RECAP: FROM REQUIREMENT RECOVERY TO RELIABLE EXECUTION IN CONTEXT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models are increasingly used in long-context settings where the context defines temporary knowledge, rules, procedures, and constraints. Many failures are near misses: models capture the main task but miss some low-salience requirements. Yet improving requirement recall alone can produce redundant and fragmented representations that are difficult to execute, making context learning a joint problem of broad requirement recovery and executable representation. We study whether such representations can be discovered from empirical feedback rather than designed entirely by hand. To this end, we perform feedback-driven search over a structured space of executable inference pipelines, treating each edit as a falsifiable hypothesis and retaining regressions, ineffective edits, and non-transferring changes as constraints for subsequent search. Within this structured search space, feedback-driven evaluation supports broad requirement enumeration followed by consolidation into a compact representation before execution. This design yields RECAP, which constructs such a representation at inference time and conditions generation on it. For weaker models, we further introduce RECAP-RL, which learns to execute self-constructed requirement representations using rubric-level rewards. On CL-bench, RECAP improves Claude Opus-4.8 from 21.09% to 27.04% strict accuracy and Qwen3.5-9B from 11.74% to 15.60%, while RECAP-RL further raises Qwen3.5-9B to 17.96%. The approach also transfers to complex instruction following and multimodal context learning. These results suggest a general recipe for context learning: discover an effective requirement representation, construct it explicitly, and learn to execute it.

1 INTRODUCTION

Large language models are increasingly used in long-context settings where context provides not only evidence but also task-specific requirements (Jin et al., 2026a; Wei et al., 2026b; Si et al., 2026; Ren et al., 2026a). Such contexts may define local terminology, rules, procedures, output formats, exceptions, and constraints that are absent from model parameters (Wei et al., 2026a; Jin et al., 2026b; Zhong et al., 2026; Gu et al., 2026; Chen et al., 2025). Models must infer these context-specific requirements and apply them correctly to the current task (Ren et al., 2026b; Qi et al., 2026; Chen et al., 2026; Shen et al., 2026), an ability we refer to as *context learning*.

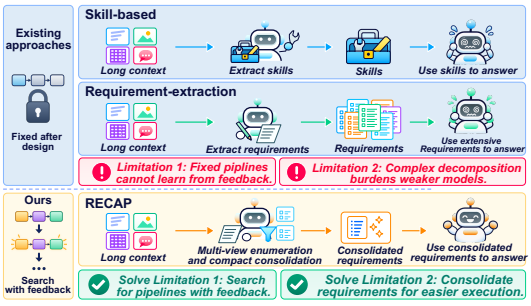

Figure 1: Comparison with existing context learning pipelines. Unlike fixed decomposition, we search requirement-representation designs from feedback: RECAP pairs multi-view enumeration with consolidation into a compact, executable representation.

Current models remain unreliable at context learning. On CL-bench (Dou et al., 2026), for example, Claude Opus-4.8 solves only 21.09% of tasks. Many failures are near misses: the model captures the main task but misses a required field, exclusion rule, or other low-salience obligation (Jin et al., 2026a; Wang et al., 2026a;b). Under all-or-nothing task scoring, a single missed requirement can invalidate an otherwise strong response.

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.

A natural approach is therefore to make requirements explicit before answering. Recent test-time methods decompose context into task specifications or reusable skills (Si et al., 2026; Zhong et al., 2026). Yet improving requirement recall creates a second challenge: extracting more requirements can produce redundant or overly detailed intermediate representations that are themselves difficult to execute, especially for weaker models (Figure 1). Context learning thus involves a fundamental tension: *requirements must be recovered broadly, yet represented in a form that models can reliably execute*. This raises a question: *what representation best balances coverage and executability, and can it be discovered from empirical feedback rather than designed entirely by hand?*

We study this question using feedback-driven search as a *discovery instrument*. Rather than allowing unconstrained program rewriting, we search a structured space of executable context-learning pipelines (Figure 2(A)). Each modification changes a single component and is treated as a falsifiable hypothesis; candidates are evaluated with matched decoding seeds, while regressions, ineffective edits, and transfer failures are retained as constraints for subsequent search. The search yields two findings: *requirement recall matters*, but *recall alone is insufficient*. Over-complete enumeration helps recover low-salience requirements, yet the resulting set becomes increasingly redundant and fragmented. The most effective intervention is to consolidate this broad set into a compact representation that reduces execution burden while aiming to retain substantive requirements.

These findings yield RECAP (**R**equirement **E**numeration, **C**onsolidation, **A**nswering, and **P**atching). RECAP enumerates requirements from complementary views, consolidates them into a compact set of actionable requirements, conditions answer generation on this representation, and applies guarded additive patching only when residual coverage can be improved (Figure 2(B)). In short, RECAP operationalizes a simple principle: *recover broadly, consolidate before execution*. For weaker models, however, constructing a useful representation does not guarantee reliable execution. We therefore introduce RECAP-RL, which uses the model itself to construct requirement sets offline and trains answering conditioned on these self-constructed representations using rubric-level rewards (Figure 2(C)). Unlike supervised approaches based on teacher-generated answers or reasoning traces (Jin et al., 2026b), RECAP-RL improves the model's ability to execute its self-constructed representation. Our framework thus follows a unified progression: *discover* an effective requirement representation, *construct* it at inference time, and *learn to execute* it. Experiments support this progression across model scales, tasks, and modalities. The proposed methods consistently improve context-learning performance and enhance complex instruction following and multimodal context learning, providing an effective interface between long-context understanding and reliable execution.

Our main contributions are fourfold: (1) We formulate context learning as a joint problem of *requirement recovery* and *representation executability*, and use feedback-driven search to provide empirical evidence that broad, multi-view enumeration followed by consolidation is an effective design; (2) We introduce RECAP, which constructs a compact requirement representation at inference time through multi-view enumeration, consolidation, coverage-oriented answering, and guarded patching; (3) We introduce RECAP-RL, which learns to execute the same representation using self-constructed requirements and rubric-level rewards, without distilling teacher-generated answers; and (4) We show that RECAP and RECAP-RL are both effective and broadly generalizable: validated across six backbones and three task families, they deliver consistent gains—RECAP improves Claude Opus-4.8 by +5.95 strict accuracy on CL-bench and by 11.5 points on average on AgentIF and MultiChallenge, two complex instruction-following benchmarks spanning agentic and multi-turn dialogue instructions, while RECAP-RL, trained using only text, improves Qwen3.5-9B's accuracy by 9.90 points on MCL-bench, a multimodal context-learning benchmark with image- and video-grounded contexts, demonstrating broad transferability across tasks and modalities.

2 Problem Setting

We study context-learning tasks: a model receives a source context c and a user task x, and must produce an answer y. Each task carries a rubric set $R^\star = \{r_1, \ldots, r_m\}$ in which every item states a necessary condition for a correct answer, and $\mathrm{pass}(y, r_j) \in \{0, 1\}$ marks whether y satisfies item r_j under the rubric grader. For a single task we score the all-or-nothing $\mathrm{strict}(y, R^\star) = \prod_{j=1}^{m} \mathrm{pass}(y, r_j)$ and the adherence rate $\mathrm{micro}(y, R^\star) = \frac{1}{m} \sum_{j=1}^{m} \mathrm{pass}(y, r_j)$. Over an evaluation set $\{(y_i, R_i^\star)\}_{i=1}^{N}$ we report strict accuracy, the mean of strict across tasks, and micro adherence, the pooled ratio $\big(\sum_i \sum_j \mathrm{pass}(y_i, r_{i,j})\big) / \sum_i m_i$ that pools rubric items across all tasks before dividing.

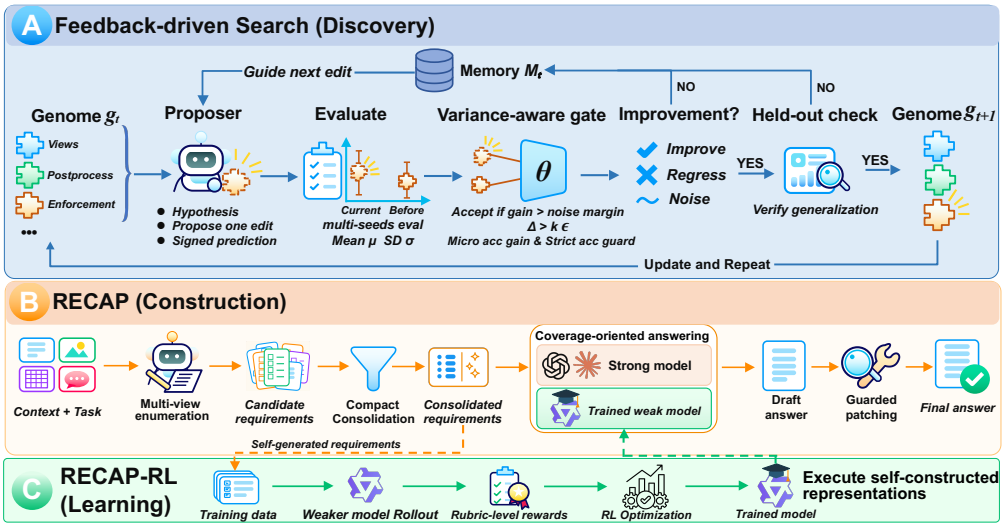

Figure 2: Overview of our framework. **(A) Discovery:** feedback-driven search over requirement-representation designs. **(B) Construction:** RECAP combines multi-view enumeration with consolidation into a compact requirement representation at inference time. **(C) Learning to Execute:** RECAP-RL trains models to execute self-constructed representations with rubric-level rewards.

