

Table 15: ClariQ native operating points. Costs are estimated CNY for decision calls over all 111 topics under the token prices in Appendix G.

Policy	Acc. (%)	Prec. (%)	Rec. (%)	F_1	ASK (%)	Cost
DPC (cascade)	63.1	66.7	25.5	0.369	16.2	0.146
SAGE-adapted	65.8	56.5	83.0	0.672	62.2	0.293

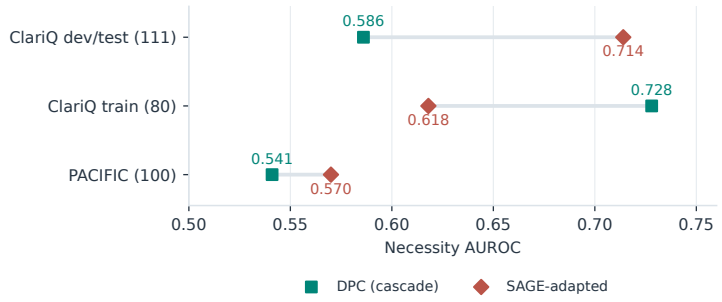

Figure 7: Cross-domain ordering is unstable. DPC (cascade) trails SAGE-adapted on ClariQ-111, leads on the disjoint ClariQ-80 split, and is slightly worse on PACIFIC-100.

DiscourseMT contrastive choice. The backend chooses translation A or B. For an ASK, it receives gold preceding context; context-free cases average across two hidden worlds. The full panel also contains 178 cases already supplied with context. Thus all reported accuracies are policy outcomes over the 267 cases, not conditional accuracy on the ASK subset. The implementation does not reproduce sequence log-likelihood evaluation of a translation model, and no natural-language translation quality claim follows.

Table 16: DiscourseMT contrastive-choice outcomes. Full-response and default-backed accuracies are over the whole panel. Utility uses the reported $\rho = 0.5$, $c = 0.02$ setting; values are retained at source precision.

Policy	ASK (%)	Full response (%)	Default-backed (%)	Utility
Direct	0.0	72.8	72.8	0.728
Always Ask	100.0	84.3	78.6	0.766
DPC joint + check	49.1	83.6	78.2	0.772
IntentSim-adapted	49.1	81.5	77.2	0.762
Sample Entropy	49.1	78.5	75.7	0.747
Ask-or-Act	88.0	83.7	78.3	0.765

A resolved-context check used the 178 already-contextualized cases in both mirrored orders (356 jobs, estimated CNY 0.228). It is not part of the above policy comparison.

Complete information conditions. On 120 CCB tasks, providing complete requirements increases pass@1 from 33.3% to 48.3%, a paired gain of $+15.0$ pp with cluster-bootstrap interval [5.6, 25.0] pp. On 180 HumanEvalComm tasks, the corresponding change is 59.4% to 96.1%, $+36.7$ pp with interval [29.1, 44.1] pp. These are paired single-sample outcomes from the same generation pathway. Gains vary markedly across CCB difficulty strata: $+43.5$ pp for easy, $+22.6$ pp for medium, and $+1.5$ pp for hard. Resolving the missing requirement has little effect on the hardest tasks. HumanEvalComm is stratified by prompt family rather than difficulty; the three multi-requirement families gain $+53$ to $+57$ pp while the three single-requirement families gain $+17$ to $+20$ pp.

These are reference results under richer information, not strict mathematical upper bounds: a particular generation pathway with complete requirements need not dominate every alternative pathway. They do not measure gains from the actual generated questions, and the stratification above shows why an average gain would overstate what a single well-chosen question can recover.

The separate 100-task HumanEvalComm revision check injects the full hidden requirement as an authoritative answer. Direct passes 51 tasks, Answer-only 94, Draft-aware 93, and Preserve 93. Preserve retains all 51 initially correct outcomes, but it does not outperform the answer-conditioned baselines. The condition tests preservation with unusually informative answers and is not evidence about question coverage.

The exploratory ClarifyBench tool pilot is retained in the companion experiment ledger, not as paper evidence: the available summary does not establish the official aggregation rules or cost provenance.