3 METHOD

3.1 DISCOVERING EXECUTABLE REQUIREMENT REPRESENTATIONS

Rather than prescribing a fixed context decomposition strategy, we ask which intermediate representations best support reliable requirement execution. We formulate pipeline design as a controlled empirical search: given a constructed development set $\mathcal{D}_{\text{dev}} = \{(c_i, x_i, R_i^\star)\}_{i=1}^n$, we explore executable pipelines and use evaluation feedback to retain only changes that pass a variance-aware acceptance test and generalize to a held-out set (Figure 2(A)).

Structured search space. We represent each test-time pipeline as an executable genome $g \in \mathcal{G}$, where executing g with backbone model π induces an answering algorithm $A_g^\pi : (c, x, s) \mapsto y$, with s denoting a decoding seed. We restrict $\mathcal{G}$ to a structured, bounded space rather than allow arbitrary code rewriting, so that the effect of each intervention remains attributable. The structural genes control the number of enumeration views, whether and how aggressively to transform the enumerated requirement set, and the enforcement strategy—emit the draft unchanged, additively patch it, or select best-of-m (full genome in Appendix A). The genome also contains the stage instructions and reading lenses that shape how each stage is prompted. Each proposal modifies exactly one structural gene, stage instruction, or lens list while leaving the control flow fixed, so any gain or regression attaches to a specific intervention rather than a confounded rewrite. Search starts from Direct, a single-pass pipeline without additional structure.

A falsification-driven search loop. At iteration t, the search maintains a current genome g_t, a search history $\mathcal{H}_t$, and a constraint memory $\mathcal{M}_t$, and treats each modification as a falsifiable hypothesis about the current failure mode. Given a failure report together with $\mathcal{H}_t$ and $\mathcal{M}_t$, a proposer model (GPT-5.5) outputs a triple (h_t, e_t, p_t), where h_t describes a hypothesized failure mechanism, e_t is a schema-checked edit producing $g_t' = e_t(g_t)$, and p_t is a signed prediction of the edit's effect on the evaluation metrics, which the subsequent evaluation then confirms or refutes. Candidates are evaluated with the discovery backbone GPT-5.1 over multiple decoding seeds using both strict accuracy and micro adherence. Because strict accuracy is all-or-nothing and thus high-variance at development scale, we use the denser micro adherence as the primary acceptance signal and strict accuracy as a regression guard, while tracking the per-seed standard deviation of both.

Variance-aware acceptance and constraint memory. Each candidate is evaluated on the development set using the same seeds as its parent, so comparisons share decoding noise. For

$\bullet \in \{\text{micro}, \text{strict}\}$, let $\widehat{\mu}_\bullet(g)$ and $\widehat{\sigma}_\bullet(g)$ denote the mean and per-seed standard deviation of metric $\bullet$ for genome g. Define $\Delta_\bullet = \widehat{\mu}_\bullet(g'_t) - \widehat{\mu}_\bullet(g_t)$ and $\sigma_\bullet = \max(\sqrt{\widehat{\sigma}_\bullet(g'_t)^2 + \widehat{\sigma}_\bullet(g_t)^2}, \sigma_{\min})$, where $\sigma_{\min}$ prevents the threshold from collapsing under near-zero observed variance. With $k = 1$, a candidate passes the variance-aware test only if

$$\Delta_{\text{micro}} > k\sigma_{\text{micro}} \qquad \text{and} \qquad \Delta_{\text{strict}} \geq -k\sigma_{\text{strict}}.$$

Thus, its micro adherence gain must exceed observed variability while any strict accuracy loss remains within tolerance. Passing this test yields only provisional acceptance: the candidate is adopted as g_{t+1} only if its gain transfers to a context-disjoint validation set; otherwise, it is rejected as a NON-TRANSFER. Outcomes are classified as ACCEPTED IMPROVEMENT, FLAGGED REGRESSION, NOISE, or NON-TRANSFER. Only failures enter the memory $\mathcal{M}_t$: regressions induce *do-not* constraints, noise records ineffective directions, and non-transfers discourage context-specific edits. Accepted improvements instead update the current genome and are recorded in the search history $\mathcal{H}_t$. Hence, $\mathcal{M}_t$ accumulates persistent negative and null evidence that constrains subsequent search.

What the search discovers. Figure 7 summarizes the search trajectory and rejected branches. Two findings emerge. First, *requirement recall matters*: reconstructing requirements improves over Direct, and adding complementary reading views initially brings further gains, whereas pruning requirements or emphasizing downstream filtering reduces strict accuracy. Second, *recall alone is insufficient*: as more requirements are enumerated, their union becomes increasingly redundant and fragmented. Consolidating this over-complete set while preserving its substantive constraints yields the largest reliable gain. A single guarded additive patch recovers near misses and is retained, but going beyond it—looping the patch, best-of-m selection, and gate-aware rewriting—provides no consistent benefit. Feedback thus consistently favors a simple design: *recover requirements broadly, then consolidate them before execution.* Instantiating this design yields RECAP.

3.2 RECAP: CONSTRUCTING THE REPRESENTATION AT INFERENCE TIME

The discovery process suggests that effective context learning requires an intermediate representation that balances broad coverage of context-derived requirements with low redundancy at execution time. RECAP operationalizes this by reconstructing a compact requirement set $\hat{R}$ from context c and task x, and conditioning answer generation on this representation, in four stages: multi-view enumeration, compact consolidation, coverage-oriented answering, and guarded patching (Figure 2(B)).

***Multi-view* requirement enumeration.** Let $\mathcal{V} = \{v_1, \ldots, v_K\}$ denote complementary reading views, with $K = 4$ in the final discovered configuration. The views target different aspects of the context, including content completeness, formatting requirements, explicit constraints, and output specifications explicitly stated in the source. Each view produces atomic, checkable requirements $Q_k = E_\pi(c, x, v_k)$, whose deduplicated union is $U = \text{dedup}(\bigcup_{k=1}^{K} Q_k)$. This stage deliberately aims to recover requirements broadly: diverse views expose low-salience obligations missed by a single reading, with redundancy deferred for removal in the next stage.

***Compact* requirement consolidation.** Broad enumeration also produces redundant, fragmented, or overly specific requirements, which inflate the instruction burden placed on the answerer. RECAP therefore computes $\hat{R} = C_\pi(U, c, x; \tau)$, where τ optionally controls the target representation size. The consolidator merges overlapping or fragmented candidates with the aim of retaining substantive constraints, producing a compact set of distinct and actionable requirements. Consolidation is therefore not simple pruning: it aims to reduce redundancy while retaining distinct, substantive requirements, reorganizing them into a set that is easier for the answerer to follow. We verify empirically that this design goal is met—consolidation cuts set size and near-duplication sharply while retaining 94% of the ground-truth requirements recovered by enumeration (Section 4.6).

***Coverage-oriented* answering.** The answerer receives the original context, task, and consolidated requirement representation, and samples $y_0 \sim \pi(\cdot \mid c, x, \hat{R}, I_{\text{ans}})$, where I_{ans} instructs the model to satisfy each reconstructed requirement and emit only the final answer. In this way, $\hat{R}$ acts as an explicit interface between context understanding and answer generation, letting the model attend to a compact, already-organized control signal instead of repeatedly recovering from the raw context.

***Guarded* additive patching.** A model auditor evaluates y_0 against $\hat{R}$ and identifies the estimated unmet requirements V_0. If $V_0 = \emptyset$, RECAP returns y_0; otherwise, a completion editor constructs $y_1 = \mathrm{Patch}_\pi(c, x, \hat{R}, y_0, V_0)$ by adding the missing content while preserving the existing answer. A second audit accepts y_1 only if estimated coverage does not decrease and high-priority misses do not increase: $\mathrm{met}(y_1) \geq \mathrm{met}(y_0)$ and $u_{\mathrm{hp}}(y_1) \leq u_{\mathrm{hp}}(y_0)$, where $\mathrm{met}(y)$ is the auditor-estimated number of satisfied requirements and $u_{\mathrm{hp}}(y)$ is the number of estimated unmet high-priority requirements; otherwise, RECAP retains y_0.

3.3 RECAP-RL: Learning to Execute the Representation

RECAP builds a more executable requirement representation at inference time, but weaker models may still struggle to turn it into a fully compliant answer. We therefore study whether a model can be trained to execute that representation more reliably. RECAP-RL reuses RECAP's enumeration-and-consolidation front end to construct explicit requirement sets offline, and trains the model to answer conditioned on these self-constructed representations (Figure 2(C)). The goal is to improve the model's ability to execute a representation it can itself construct.

Training data. GPT-5.1 synthesizes training examples spanning the four CL-bench categories (Jin et al., 2026b); the full corpus statistics and sources are reported in Appendix D. Each example contains a self-contained context c, a reusable system instruction, and several context-dependent tasks. A task consists of a question x and atomic binary training rubrics $R^{\mathrm{tr}} = \{r_1, \ldots, r_m\}$, which are used only for reward computation and are never exposed to the trained model. For each (c_i, x_i), the model runs only the requirement-construction front end of RECAP—multi-view enumeration followed by consolidation, without answering or patching—to produce $\hat{R}_i$, and then samples $y_i \sim \pi_\theta(\cdot \mid c_i, x_i, \hat{R}_i)$. The model is thus trained to answer conditioned on requirement representations generated by its own requirement-construction pipeline.

Rubric-level reward. A separate open-source judge, Qwen3.8-27B, evaluates each answer against rubrics, producing the mean rubric-adherence reward $r_i = \frac{1}{m_i}\sum_{j=1}^{m_i} \mathrm{pass}_J(y_i, r_{i,j})$, where $\mathrm{pass}_J(y, r) \in \{0, 1\}$ indicates whether answer y satisfies rubric r. This signal is dense: it gives partial credit for partially satisfied requirements and directly optimizes the mapping from a model-generated requirement representation to a compliant answer.

Optimization. We optimize π_θ with GRPO (Shao et al., 2024). For each prompt, we sample G answers from the behavior policy $\pi_{\theta_{\mathrm{old}}}$ and normalize their rewards within the group as $A_i = (r_i - \bar{r})/(\mathrm{std}(\mathbf{r}) + \epsilon_{\mathrm{std}})$, where $\mathbf{r} = \{r_1, \ldots, r_G\}$, $\bar{r}$ is the group mean, and ϵ_{std} is a small constant for numerical stability. Following Cheng et al. (2026), we reweight token-level updates using policy entropy as $\tilde{A}_{i,t} = A_i + \min(\alpha H_{i,t}^{\mathrm{detach}}, |A_i|/\kappa)$, where $H_{i,t}$ is the entropy at token t, α scales the entropy bonus, and κ bounds its contribution relative to $|A_i|$. Detaching the entropy blocks gradients through its computation, while the clip bounds its contribution relative to the reward advantage. Let $\rho_{i,t}(\theta) = \pi_\theta(y_{i,t} \mid c_i, x_i, \hat{R}_i, y_{i,<t})/\pi_{\theta_{\mathrm{old}}}(y_{i,t} \mid c_i, x_i, \hat{R}_i, y_{i,<t})$ denote the token-level importance ratio. The resulting objective is

$$\mathcal{J}(\theta) = \mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|y_i|}\sum_{t=1}^{|y_i|}\min\Big(\rho_{i,t}(\theta)\tilde{A}_{i,t}, \mathrm{clip}\big(\rho_{i,t}(\theta), 1-\epsilon, 1+\epsilon\big)\tilde{A}_{i,t}\Big)\right] - \beta\,\mathrm{KL}(\pi_\theta \,\|\, \pi_{\mathrm{ref}})\,,$$

where ϵ is the clipping range, β is the KL coefficient, and π_{ref} is a frozen reference policy. Full optimization details and hyperparameters are provided in Appendix D.

4 Experiments

4.1 Experimental Setup

Backbones. We evaluate RECAP as a pure test-time method on all six backbones: Claude Opus-4.8, GPT-5.1, GPT-4.1, Qwen3.5-9B, Qwen3.5-4B, and Llama-3.2-3B-Instruct. For Qwen3.5-9B, Qwen3.5-4B, and Llama-3.2-3B-Instruct, we additionally train RECAP-RL checkpoints and evaluate them with both Direct inference and RECAP.

Table 1: CL-bench results across different models. Subscripts denote absolute changes from direct inference. Best values are **bold**; shaded rows are ours; † denotes values taken from original papers.

		Overall		DKR		RSA		PTE		EDS	
Backbone	**Method**	Strict	Rubric	Strict	Rubric	Strict	Rubric	Strict	Rubric	Strict	Rubric
Claude Opus-4.8	Direct	21.09	81.71	23.60	84.53	19.43	78.58	23.03	83.21	12.76	73.86
	Ctx2Skill	$22.71_{+1.62}$	$82.30_{+0.59}$	24.02	84.98	21.06	78.63	25.43	84.07	16.67	76.30
	PSCI	$26.34_{+5.25}$	$83.33_{+1.62}$	**29.39**	**86.48**	23.19	79.88	29.12	84.40	18.46	76.50
	RECAP	**27.04**$_{+5.95}$	**83.78**$_{+2.07}$	27.37	86.32	**24.33**	**81.11**	**32.62**	**84.93**	**20.31**	**76.52**
GPT-5.1 (medium)	Direct	22.18	82.05	23.87	84.95	21.55	79.56	23.57	82.39	15.08	76.70
	Ctx2Skill	$23.18_{+1.00}$	$81.46_{-0.59}$	24.92	85.03	21.91	77.95	25.05	82.18	16.58	74.38
	PSCI†	**28.14**$_{+5.96}$	$81.13_{-0.92}$	**31.67**	84.84	25.18	76.70	29.15	81.04	**21.21**	79.12
	RECAP	$27.78_{+5.60}$	**83.31**$_{+1.26}$	29.91	**87.00**	**25.44**	**79.84**	**30.43**	**83.05**	21.11	**79.91**
GPT-4.1	Direct	14.01	76.86	14.50	79.85	15.90	**74.89**	13.38	77.46	8.54	68.72
	Ctx2Skill†	$16.50_{+2.49}$	$72.75_{-4.11}$	16.82	76.65	**17.58**	70.70	17.61	72.59	9.69	63.07
	PSCI	$16.81_{+2.80}$	$77.46_{+0.60}$	**17.98**	81.35	16.25	73.50	18.47	78.12	10.55	70.64
	RECAP	**17.19**$_{+3.18}$	**78.24**$_{+1.38}$	16.79	**81.36**	16.81	74.07	**20.38**	**79.67**	**12.06**	**73.30**
Llama-3.2-3B-Instruct	Direct	3.01	52.29	2.72	55.91	2.30	47.78	4.71	56.19	**2.01**	38.05
	Ctx2Skill	$2.58_{-0.43}$	$51.89_{-0.40}$	2.41	54.20	2.30	47.29	3.82	56.91	1.01	39.16
	PSCI	$2.11_{-0.90}$	$43.67_{-8.62}$	1.36	45.76	2.48	40.66	3.20	47.66	1.02	29.45
	RECAP	$3.50_{+0.49}$	$51.10_{-1.19}$	2.29	53.99	**5.14**	46.51	4.50	56.94	0.51	32.37
	RECAP-RL + Direct	$3.53_{+0.52}$	$57.15_{+4.86}$	2.56	59.91	3.00	53.04	6.58	62.01	1.01	41.66
	RECAP-RL + RECAP	**4.42**$_{+1.41}$	**58.90**$_{+6.61}$	**3.17**	**62.05**	4.77	**54.09**	**6.79**	**64.16**	**2.01**	**42.48**
Qwen3.5-4B	Direct	10.70	70.49	11.03	73.78	11.31	68.22	10.85	71.38	7.54	60.93
	Ctx2Skill	$8.49_{-2.21}$	$68.35_{-2.14}$	8.46	71.25	8.30	64.75	10.21	70.66	5.05	59.27
	PSCI	$11.40_{+0.70}$	$71.14_{+0.65}$	12.99	74.70	10.78	66.98	12.34	73.95	5.61	58.30
	RECAP	$11.80_{+1.10}$	$70.61_{+0.12}$	11.89	74.03	11.68	66.03	12.90	72.84	9.18	62.51
	RECAP-RL + Direct	$9.93_{-0.77}$	$72.39_{+1.90}$	11.21	75.86	9.72	68.95	10.00	74.08	6.06	62.83
	RECAP-RL + RECAP	**14.17**$_{+3.47}$	**76.17**$_{+5.68}$	**16.47**	**79.73**	**12.37**	**71.62**	**14.65**	**77.30**	**10.55**	**71.69**
Qwen3.5-9B	Direct	11.74	71.91	12.22	74.97	12.01	68.70	11.46	73.01	10.05	65.29
	Ctx2Skill	$13.11_{+1.37}$	$73.37_{+1.46}$	12.82	77.08	11.70	68.89	16.17	74.92	10.77	65.01
	PSCI	$14.56_{+2.82}$	$74.99_{+3.08}$	15.30	78.48	15.60	71.14	13.22	76.30	**12.24**	67.19
	RECAP	$15.60_{+3.86}$	$75.53_{+3.62}$	15.56	79.34	14.66	70.85	18.51	77.23	11.56	68.46
	RECAP-RL + Direct	$14.38_{+2.64}$	$76.47_{+4.56}$	15.38	79.86	12.90	71.60	16.56	78.96	10.05	68.98
	RECAP-RL + RECAP	**17.96**$_{+6.22}$	**79.80**$_{+7.89}$	**19.46**	**82.67**	**17.14**	**75.70**	**20.81**	**82.56**	8.54	**71.12**