Question-match audit. A second API alias, `deepseek-v4-pro-0813`, audited 50 questions selected by stable-hash stratification: 25 primary-matcher hits and 25 misses. The audit did not display original labels or method names, but did see visible requests, questions, and gold ambiguity units. Agreement is 90%, Cohen's κ is 0.80, 24/25 primary hits are confirmed, and 21/25 primary misses remain misses. Five disagreements have not been adjudicated by humans. The balanced sample is not a prevalence-representative estimate of natural question quality. Because matching gates answers and bypass behavior, agreement alone does not eliminate downstream protocol sensitivity.

G Inference Configuration and Cost Accounting

Table 17: Main inference settings, all using `deepseek-flash-guan`. Other model aliases appear in Table 18. API reasoning mode is disabled through the provider-specific request field.

Component	Temperature	Maximum output tokens
Joint policy / context critic	0	2304
Explicit-conflict check	0	1024
Cascade Context	0	600
Cascade Value	0	1400
SAGE-adapted	0	1600
One-shot Question	0	384
Code revision robustness	0.2	4096
HumanEvalComm code / DiscourseMT choice	0	4096 / 256
Primary matcher	1	24 per vote
Uncertainty sampling	0.7	Method-specific manifest
Uncertainty clustering	0	Method-specific manifest

No explicit generation seed was supplied by the API. Resampling the default and revisions at temperature 0.2 produces three generation replicates, not three known seeds or independent model checkpoints. The CCB configuration allows two transport/format attempts; it does not select the best correct output. All final generation records have status `COMPLETE`; this denotes API completion, not successful test execution.

The representative cost panel contains 300 Context calls and 87 Value calls for DPC (cascade). Two-call joint uses 300 policy and 300 critic calls; joint with conflict checking adds 300 calls. Token-price estimates use CNY/M-token rates of 1.7775 for input, 0.0595 for cached input, and 5.3325 for output. These are estimates, not invoices. The matched total costs are CNY 1.389216 for two-call joint, 1.837112 for joint with conflict checking, 0.555496 for cascade, and 0.7598 for SAGE-adapted. An

Table 18: Executors used in this paper, by experiment family. The second-model audit and the controlled and cross-executor panels use different aliases and are never pooled with group 1–3 outcomes.

Executor alias	Experiment families and temperature
`deepseek-flash-guan`	Main revision study, policy panel, proxy panels, cascade, and boundary studies. Temperature 0 for controllers and 0.2 for revision.
`deepseek-v4-pro-0813`	Question-match audit. Temperature 0.
`gemini-3.8-flash`	Template cost-profile and family-disjoint development panels; CCB codegen reliability. Historical residual-value and context-cost diagnostics remain in the companion ledger. Temperature 0.
`gpt-5.4-nano`	Holdout-24 state-stability probe and the holdout-24 codegen arms (downstream codegen). Temperature 0.
`gpt-5.4`	Capacity probe. Temperature 0.

earlier summary omitted the conflict stage's cost while retaining its ranking scores; Table 4 corrects this by pairing each configuration's scores and complete decision-stage cost.

API time is summed across calls and averaged per task: 4.554 seconds for two-call joint, 6.171 for joint with conflict checking, and 2.393 for cascade. It is not concurrent throughput or user-experienced latency and excludes drafting, user simulation, revision, and testing. Relative to joint with conflict checking, cascade reduces decision tokens by 65.2%, estimated cost by 69.8%, and summed API time by 61.2%. Relative to two-call joint, the token reduction is 51.3%. Different prompts and call graphs prevent attribution to early exit alone.

Table 19: Additional cost of explicit-conflict checking, excluding policy and context-critic calls. Full-panel totals cover 1,692 task records (1,695 attempts, including three retries); all tasks had at least one issue. Costs are estimates from logged usage.

	Full panels			Matched 100 per panel	
Panel	Tasks	Tokens	CNY	Tokens	CNY
CCB	499	443,037	1.097317	88,346	0.219504
DiscourseMT	267	99,653	0.213583	37,166	0.080215
HumanEvalComm	926	573,739	1.417024	60,234	0.148177
Total	1,692	1,116,429	2.727924	185,746	0.447896

The conflict detector processes all candidate issues in one request per task, with a zero-token local bypass if no issues exist. It detects conflicts on 12 CCB, 6 DiscourseMT, and 333 HumanEvalComm tasks. On the full panels, adding it changes joint AUROC from 0.661/0.865/0.636 to 0.653/0.856/0.729, respectively. Its ranking benefit is concentrated in HumanEvalComm rather than uniform across panels. Table 19 accounts for its additional computation.

For the repeated revision study, the logged estimated costs total 1.134399 CNY for Answer-only, 0.910283 CNY for draft-aware rewriting, and 0.665412 CNY for Preserve. Each arm makes 117 calls over 300 evaluated task–replicate pairs; unmatched cases copy the default at zero incremental generation cost. Table 2 divides each total by three to report cost per 100 evaluated cases, not per 100 revision calls. The shared 300 default generations cost 3.220214 CNY and are excluded from this incremental comparison, as are question generation, answer matching, and test execution. The 26.9% reduction uses the unrounded revision totals and does not imply the same reduction in end-to-end cost.

Development-stage cost records are preserved separately in the companion ledger. They cover different combinations of reused generation, controller scoring, recovery calls, and execution, and are not used for a matched efficiency claim.

Operational interfaces. Context returns a textual contract rather than executable code. Value uses that contract rather than the default generated later for CCB. Native non-response is represented by the Direct branch in offline outcome mixtures. A user-facing default option, asynchronous timeout, and late-answer handling are not implemented in the code evidence summarized here. Reading

README files, repository code, and tests as contextual evidence is a possible extension, not an input capability demonstrated by the three proxy panels.

H A Recorded Matched-Answer Regression

The implementation audit identifies a first-replicate CCB case derived from source problem `abc396_g`. The request permits row and column flips in a binary grid and asks for the final number of ones, but omits the minimization objective. One-shot Question asks whether the desired count is arbitrary or the minimum over all flips. The matched gold answer specifies the minimum.

The recorded default draft uses a fast Walsh–Hadamard transform for XOR convolution. Its cost kernel is $\min(\mathrm{popcount}(m), W - \mathrm{popcount}(m))$, where W is the grid width and m encodes a row after column flips. The draft already solves the minimization objective and passes the hidden tests. Draft-aware global rewriting optimizes column flips alone, omitting the allowed row flips; the rewritten code fails. Preserve retains the original algorithm, removing only a comment, and still passes.

This is a compact paraphrase rather than a verbatim code diff. The default, revised outputs, and test outcomes were checked against the raw robustness run. The case concerns a matched answer and an actual revision call, not the automatic miss bypass. Its full trace is retained in the research archive (variant identifier beginning `1dd4ec30dbd0`, replicate 1).

I Version Map and Detailed Numerical Results

Table 20: Version and evidence map. Controller efficiency, proxy ranking, and code revision are not interchangeable evaluations of one frozen call graph.

Component	Implementation	Evidence reported here
Joint + conflict check	Joint policy, context critic, then explicit-conflict detector	Full proxy panels; matched-300 comparison; DiscourseMT policy replay
DPC (joint)	Joint policy + context critic, without conflict detector	Matched 300-case decision-cost study
DPC (cascade)	Context, then conditional Value	160-variant executable selection; 300-case decision cost; ClariQ transfer
Preserve	Draft + answer; prompt-constrained revision and unmatched-answer bypass	CCB paired revisions and fixed-policy replay
Controlled panels	Context/equivalence filter, cost-aware gate, default-backed action	Appendix J; not pooled with the rows above

Table 21 gives each generation replicate. Figure 3 plots the same values, while Table 2 reports pooled pass rates and transition counts.

Table 21: Pass@1 (%) in each paired generation replicate on the same 100 variants. Each replicate samples a new shared default, then applies the three answer-incorporation arms.