Benchmarks. Our primary benchmark is CL-bench (Dou et al., 2026), comprising 1,899 long-context learning tasks and 31,607 rubric items across four families: domain-knowledge reasoning (DKR), rule-system application (RSA), procedural task execution (PTE), and empirical discovery & simulation (EDS), with contexts averaging 10.4K tokens. We evaluate cross-task transfer on AgentIF (Qi et al., 2025) and MultiChallenge (Deshpande et al., 2025), and cross-modal transfer on MCL-bench, a multimodal context-learning benchmark.[1] We further use GPQA-Diamond (Rein et al., 2023), AIME 2024/2025/2026, and MMLU-Pro (Wang et al., 2024) to assess whether RECAP-RL preserves general capabilities. We use GPT-5.1 with low reasoning effort to evaluate CL-bench and GPT-5.5 with default reasoning effort for all other benchmarks. On CL-bench, we report all-or-nothing strict accuracy and pooled rubric-level micro adherence. All CL-bench results are averaged over three independent runs.

Baselines and training. We compare RECAP against Direct, Ctx2Skill (Si et al., 2026), and PSCI (Zhong et al., 2026) on the same backbone. For the smaller backbones, RECAP-RL + Direct denotes direct inference with the trained checkpoint, while RECAP-RL + RECAP applies RECAP to the same checkpoint. Full training details are provided in Appendix D.

4.2 Main Results on CL-bench

RECAP consistently improves context learning. Across all six backbones, RECAP improves strict accuracy over Direct, while rubric-level accuracy improves on five of six, indicating broad gains in both task-level success and fine-grained requirement adherence. Among the evaluated test-time methods, RECAP achieves the highest overall strict accuracy on five of six backbones and outperforms Ctx2Skill on all six. For example, Claude Opus-4.8 improves from 21.09% to 27.04% strict accuracy and from 81.71% to 83.78% rubric accuracy. On GPT-5.1, PSCI is slightly higher in strict accuracy (28.14% vs. 27.78%), while RECAP achieves higher rubric accuracy (83.31% vs. 81.13%). The sole rubric-level exception is Llama-3.2-3B-Instruct, where rubric accuracy decreases slightly from 52.29% to 51.10% despite improved strict accuracy, suggesting that the smallest backbone may still struggle to consistently execute reconstructed requirements.

RECAP-RL complements test-time RECAP on weaker models. Across all three smaller backbones, RECAP-RL + RECAP achieves the highest overall strict and rubric-level accuracy. On Qwen3.5-9B, RECAP-RL + Direct improves from 11.74% to 14.38% strict accuracy, while combining the trained checkpoint with RECAP further raises performance to 17.96% strict and 79.80% rubric accuracy. Similar gains appear on Qwen3.5-4B and Llama-3.2-3B-Instruct. Notably, RECAP-RL alone does not uniformly improve strict accuracy, whereas RECAP-RL + RECAP con-

[1] https://huggingface.co/datasets/GFRbsc/Multimodal-CL-Bench

Table 2: Transfer to multimodal context learning on MCL-bench, grouped by dialogue turn depth.

		Overall		Turn 1		Turn 2		Turn 3		Turn ≥4	
Backbone	**Method**	Strict	Rubric	Strict	Rubric	Strict	Rubric	Strict	Rubric	Strict	Rubric
Qwen3.5-4B	Direct	10.94	51.29	8.85	44.54	13.57	52.05	10.09	55.50	11.74	55.93
	Ctx2Skill	$10.38_{-0.56}$	$50.90_{-0.39}$	11.21	41.88	11.07	52.28	9.21	58.69	9.39	54.29
	PSCI	$8.40_{-2.54}$	$48.73_{-2.56}$	8.55	41.92	7.50	48.19	9.21	54.92	8.45	52.90
	RECAP	$11.23_{+0.29}$	$50.54_{-0.75}$	10.32	45.54	8.93	47.82	11.40	55.41	15.49	56.07
	RECAP-RL + Direct	$13.77_{+2.83}$	$56.63_{+5.34}$	11.80	46.13	14.64	58.34	14.91	62.80	14.55	**63.61**
	RECAP-RL + RECAP	**17.64**$_{+6.70}$	**59.18**$_{+7.89}$	**14.16**	**49.88**	**17.50**	**59.13**	**22.37**	**68.67**	**18.31**	62.97
Qwen3.5-9B	Direct	11.89	55.08	12.09	47.41	11.07	55.83	9.21	58.90	15.49	61.45
	Ctx2Skill	$12.92_{+1.03}$	$54.04_{-1.04}$	12.68	44.11	12.50	55.79	10.53	60.23	16.43	60.19
	PSCI	$11.98_{+0.09}$	$55.30_{+0.22}$	11.50	49.79	12.14	54.09	10.53	59.49	14.08	60.44
	RECAP	$11.89_{+0.00}$	$54.50_{-0.58}$	12.09	49.75	9.64	53.71	12.72	59.21	13.62	57.48
	RECAP-RL + Direct	$17.26_{+5.37}$	$59.01_{+3.93}$	15.34	51.18	17.14	60.42	18.42	64.79	19.25	62.83
	RECAP-RL + RECAP	**21.79**$_{+9.90}$	**63.65**$_{+8.57}$	**17.11**	**54.82**	**24.29**	**65.91**	**18.86**	**67.77**	**29.11**	**69.71**

Table 3: Transfer to complex instruction following tasks on AgentIF and MultiChallenge.