Operator	Run 1	Run 2	Run 3	Pooled
Direct	45.0	51.0	44.0	46.7
Answer-only	44.0	47.0	46.0	45.7
Draft-aware rewrite	44.0	51.0	**47.0**	47.3
Preserve	**48.0**	**52.0**	**47.0**	**49.0**

Earlier regeneration ablation. Table 22 reports the earlier single-run ablation. Its source run is `ccb_contrastive_revision/new100_originalq_revision_confirm_v1_001`, which shares its 100 tasks with the repeated study of Table 2 (verified: identical `task_public_id` sets). The two runs are otherwise distinct, and the ablation carries one generation per task–arm against the repeated study's three, so the two studies must not be read as replicates of one another.

Table 22: Earlier single-run ablation on 100 variants, using separately sampled defaults. Naive regeneration omits the draft anchor, so this comparison changes both draft access and revision instructions. Cost is estimated incremental generation cost per 100 evaluated cases.

Operator	pass@1 (%)	Δ vs Direct (pp)	Repairs	Regressions	CNY/100
Direct	48.0	–	–	–	–
Naive regeneration	42.0	−6.0	7	13	0.811
Preserve	51.0	+3.0	3	0	0.271

The matched-panel quality and controller-cost comparison is reported in main-text Figure 5 and Table 4; the larger full-panel AUROCs remain a separate evaluation.

J Controlled Development Panels

These template-built panels were used to develop context- and cost-aware selection. A case comprises a request, a candidate question, and hidden specifications. Labels are determined by the templates and their cost model, not human annotation. Consequently, beneficial-ask precision measures whether an interruption pays under that model; it is not comparable to necessity-label AUROC or public-benchmark pass@1.

The cost-profile panel uses 16 program families and 256 profiles, split into four development and 12 additional families. Each family includes ambiguous, clear-matching, clear-opposite, and behavior-invariant conditions. Of 64 ambiguous profiles, 32 are designed to benefit from asking. The recorded backend is `gemini-3.8-flash`. Table 23 retains the main development comparison; these controllers predate the public-panel joint and cascade implementations.

Table 23: Template cost-profile development panel (256 profiles). Sampling baselines use the same split-specific ASK budget as the development selector. The binary SAGE-style rule asks on every unresolved case; it is distinct from the richer public-panel SAGE adaptation.

Method	Asked	Ben. prec. (%)	Ben. rec. (%)	Action acc. (%)	Net utility
Development selector	30	100.00	93.75	99.22	0.93716
One-shot Question	34	91.18	96.88	98.44	0.93692
Sample Entropy	30	36.67	34.38	84.38	0.92935
IntentSim	30	16.67	15.63	79.69	0.92325
Prompted need score	30	13.33	12.50	78.91	0.92300
IntentSim-cost	0	–	0.00	87.50	0.93350
Binary SAGE-style / context-only	64	50.00	100.00	87.50	0.93350

The prompted need score is a local uncertainty judgment without an interaction-cost term (legacy code name `self_ask`), not a reproduction of a published Self-Ask reasoning system. The development selector ties One-shot Question on the four-family split; the small pooled advantage comes entirely from the other 12 families. On a separate 96-cell protocol using those 12 families, it trails context-only by 0.0081 utility and cost-matched One-shot Question by 0.0082. An eight-family, 64-cell action-level check finds beneficial-ask precision/recall of 100%/83.3%, but uses a different oracle. These results illustrate sensitivity to the cost and outcome protocol rather than establish a general advantage.

A separate executable-world panel (96 cells, 120 hidden records) and its follow-ups are retained in the companion ledger. Their manifests omit the model field, so they are not used as execution evidence in this paper.

K Cross-Executor and Placebo-Controlled Probes

A placebo reply controls for extra context and another generation call without adding task-specific information. The development probes in Table 24 use such controls, but differ in task sets and execution protocols from the main paired study. They do not establish cross-executor robustness.

Table 24: Exploratory answer-specificity checks. Rates, differences, and counts are labeled explicitly; they must not be pooled across rows.

Probe and executor	Observed comparison and limitation
6 questions × 3 samples, `gemini-3.8-flash`	Answered/Placebo/Direct pass rates: 0.611/0.333/0.278. Only 37 of 54 planned executions completed; 17 transport or format failures count as failures.
24 tasks × 3 replicates, `gpt-5.4-nano`	Mean answer-specific gain: $+0.0417$; mean placebo disturbance: -0.0417. Only one task is consistently improved; evaluation settings were aligned after generation.
14 tasks, four arms, `gpt-5.4`	True-answer, self-visible, and placebo arms each pass 3 tasks. Only 1 of 14 tasks is repaired specifically by the true answer.

The six-question Gemini probe contains both gains and regressions: three questions improve against both controls in all samples, but two favor the controls. On a nano-class executor, two other subsets reverse direction: answered versus placebo is 0.250 versus 0.375 on eight questions, and 0.625 versus 0.438 on a disjoint 16-question subset. Small panels and incomplete generations make these estimates inconclusive. The main study therefore compares revision operators on one primary backend, without claiming that the effect is unique to it or already replicated elsewhere.

L Inconclusive Development Studies

Table 25 summarizes development directions that did not yield evidence for the final method. Full run histories, including conflicting evaluation artifacts, are retained in the companion experiment ledger. None of these probes supports the main empirical claims.

Table 25: Development outcomes and reasons for exclusion from the main evidence.

Probe	Outcome or unresolved issue
Ambig-SWE value prediction	AUROC fell from 0.750 to 0.438 on a new panel; early ranking gains did not transfer.
Context and residual-value signals	Context classification reached only 0.646 accuracy on 48 cells. Oracle-question probes recovered context/materiality structure, but utility conclusions changed under post-hoc scoring and calibration did not transfer.
Question sufficiency and contract checks	Some accepted questions or certified repairs failed the hidden evaluation; public checks alone were insufficient.
Fixed-draft and regret gates	An unrelated answer changed an observable outcome in 23% of the causal-arm probe. A separate gate had precision 0.833 and recall 0.417, below its development targets.
Open-world and cost-only gates	The open-world gate asked on 0 of 6 valid panels; a calibrated cost-only variant also never asked.
SWE-bench-Live extension (Zhang et al., 2025)	An eight-task panel showed no truthful-answer effect in its causal arms; one- or two-task harness checks were too small for benchmark conclusions.
Transport and artifact checks	Missing or invalid outputs, incomplete execution, and inconsistent frozen decision files prevented valid comparisons in several runs.

M PROMPT TEXT AND PAYLOADS

The blocks below are literal system prompts extracted from the experimental scripts. They are included as reproducibility material, including internal terminology that differs from the manuscript labels. The legacy code name `SAFE_REVISION` refers to Preserve and does not assert a safety guarantee. Prompt strings and their SHA-256 hashes are also provided in the companion `prompts/` directory.

Inputs. Selectors receive the task instruction and visible input only. Joint and cascade value prompts return a proposed default contract; they do not receive generated code or hidden tests. The context critic additionally receives proposed issues. Revision prompts receive the original requirement, sampled draft (except Answer-only), delivered question, and matched answer. An explicit irrelevant-question response triggers the same programmatic copy of the default in all three answer-conditioned arms.

Context-critic version. The SHA-256 prompt hashes in all three full-panel run plans match `CRITIC_SYSTEM_V1`, reproduced below. Those results additionally use `CONFLICT_SYSTEM`: explicit incompatible requirements reopen an issue as unresolved, then its scalar score is deterministically recomputed from existing policy fields without another Value-model call. The matched 300-case comparison reports joint both with and without this check, with corresponding stage costs. Later v2 and v3 critic prompts are not substituted. The cascade uses its own context prompt, reproduced separately.

Scoring and execution. The selector score is computed outside the model from the returned fields using Equation 4. The joint code also distinguishes blocking and default-backed ASK using an irreversibility/rework score; the reported one-turn comparisons collapse both into ASK. Revision produces one complete program and uses no test feedback.