		Overall	AgentIF			MultiChallenge			
Backbone	**Method**	**Mean**	**CSR**	**ISR**	**Avg**	IR	IM	RVE	SC
Claude Opus-4.8	Direct	43.2	61.7	29.0	57.4	56.5	62.8	56.1	54.0
	PSCI	$52.5_{+9.3}$	$67.1_{+5.4}$	$39.9_{+10.9}$	$65.1_{+7.7}$	65.2	70.8	58.5	**66.0**
	RECAP	**54.7**$_{+11.5}$	**67.9**$_{+6.2}$	**40.9**$_{+11.9}$	**68.4**$_{+11.0}$	**72.5**	**74.3**	**61.0**	**66.0**
GPT-5.1	Direct	47.7	64.0	38.9	56.5	66.7	71.7	41.5	46.0
	PSCI	$50.6_{+2.9}$	$66.0_{+2.0}$	$39.0_{+0.1}$	$62.2_{+5.7}$	69.6	69.9	**51.2**	58.0
	RECAP	**53.0**$_{+5.3}$	**71.0**$_{+7.0}$	**41.3**$_{+2.4}$	**64.7**$_{+8.2}$	**72.5**	**75.2**	**51.2**	**60.0**
GPT-4.1	Direct	34.2	**64.1**	32.2	36.2	53.6	40.7	24.4	26.0
	PSCI	$41.7_{+7.5}$	$61.9_{-2.2}$	$33.9_{+1.7}$	**49.5**$_{+13.3}$	**60.9**	48.7	**46.3**	**42.0**
	RECAP	**43.4**$_{+9.2}$	$63.7_{-0.4}$	**38.6**$_{+6.4}$	$48.2_{+12.0}$	56.5	**50.4**	43.9	**42.0**
Llama-3.2-3B-Instruct	Direct	15.7	48.3	**14.3**	17.2	20.3	12.4	22.0	14.0
	PSCI	$19.8_{+4.1}$	$46.4_{-1.9}$	$13.7_{-0.6}$	$25.8_{+8.6}$	**36.2**	22.1	**26.8**	18.0
	RECAP	$15.9_{+0.2}$	$46.1_{-2.2}$	$10.5_{-3.8}$	$21.3_{+4.1}$	18.8	24.8	19.5	22.0
	RECAP-RL + Direct	$14.4_{-1.3}$	**48.7**$_{+0.4}$	$13.4_{-0.9}$	$15.3_{-1.9}$	15.9	14.2	17.1	14.0
	RECAP-RL + RECAP	**20.9**$_{+5.2}$	$48.3_{+0.0}$	$13.6_{-0.7}$	**28.2**$_{+11.0}$	30.4	**28.3**	22.0	**32.0**
Qwen3.5-4B	Direct	35.3	58.4	24.9	45.7	63.8	54.9	34.1	30.0
	PSCI	$37.2_{+1.9}$	$54.7_{-3.7}$	$23.9_{-1.0}$	$50.5_{+4.8}$	59.4	45.1	**41.5**	56.0
	RECAP	$39.2_{+3.9}$	$56.9_{-1.5}$	$26.5_{+1.6}$	$51.8_{+6.1}$	**68.1**	48.7	36.6	54.0
	RECAP-RL + Direct	$37.8_{+2.5}$	**59.8**$_{+1.4}$	$27.1_{+2.2}$	$48.5_{+2.8}$	62.3	57.5	34.1	40.0
	RECAP-RL + RECAP	**42.2**$_{+6.9}$	$56.8_{-1.6}$	**28.0**$_{+3.1}$	**56.4**$_{+10.7}$	63.8	**63.7**	34.1	**64.0**
Qwen3.5-9B	Direct	42.6	60.6	29.6	55.6	71.0	66.4	39.0	46.0
	PSCI	$42.7_{+0.1}$	$58.3_{-2.3}$	$27.6_{-2.0}$	$57.7_{+2.1}$	60.9	59.3	48.8	**62.0**
	RECAP	**47.7**$_{+5.1}$	**62.2**$_{+1.6}$	**32.7**$_{+3.1}$	**62.7**$_{+7.1}$	**72.5**	66.4	**56.1**	56.0
	RECAP-RL + Direct	$44.6_{+2.0}$	$60.7_{+0.1}$	$29.8_{+0.2}$	$59.3_{+3.7}$	66.7	**76.1**	46.3	48.0
	RECAP-RL + RECAP	$45.1_{+2.5}$	$61.4_{+0.8}$	$31.8_{+2.2}$	$58.4_{+2.8}$	65.2	68.1	46.3	54.0

sistently gives the strongest overall results. This supports the expected complementarity: RECAP-RL improves execution of reconstructed requirements, while RECAP provides the explicit requirement representation at inference time. RECAP-RL also preserves general capabilities (Figure 11).

4.3 Generalization Beyond CL-bench

Transfer to instruction following and multimodal context learning. We further evaluate the generalization of RECAP and RECAP-RL beyond text-based context learning. On MCL-bench, which contains 339 image/video-grounded dialogues and 1,060 turn-level tasks with each turn conditioned on the full preceding dialogue, Table 2 shows that RECAP-RL transfers effectively from text-only training, and combining it with RECAP improves strict accuracy from 11.89% to 21.79% on Qwen3.5-9B. On AgentIF and MultiChallenge, Table 3 shows that RECAP consistently improves instruction-following performance across stronger backbones, increasing the mean score from 43.2 to 54.7 on Claude Opus-4.8, while RECAP-RL further improves requirement execution on weaker models. These results demonstrate that the requirement-centric framework generalizes effectively across modalities, instruction complexity, and model capability levels.

Why context learning and complex instruction following share a bottleneck. Both context learning and complex instruction following require models to transform dense specifications into executable actions. In context learning, requirements are implicit and distributed across contexts, while in complex instruction following, they are explicit but numerous and interacting. Despite these differences, both suffer from the same bottleneck: incomplete requirement recovery and insufficient coverage during generation. RECAP addresses this bottleneck by constructing compact, actionable requirement representations before generation. Since it operates on requirements rather than their surface forms, RECAP naturally extends to complex instructions and multimodal contexts.

4.4 RECAP Ablations

Table 4 analyzes RECAP from two perspectives. **(a) requirement recovery and consolidation.** Requirement reconstruction improves over Direct, and adding diverse views further helps initially.

Table 4: RECAP pipeline ablation on CL-bench (GPT-5.1, medium). Details in Appendix A.

(a) Building the RECAP pipeline

Configuration	Strict	Rubric
Direct answer	22.18	82.05
+ req. reconstruction (1 view)	25.03	83.13
+ 2nd complementary view	26.71	**83.69**
+ 3rd complementary view	26.08	79.25
+ requirement patching	27.17	79.50
+ source view + robust answer	26.52	83.01
coverage-oriented answer	27.46	80.97
+ coarse consolidation	26.98	83.57
+ tight consolidation (RECAP)	**27.78**	83.31

(b) Rejected alternatives and controls

Configuration	Strict	Rubric
RECAP	**27.78**	**83.31**
Requirement-processing alternatives		
repeated patching ($\times 3$)	26.64	79.63
remove simple requirements	24.33	80.31
gate-aware rewriting	25.97	78.51
best-of-m selection	27.36	79.17
Budget-matched controls (no reconstruction)		
direct + self-refine ($\times 3$)	24.16	82.35
best-of-8 direct answers	25.21	82.77

Table 5: RECAP-RL ablation on CL-bench over requirement source and reward scheme.

				Overall		DKR		RSA		PTE		EDS	
Backbone	**Req. source**	**Reward**	**Mode**	Strict	Rubric	Strict	Rubric	Strict	Rubric	Strict	Rubric	Strict	Rubric
Qwen3.5-4B	GPT-5.5	Rubric-mean	Direct	11.64	73.10	12.22	75.53	12.54	69.64	11.68	75.66	7.04	64.94
			RECAP	13.53	74.25	14.48	76.29	**13.07**	70.09	**15.07**	77.03	8.04	68.93
	GPT-5.5	All-or-nothing	Direct	9.32	72.03	11.61	75.07	8.13	68.97	9.55	73.73	4.52	63.17
			RECAP	12.53	71.45	12.07	73.67	12.90	67.87	14.65	73.64	8.04	65.72
	Self	Rubric-mean	Direct	9.93	72.39	11.21	75.86	9.72	68.95	10.00	74.08	6.06	62.83
			RECAP	**14.17**	**76.17**	**16.47**	**79.73**	12.37	**71.62**	14.65	**77.30**	**10.55**	**71.69**
Qwen3.5-9B	GPT-5.5	Rubric-mean	Direct	12.34	76.13	13.75	79.31	12.01	72.45	12.98	77.66	7.04	69.18
			RECAP	15.97	78.16	17.22	80.82	15.19	73.90	17.66	80.81	**10.05**	**71.20**
	GPT-5.5	All-or-nothing	Direct	11.43	73.51	12.67	77.38	10.60	70.84	11.89	74.54	8.54	62.39
			RECAP	14.85	75.93	15.23	79.36	15.02	71.88	16.56	78.08	9.05	66.48
	Self	Rubric-mean	Direct	14.38	76.47	15.38	79.86	12.90	71.60	16.56	78.96	**10.05**	68.98
			RECAP	**17.96**	**79.80**	**19.46**	**82.67**	**17.14**	**75.70**	**20.81**	**82.56**	8.54	**71.12**

However, excessive enumeration introduces redundancy and hurts rubric accuracy. Guarded patching recovers near-miss cases, while consolidation is crucial: combining source-specification analysis, coverage-oriented answering, and final consolidation achieves the strongest performance. **(b) gains stem from structured requirements, not compute.** Enumeration–patch variants, including pruning, iterative patching, gate-aware rewriting, and best-of-m selection, all underperform RECAP. Budget-matched baselines with a comparable number of backbone calls but without requirement reconstruction (self-refine and best-of-N direct) also fall short, showing that RECAP gains come from building and executing a compact requirement representation.

4.5 RECAP-RL Ablations

Self-constructed requirements provide better-aligned supervision. Self-constructed requirements are competitive with those generated by GPT-5.5, yielding the highest rubric accuracy under RECAP on both Qwen3.5 backbones and the strongest overall results on Qwen3.5-9B (Table 5). Interestingly, GPT-5.5 requirements achieve higher training rewards in Figure 3, but this does not consistently translate into better downstream performance. This suggests that alignment with the model's own requirement representation matters more than reward magnitude alone.

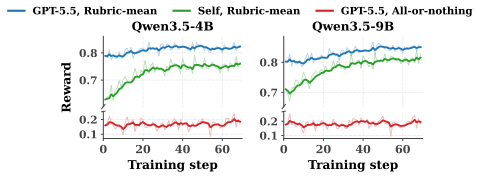

Figure 3: RECAP-RL training dynamics.

Rubric-level rewards mitigate reward sparsity. All-or-nothing rewards provide credit only when every requirement is satisfied, whereas Rubric-mean provides dense, requirement-level feedback for partial progress. This denser signal yields smoother training dynamics and stronger final performance in Figure 3. All-or-nothing training can even reduce rubric-level accuracy relative to Direct-mode inference, as observed for Qwen3.5-4B and Llama-3.2-3B-Instruct (Tables 5 and 11).

4.6 Analysis: Why Consolidation Matters

Consolidation turns over-complete enumeration into a compact representation. Figure 4 shows that raw enumeration produces 27–112 requirements per task across backbones, well above the CL-bench average of 16 ground-truth rubrics per task. After consolidation, the range shrinks to 14–24, with five of six backbones between 14 and 18. Combined with the finding that simple pruning

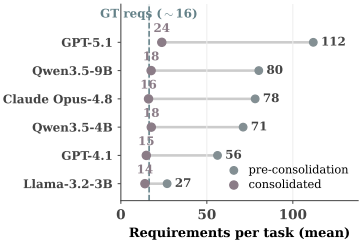

Figure 4: Mean number of requirements before and after consolidation; dashed line marks the ground-truth rubric count.

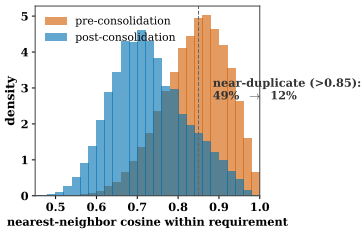

Figure 5: Nearest-neighbor similarity of requirements before and after consolidation.

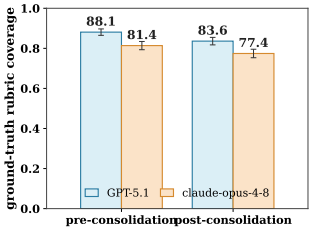

Figure 6: Ground-truth rubric coverage before and after consolidation; error bars are 95% CIs over tasks.

hurts performance, this suggests that consolidation helps not merely by reducing the number of requirements, but by transforming an over-complete set into a compact representation that is easier to execute reliably.

Consolidation removes semantic redundancy. We embed each task's requirements with BGE-M3 and define the nearest-neighbor similarity of requirement i as $s_i = \max_{j \neq i} \langle \mathbf{e}_i, \mathbf{e}_j \rangle$ after $L2$ normalization. The near-duplicate rate is $\rho = \frac{1}{N} \sum_i \mathbf{1}[s_i > 0.85]$. As shown in Figure 5, across 300 CL-bench tasks answered by GPT-5.1, consolidation reduces the near-duplicate rate from 49% to 12% and mean nearest-neighbor similarity from 0.84 to 0.73; the near-duplicate rate decreases on 293 of 300 tasks. Thus, much of the inflation in raw requirement sets comes from semantically overlapping restatements that consolidation removes before generation. Using GPT-5.5 as an entailment judge, we count a ground-truth rubric item as covered if some generated requirement expresses or entails it, and report the fraction covered, averaged over the 300 tasks. Consolidation lowers coverage only marginally (GPT-5.1: $88.1\% \rightarrow 83.6\%$; claude-opus-4-8: $81.4\% \rightarrow 77.4\%$), retaining 94.2% and 92.8% of the items covered beforehand (Figure 6). Consolidation thus substantially reduces redundancy while preserving most of the ground-truth rubric coverage.

5 Related Work

Context learning for LLMs. Unlike long-context benchmarks that primarily evaluate retrieval and reasoning over inputs (Hsieh et al., 2024; Bai et al., 2025), context learning requires models to infer and apply temporary rules, formats, and exceptions from context (Wei et al., 2026a; Zhou et al., 2026). Recent work improves this ability with explicit intermediate representations and structured inference (Si et al., 2026; Zhong et al., 2026; Lu et al., 2026; Liu et al., 2026; Yu et al., 2026). We build on this direction, studying how requirement representations support reliable execution.

Feedback-driven search and automated optimization. Feedback-driven search has been used to optimize prompts, inference procedures, and agentic systems (Ning et al., 2026; Xu et al., 2026; Jin et al., 2026c; Zhang et al., 2026b; Ouyang et al., 2026; Zhang et al., 2026a). Agentic Harness Engineering (Lin et al., 2026) evolves coding-agent components from trajectory evidence. In contrast, we search over context-learning pipelines to discover principles for constructing executable requirement representations.

6 Conclusion

We study context learning as recovering and representing requirements from long contexts for reliable execution. Our analysis shows that broad enumeration alone is insufficient: requirements must be consolidated into compact, executable representations. This motivates RECAP, which builds such representations at inference time via multi-view enumeration and consolidation, with guarded patching for residual gaps. For weaker models, RECAP-RL trains execution from self-constructed requirements using rubric-level rewards, without distilling teacher answers or reasoning traces. Across CL-bench, AgentIF, MultiChallenge, and multimodal MCL-bench, RECAP and RECAP-RL deliver consistent gains across models, tasks, and modalities.

AI USE STATEMENT

LLMs were used for research ideation and execution, synthetic data generation, and language polishing. GPT-5.5 and GPT-5.1 supported feedback-driven pipeline discovery and evaluation. GPT-5.5 was used to synthesize the development and validation data, while GPT-5.1 was used to synthesize the RECAP-RL training data. GPT-5.5 was also used to polish the language of the initial version of this manuscript. The authors reviewed all AI-assisted content and take full responsibility for the final manuscript.

REFERENCES

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, et al. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3639–3664, 2025.

Guanzheng Chen, Xin Li, Michael Qizhe Shieh, and Lidong Bing. Longpo: Long context self-evolution of large language models through short-to-long preference optimization. In *International Conference on Learning Representations*, volume 2025, pp. 6638–6654, 2025.

Yifan Chen, Fei Yin, Qingyan Bai, Zicheng Lin, and Yujiu Yang. Mmcl-bench: Multimodal context learning from visual rules, procedures, and evidence. *arXiv preprint arXiv:2605.12703*, 2026.

Daixuan Cheng, Shaohan Huang, Xuekai Zhu, Bo Dai, Xin Zhao, Zhenliang Zhang, and Furu Wei. Reasoning with exploration: An entropy perspective. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 30377–30385, 2026.

Kaustubh Deshpande, Ved Sirdeshmukh, Johannes Baptist Mols, Lifeng Jin, Ed-Yeremai Hernandez-Cardona, Dean Lee, Jeremy Kritz, Willow E Primack, Summer Yue, and Chen Xing. Multichallenge: A realistic multi-turn conversation evaluation benchmark challenging to frontier llms. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 18632–18702, 2025.

Shihan Dou, Ming Zhang, Zhangyue Yin, Chenhao Huang, Yujiong Shen, Junzhe Wang, Jiayi Chen, Yuchen Ni, Junjie Ye, Cheng Zhang, et al. Cl-bench: A benchmark for context learning. *arXiv preprint arXiv:2602.03587*, 2026.

Zhuohan Gu, Qizheng Zhang, Omar Khattab, and Samuel Madden. Peek: Context map as an orientation cache for long-context llm agents. *arXiv preprint arXiv:2605.19932*, 2026.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Hongbo Jin, Chi Wang, Haoran Tang, Zhongjing Du, Xu Jiang, Jingqi Tian, Qiaoman Zhang, and Jiayu Ding. Contextguard: Structured self-auditing for context learning in language models. *arXiv preprint arXiv:2605.26827*, 2026a.

Hongbo Jin, Mingnan Zhu, Jingqi Tian, Xu Jiang, Zhongjing Du, Haoran Tang, Siyi Xie, Qiaoman Zhang, and Jiayu Ding. Context-cot: Enhancing context learning via high-quality reasoning synthesis. *arXiv preprint arXiv:2605.25354*, 2026b.

Jiajie Jin, Yuyang Hu, Kai Qiu, Qi Dai, Chong Luo, Guanting Dong, Xiaoxi Li, Tong Zhao, Xiaolong Ma, Gongrui Zhang, et al. Toward generalist autonomous research via hypothesis-tree refinement. *arXiv preprint arXiv:2606.11926*, 2026c.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, et al. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*, 2026.

Hongyi Liu, Haoyan Yang, Tao Jiang, Bo Tang, Feiyu Xiong, Yuyu Luo, and Zhiyu Li. Skillsvote: Lifecycle governance of agent skills from collection, recommendation to evolution. *arXiv preprint arXiv:2605.18401*, 2026.

Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. Skill0: In-context agentic reinforcement learning for skill internalization. *arXiv preprint arXiv:2604.02268*, 2026.

Xuying Ning, Katherine Tieu, Dongqi Fu, Tianxin Wei, Zihao Li, Yuanchen Bei, Jiaru Zou, Mengting Ai, Zhining Liu, Ting-Wei Li, et al. Code as agent harness. *arXiv preprint arXiv:2605.18747*, 2026.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, volume 2026, pp. 94327–94354, 2026.