M.1 DEFAULT CODE GENERATION

Source: `run_ccb_real_how_downstream.py`
Symbol: `CODE_SYSTEM`.

```
Solve the programming task in Python 3. Return only one complete
solution inside a ```python code block. Read stdin and write stdout. The
↪  dialogue
may contain one clarification answer; use it when relevant. Do not ask
↪  another
question and do not discuss the solution outside the code block.
```

M.2 SHARED ONE-SHOT QUESTION

Source: `run_ccb_real_how_downstream.py`
Symbol: `QUESTION_SYSTEM`.

```
You are the clarification component for a programming assistant.
The user requirement may omit behavior that changes the correct program.
Ask exactly one concise, user-answerable question targeting the single
↪  most
important unresolved behavior. Do not solve the task, list several
↪  questions,
or assume hidden information. Return only JSON: {"question": "..."}.
```

M.3 DEFAULT-PRESERVING REVISION

Source: `run_ccb_contrastive_revision.py`
Symbol: `REVISION_SYSTEM`.

```
You are revising a default Python solution after one authoritative
clarification. The clarification changes only the behavior it explicitly
↪  addresses;
all other requirements remain unchanged. First check silently whether
↪  the draft
conflicts with the answer. If it does not, preserve the draft. If it
↪  does, make the
smallest correctness-preserving edit. Return only one complete solution
↪  inside a
```python code block. Read stdin and write stdout. Do not ask questions
↪  or explain.
```

M.4 ANSWER-ONLY GENERATION

Source: `run_ccb_revision_robustness.py`
Symbol: `ANSWER_ONLY_SYSTEM`.

```
Solve the programming task in Python 3 using the supplied
authoritative clarification. Return only one complete solution inside a
↪  ```python
code block. Read stdin and write stdout. Do not ask questions or
↪  explain.
```

M.5 DRAFT-AWARE GLOBAL REWRITING

Source: `run_ccb_revision_robustness.py`
Symbol: `DRAFT_AWARE_REWRITE_SYSTEM`.

```
Produce the best final Python solution after reviewing
an existing draft and one authoritative clarification. Use the draft as
↪  a useful
starting point, but independently verify the whole algorithm and freely
↪  rewrite any
part when that could improve correctness. Preserve correct behavior
↪  where appropriate.
Return only one complete solution inside a ```python code block. Read
↪  stdin and write
stdout. Do not ask questions or explain.
```
```

M.6 JOINT POLICY

Source: `run_public_external_selector.py`
Symbol: `POLICY_SYSTEM`.

```
You are a clarification controller for an assistant. The downstream task
↪  may be coding, question answering, NLI, or translation. Read only
↪  the task instruction and visible input. Do not solve the task. State
↪  a concise default behavioral contract that the downstream assistant
↪  can follow immediately if the user never replies. Propose only
↪  user-answerable questions whose answers could change the externally
↪  observable task output. Do not ask merely because the task is
↪  difficult, because your own knowledge is weak, or because wording
↪  admits equivalent paraphrases. An independent critic will later
↪  determine whether the full visible context already resolves each
↪  candidate, so do not perform that gate yourself. Each issue must
↪  contain one focused question and one explicit default. Estimate
↪  probabilities in [0,1]. Return JSON only:
{"default_contract":string,"issues":[{"issue_id":string,"question":stri
↪  ng,"default_answer":string,"fallback_behavior":string,"p_default_co
↪  rrect":number,"p_behavior_change":number,"p_user_can_answer":number
↪  ,"p_backend_uptake":number,"irreversibility":number,"rework_cost":n
↪  umber,"user_effort":number,"reason":string}]}
```

M.7 JOINT CONTEXT CRITIC, VERSION 1

Source: `run_public_context_critic.py`
Symbol: `CRITIC_SYSTEM_V1`.

```
You are a strict context-entailment critic. You receive a task
↪  instruction, the complete visible input, and candidate clarification
↪  issues. Do not solve the task and do not use outside conventions.
↪  For each issue, decide whether the visible input itself settles the
↪  answer.

RESOLVED: a specific visible statement, example, or conversation context
↪  entails one answer strongly enough to act without asking. Cite the
↪  sufficient visible evidence.
EQUIVALENT: multiple answers remain but they cannot change externally
↪  observable task behavior.
UNRESOLVED: multiple behavior-changing answers remain, or the claimed
↪  answer relies on an unstated convention, the absence of a statement,
↪  world knowledge, or model inference. "Usually", "standard
↪  convention", and "not mentioned" are not resolution evidence.

Return every issue exactly once as JSON only:
{"issues":[{"issue_id":string,"context_relation":"RESOLVED"|"EQUIVALENT
↪  "|"UNRESOLVED","sufficient_visible_evidence":string,"reason":string
↪  }]}
```