Jinhu Qi, Minda Hu, Wentao Zhang, Weiqiang Jin, Yanyu Chen, Junli Wang, and Irwin King. Compile, don't memorize: A context compilation architecture (cca) for in-context learning. *arXiv preprint arXiv:2609.00759*, 2026.

Yunjia Qi, Hao Peng, Xiaozhi Wang, Amy Xin, Youfeng Liu, Bin Xu, Lei Hou, and Juanzi Li. Agentif: Benchmarking instruction following of large language models in agentic scenarios. *arXiv preprint arXiv:2505.16944*, 2025.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Qingyu Ren, Qianyu He, Powei Chang, Jie Zeng, Zeye Sun, Fei Yu, Jiaqing Liang, and Yanghua Xiao. Instructions are all you need: Self-supervised reinforcement learning for instruction following. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4755–4776, 2026a.

Tao Ren, Weiyao Luo, Hui Yang, Rongzhi Zhu, Xiang Huang, Yuchuan Wu, Bingxue Chou, Jieping Ye, Jiafeng Liang, Yongbin Li, et al. Scaling self-evolving agents via parametric memory. *arXiv preprint arXiv:2606.04536*, 2026b.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Weixiang Shen, Bailiang Jian, Jun Li, Che Liu, Johannes Moll, Xiaobin Hu, Daniel Rueckert, Hongwei Bran Li, and Jiazhen Pan. Evo-medagent: Beyond one-shot diagnosis with agents that remember, reflect, and improve. *arXiv preprint arXiv:2604.14475*, 2026.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

Shuzheng Si, Haozhe Zhao, Yu Lei, Qingyi Wang, Dingwei Chen, Zhitong Wang, Zhenhailong Wang, Kangyang Luo, Zheng Wang, Gang Chen, et al. From context to skills: Can language models learn from context skillfully? *arXiv preprint arXiv:2604.27660*, 2026.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024.

Yudong Wang, Zhe Yang, Wenhan Ma, Rang Li, Qibin Yang, Weimin Xiong, Jiangshan Duo, Liang Zhao, and Zhifang Sui. From refuse to richness: Rubric rewards for long-form hallucination reinforcement learning. *arXiv preprint arXiv:2608.12337*, 2026a.

Yuyao Wang, Zhongjian Zhang, Mo Chi, Kaichi Yu, Yuhan Li, Miao Peng, Bing Tong, Chen Zhang, Yan Zhou, and Jia Li. Evomembench: Benchmarking agent memory from a self-evolving perspective. *arXiv preprint arXiv:2605.18421*, 2026b.

Lai Wei, Chengqi Li, Jiapeng Li, Ruina Hu, Yue Wang, and Weiran Huang. Clbench-v: Evaluating multimodal context learning from grounding to knowledge acquisition. *arXiv preprint arXiv:2607.25294*, 2026a.

Tianxin Wei, Zhan Shi, Minhua Lin, Bing He, Zewen Liu, Yisi Sang, Yuanchen Bei, Xuying Ning, Jiaru Zou, Ting-Wei Li, et al. Evo-harness: Context-to-harness skill compilation for self-evolving agents. *arXiv preprint arXiv:2608.15071*, 2026b.

Tianshi Xu, Huifeng Wen, and Meng Li. Adapting the interface, not the model: Runtime harness adaptation for deterministic llm agents. *arXiv preprint arXiv:2605.22166*, 2026.

Zhuoyun Yu, Xin Xie, Wuguannan Yao, Chenxi Wang, Lei Liang, Xiang Qi, and Shumin Deng. Skilladaptor: Self-adapting skills for llm agents from trajectories. *arXiv preprint arXiv:2606.01311*, 2026.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: open-ended evolution of self-improving agents. In *International Conference on Learning Representations*, volume 2026, pp. 104223–104294, 2026a.

Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents. *arXiv preprint arXiv:2603.19461*, 2026b.

Jike Zhong, Ming Li, Yuxiang Lai, Ziyan Yang, Jingyu Xie, Jihyung Kil, Zheda Mai, Shao-Yuan Lo, Ren Xiang, Konstantinos Psounis, et al. Agentic context learning with self-discovered specification. *arXiv preprint arXiv:2607.09794*, 2026.

Chenyu Zhou, Huacan Chai, Wenteng Chen, Zihan Guo, Rong Shan, Yuanyi Song, Tianyi Xu, Yingxuan Yang, Aofan Yu, Weiming Zhang, et al. Externalization in llm agents: A unified review of memory, skills, protocols and harness engineering. *arXiv preprint arXiv:2604.08224*, 2026.

A Additional Discovery Details

Table 6 gives the full pseudocode for the falsification-driven discovery loop of Section 3.

Discovery configuration. The discovery loop separates two model roles: GPT-5.5 (medium reasoning) serves as the proposer P_ϕ, generating hypotheses, edits, and predictions from the failure report and search history, while GPT-5.1 (medium reasoning) serves as the backbone π, executing each candidate genome A_g^π under the CL-bench evaluation setting. Search uses 400 development examples and a context-disjoint validation set of 400, both synthesized by GPT-5.5 using the category-specific prompts in Appendix F.3. Each genome is evaluated over $K{=}3$ decoding seeds at temperature 0.7, starting from Direct for up to 12 iterations.

The structural genome is

$$g = (\texttt{n_views}, \texttt{use_postprocess}, \texttt{postprocess_target}, \texttt{enforcement}, \texttt{m}),$$

where $\texttt{n_views} \in [1, 4]$ controls the number of requirement-reading views; `use_postprocess` enables an intermediate transformation of their union, with $\texttt{postprocess_target} \in \{\texttt{null}, 9, 15\}$ as a soft granularity hint; `enforcement` selects direct output, additive patching, or best-of-m selection; and $m \in [1, 6]$ when best-of-m is active.

The terminal genome uses four views, post-processing with target 9, additive patching, and $m = 1$. Its post-processing stage consolidates the unioned requirements into a compact set, corresponding to `+consolidate` in Figure 7. Before consolidation, dedup removes only exact duplicates after text normalization, leaving semantic redundancy for consolidation. We retain both the raw union and consolidated set for subsequent analysis.

Figure 7 shows the full search tree, including four rejected branches from the three-view+patch configuration and the temporary strict accuracy regression introduced by the source-specification view before coverage-oriented answering and consolidation recover performance.

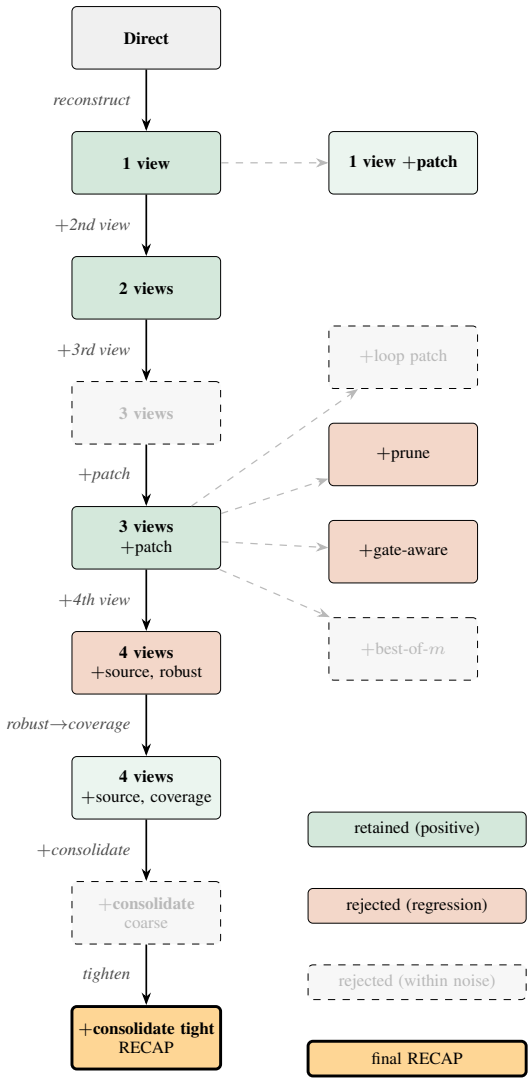

Figure 7: Evolution of the CL-bench pipeline as a falsification-driven search tree.

Table 6: Falsification-driven pipeline discovery. An edit is accepted only when it yields a micro-adherence gain that exceeds a variability-based threshold scaled by per-seed variance and leaves strict accuracy free of a comparable regression, and is adopted as the new best only if its gain also transfers to a context-disjoint validation set (the held-out check acts as an adoption gate).