M.8 CASCADE CONTEXT CHECK

Source: `run_real_context_value_cascade.py`
Symbol: `CONTEXT_SYSTEM`.

```
You are the cheap first stage of a clarification cascade. Read the task
↪  instruction and all visible input. Decide only whether the visible
↪  context already specifies one externally observable behavior
↪  strongly enough to act. Do not estimate the benefit or cost of
↪  asking, do not solve the task, and do not ask about model knowledge
↪  or implementation strategy. Use adjacent discourse, examples,
↪  signatures, stated referents, and other visible semantic frames as
↪  disambiguating evidence. Do not keep remote dictionary senses alive
↪  when one reading is clearly coherent in context.

RESOLVED: visible evidence entails or strongly selects one behavior;
↪  contextual coherence is sufficient even without an explicit
↪  paraphrase. EQUIVALENT: alternatives cannot change externally
↪  observable output. UNRESOLVED: at least two contextually plausible
↪  user-intended behaviors remain and would change the output. Mere
↪  lexical polysemy or a theoretically imaginable alternative is
↪  insufficient. Absence of a statement is not by itself resolution
↪  evidence.

Return JSON only:
{"context_status":"RESOLVED"|"EQUIVALENT"|"UNRESOLVED","default_contrac⌋
↪  t":string,"unresolved_aspect":string,"visible_evidence":string,"rea⌋
↪  son":string}
```

M.9 CASCADE VALUE ASSESSMENT

Source: `run_real_context_value_cascade.py`
Symbol: `VALUE_SYSTEM`.

```
You are the expensive value stage of a clarification controller. The
↪  cheap context stage has already found a behavior-changing unresolved
↪  aspect. Propose exactly one focused, user-answerable question and an
↪  explicit default that lets the backend continue if the user does not
↪  reply. Estimate each factor in [0,1]. Do not ask about
↪  implementation strategy, facts the assistant should know, or
↪  equivalent paraphrases.

Return JSON only:
{"question":string,"default_answer":string,"fallback_behavior":string,"⌋
↪  p_default_correct":number,"p_behavior_change":number,"p_user_can_an⌋
↪  swer":number,"p_backend_uptake":number,"irreversibility":number,"re⌋
↪  work_cost":number,"user_effort":number,"reason":string}
```

M.10 SAGE ADAPTATION

Source: `run_sage_evpi_adapted_baseline.py`
Symbol: `SAGE_SYSTEM`.

```
You implement a single-turn adaptation of SAGE-Agent's structured
↪  uncertainty module. Read the task instruction and complete visible
↪  input. Do not use outside hidden information and do not solve
↪  ordinary model uncertainty by asking the user. Treat an adjacent
↪  discourse context, explicit example, signature, or stated referent
↪  as valid disambiguating evidence. A merely lexical or theoretically
↪  imaginable alternative is not a live interpretation when the visible
↪  context makes one reading clearly coherent.

Enumerate 1-6 mutually exclusive, contextually live interpretations that
↪  would cause different externally observable outputs. Assign
↪  calibrated prior probabilities summing approximately to 1. If
↪  visible context entails or strongly selects one behavior, set
↪  resolved_by_context=true and do not invent remote alternatives.
↪  Otherwise generate up to 3 focused, user-answerable clarification
↪  questions. For each question, partition every interpretation id into
↪  exactly one answer group according to how the user's answer would
↪  identify the intended behavior. Questions about implementation
↪  strategy, facts the assistant should know, or equivalent outputs are
↪  invalid.

Return JSON only:
{"resolved_by_context":boolean,"interpretations":[{"id":string,"behavio⌋
↪  r":string,"probability":number}],"questions":[{"question":string,"t⌋
↪  arget_aspect":string,"answer_groups":[{"answer_label":string,"candi⌋
↪  date_ids":[string]}]}],"reason":string}
```

M.11 INTENTSIM ADAPTATION: BATCHED ANSWERS

Source: `run_public_when_strong_baselines.py`
Symbol: `INTENT_SIM_SYSTEM`.

```
Simulate ten plausible direct user answers to the available
↪  clarification question. Condition only on the visible task and
↪  question. Explicit visible context is binding: if it already entails
↪  one answer, the samples should consistently express that answer. The
↪  wording of the candidate question and any suggested/default option
↪  are not evidence about the user's intent. If a behavior-changing
↪  choice remains unresolved, preserve the plausible intent
↪  uncertainty: for a binary or contrastive question, affirmative and
↪  negative answers can both be plausible unless visible context favors
↪  one. Do not deliberately maximize diversity, but do not collapse the
↪  samples merely because the question is phrased in a leading way.

Return JSON only with exactly:
{"answers":[string,string,string,string,string,string,string,string,str⌋
↪  ing,string]}
```

M.12 SAMPLE ENTROPY ADAPTATION: BATCHED PREDICTIONS

Source: `run_public_when_strong_baselines.py`
Symbol: `SAMPLE_ENTROPY_SYSTEM`.

```
Simulate ten plausible direct assistant predictions made without asking
↪  a clarification question. Condition only on the visible task. For
↪  translation, output the predicted translation; for classification or
↪  QA, output the predicted label or answer; for coding, output a
↪  concise observable behavior contract describing the assumptions the
↪  generated program would implement, not source code. Samples should
↪  reflect natural model uncertainty rather than deliberately
↪  maximizing diversity.

Return JSON only with exactly:
{"predictions":[string,string,string,string,string,string,string,string
↪  ,string,string]}
```

M.13 INTENT CLUSTERING

Source: `run_public_when_strong_baselines.py`
Symbol: `CLUSTER_INTENTS_SYSTEM`.

```
Cluster the ten simulated user answers by behaviorally relevant user
↪  intent. Answers that require the same externally observable task
↪  output belong to the same cluster even if worded differently. Use
↪  consecutive integer ids starting at 0 in first-occurrence order.

Return JSON only with exactly:
{"cluster_ids":[integer,integer,integer,integer,integer,integer,integer
↪  ,integer,integer,integer],"reason":string}
```

M.14 PREDICTION CLUSTERING

Source: `run_public_when_strong_baselines.py`
Symbol: `CLUSTER_OUTPUTS_SYSTEM`.

```
Cluster the ten direct assistant predictions by externally observable
↪  task behavior or final answer. Surface paraphrases and
↪  implementation details with identical judged behavior belong to the
↪  same cluster. Use consecutive integer ids starting at 0 in
↪  first-occurrence order.

Return JSON only with exactly:
{"cluster_ids":[integer,integer,integer,integer,integer,integer,integer
↪  ,integer,integer,integer],"reason":string}
```

M.15 ASK-OR-ACT BINARY PROMPT

Source: `run_public_when_strong_baselines.py`
Symbol: `AOA_SYSTEM`.

```
You are an Intent Agent monitoring an assistant before it acts. Decide whether the visible task contains ambiguity or missing user-intent information that can materially change the required output. Do not ask because the task is merely difficult or because you lack knowledge. Explicit visible context can settle intent. The available candidate question is supplied only as a possible clarification target.

Return JSON only with exactly:
{"needs_clarification":boolean,"reason":string}
```

M.16 FULL-PANEL EXPLICIT-CONFLICT CHECK

Source: `run_public_conflict_override.py`
Symbol: `CONFLICT_SYSTEM`.

```
You are a narrow visible-evidence conflict detector. You receive a task instruction, the complete visible input, and candidate clarification issues. Do not solve the task and do not use private references or outside conventions.

For each issue, set conflict=true only when at least two explicit, relevant parts of the visible input (for example prose, function signature, examples, tests, or conversation statements) require incompatible answers to that issue. Quote both sides briefly. Mere ambiguity, missing information, broad wording, model uncertainty, or lack of evidence is NOT a conflict. If there are not two explicit incompatible pieces, return conflict=false.

Return every issue exactly once as JSON only:
{"issues":[{"issue_id":string,"conflict":boolean,"evidence_a":string,"evidence_b":string,"reason":string}]}
```