Input: initial genome g_0 (Direct, no reconstruction), development set $\mathcal{D}_{\text{dev}}$, validation set $\mathcal{D}_{\text{val}}$, seed set $\mathcal{S}$, proposer P_ϕ, threshold factor k, floor $\sigma_{\min}$.
Initialize: constraint memory $\mathcal{M} \leftarrow \emptyset$, history $\mathcal{H} \leftarrow \emptyset$, current genome $g \leftarrow g_0$; evaluate g on $\mathcal{D}_{\text{dev}}$ over $\mathcal{S}$.
for $t = 0, 1, \ldots, T-1$ **do**
  Build a failure report from g's failed tasks, missed rubrics, and recent history $\mathcal{H}$.
  Query proposer: $(h_t, e_t, p_t) \leftarrow P_\phi(\text{report}(g), \mathcal{H}, \mathcal{M})$, where h_t is a causal hypothesis, e_t is one edit (one structural gene, one stage prompt, or the lens list), and p_t predicts the signed effect on micro/strict.
  Apply and validate: $g' \leftarrow e_t(g)$; reject if $g' \notin \mathcal{G}$, if g' equals g, or if the smoke test (non-empty answer from a non-empty enumeration on a few dev items) fails.
  Evaluate g' on the same $\mathcal{D}_{\text{dev}}$ and seeds $\mathcal{S}$; obtain means and per-seed deviations for both metrics.
  Set thresholds $\sigma_\bullet = \max\big(\sqrt{\widehat{\sigma}_\bullet(g')^2 + \widehat{\sigma}_\bullet(g)^2},\ \sigma_{\min}\big)$ for $\bullet \in \{\text{micro}, \text{strict}\}$.
  if $\Delta_{\text{micro}} > k\,\sigma_{\text{micro}}$ **and** $\Delta_{\text{strict}} \geq -k\,\sigma_{\text{strict}}$ **then**
    Provisionally accept; evaluate g' once on $\mathcal{D}_{\text{val}}$.
    if the development gain transfers to $\mathcal{D}_{\text{val}}$ **then** adopt $g \leftarrow g'$; verdict ACCEPTED-IMPROVEMENT.
    else reject (keep g); verdict NON-TRANSFER $\rightarrow$ CONTEXT-SPECIFIC constraint.
  else if $\Delta_{\text{micro}} < -k\,\sigma_{\text{micro}}$ **or** $\Delta_{\text{strict}} < -k\,\sigma_{\text{strict}}$ **then**
    Reject; verdict FLAGGED-REGRESSION $\rightarrow$ DO-NOT constraint.
  else
    Reject; verdict NOISE $\rightarrow$ INEFFECTIVE constraint.
  end if
  if the verdict is a failure (FLAGGED-REGRESSION, NOISE, or NON-TRANSFER) **then** update $\mathcal{M}$ with a natural-language summary of the edit, prediction, measured outcome, and lesson; accepted improvements are not written to $\mathcal{M}$. Always append $(h_t, e_t, \text{verdict})$ to $\mathcal{H}$.
end for
Return: best accepted genome $g^\star$ and constraint memory $\mathcal{M}$.

Configurations behind Table 4. Each row corresponds to one genome re-evaluated on the full benchmark with GPT-5.1. Panel (a) traces the retained path: Direct; one-view requirement reconstruction; additional complementary views; guarded additive patching; a fourth source-specification view with a robust answerer; coverage-oriented answering; and coarse or tight consolidation (`postprocess_target`$=15$ or 9). The robust answerer may override reconstructed requirements it judges incorrect, whereas the coverage-oriented answerer attempts to address every reconstructed requirement. The tight consolidation setting is RECAP. Its target is a soft granularity hint rather than a hard size cap: the prompt targets rubric-level granularity ($\sim$8–20 conditions; Figure 17), yielding 14–24 conditions in practice (Figure 4).

Panel (b) contains rejected variants and budget-matched controls. Repeated patching ($\times 3$) applies the additive patch three times; remove simple requirements prunes low-risk requirements; gate-aware rewriting permits rewriting and deletion rather than additive-only edits; and best-of-m selection samples m answers and selects the one with the highest requirement coverage. The final two controls match RECAP's roughly 7–9 backbone calls per task without reconstructing an explicit requirement set: Self-refine ($\times 3$) uses 7 calls (one draft followed by three critique–revision rounds), while Best-of-8 direct uses 9 (eight independent answers plus one selection turn). Both preserve the original Direct context.

Relation to specification-based pipelines (PSCI). PSCI and RECAP share an extract–condition–repair skeleton, but differ in how the intermediate representation is constructed. PSCI extracts a task-specific specification in a single pass, whereas RECAP separates requirement recovery from compression: complementary views first *recover* a multi-view requirement set, which is then *consolidated* into a compact representation for execution. This factorization makes the recall–executability trade-off explicit and treats compression as a first-class design choice. The guarded additive patch is thus a downstream safeguard. More broadly, our framework suggests a general recipe for context learning: use feedback-driven search over requirement-representation designs to *discover* an effec-

Table 7: Matched comparison of requirement-representation designs on CL-bench with GPT-5.1 (medium effort). Toggling multi-view enumeration and consolidation across the PSCI and RECAP skeletons isolates the effect of the proposed factorization. Full RECAP achieves the highest rubric adherence (83.31) with comparable strict accuracy to PSCI.

Configuration	multi-view	consolidate	Strict	Rubric
Direct	–	–	22.18	82.05
PSCI skeleton (single-pass spec → condition → repair)				
PSCI	○	○	**28.14**	81.13
PSCI + multi-view	●	○	27.59	80.42
PSCI + consolidation	○	●	27.90	82.57
RECAP skeleton (over-recover → consolidate → execute)				
RECAP − consolidation	●	○	27.46	80.97
RECAP − multi-view	○	●	25.60	82.44
RECAP (full)	●	●	27.78	**83.31**

tive representation, use RECAP to explicitly *construct* it at inference time, and use RECAP-RL to *learn to execute* self-constructed representations.

Matched comparison isolating the factorization. PSCI and RECAP have comparable strict accuracy on GPT-5.1 (28.14 vs. 27.78; Table 1), while RECAP achieves higher rubric adherence (83.31 vs. 81.13). To isolate whether this gain comes from the proposed factorization rather than requirement prompting alone, we transplant multi-view enumeration and consolidation individually onto PSCI and remove each from RECAP, holding the backbone and evaluation fixed (Table 7). Neither component alone reproduces the full gain: adding multi-view enumeration or consolidation to PSCI yields 80.42 and 82.57 rubric adherence, respectively, while removing consolidation or multi-view enumeration from RECAP yields 80.97 and 82.44. Multi-view enumeration without consolidation even reduces PSCI from 81.13 to 80.42, while removing multi-view enumeration from RECAP lowers strict accuracy to 25.60. Only the full over-recover-then-consolidate pipeline reaches 83.31 rubric adherence while maintaining strict accuracy comparable to PSCI, supporting the two-stage construction of the requirement representation.

B Full Complex Instruction-Following Transfer Results

Table 8 expands Table 3 with all RECAP-RL reward configurations for the open-weight backbones, each evaluated in Direct and RECAP mode. The same reward axes are ablated on CL-bench in Table 5.

C Additional CL-bench Figures

Figures 8 and 9 visualize the CL-bench strict-accuracy results of Table 1 from Section 4.2.

Reducing missed requirements drives strict accuracy. Figure 10 plots unmet scored requirements per task on Claude Opus-4.8; strict-correct answers are exactly the zero-miss cases. Relative to Direct, RECAP shifts mass toward zero misses and thins the six-or-more tail. Because strict accuracy is all-or-nothing, clearing the last one or two misses barely changes pooled micro adherence but can flip the task outcome, yielding a clear gain in strict accuracy.

General ability is preserved. Figure 11 compares each backbone before and after RECAP-RL on the held-out general benchmarks discussed in Section 4.2. The five-benchmark average stays within about one point for each backbone.

D RECAP-RL Training Details

We train with GRPO on the verl framework. Rewards come from the Qwen3.8-27B judge scoring rubric adherence, and we train for 69 steps. The entropy advantage shaping of Section 3 adds a

Table 8: **Full** transfer results for AgentIF and MultiChallenge, expanding Table 3. For open-weight backbones, rows include all RECAP-RL reward configurations in Direct and RECAP mode. All values are percentages; subscripts are gains over same-backbone Direct.

		Overall	AgentIF		MultiChallenge				
Backbone	**Method**	**Mean**	**CSR**	**ISR**	**Avg**	IR	IM	RVE	SC
Claude Opus-4.8	Direct	43.2	61.7	29.0	57.4	56.5	62.8	56.1	54.0
	PSCI	52.5$_{+9.3}$	67.1$_{+5.4}$	39.9$_{+10.9}$	65.1$_{+7.7}$	65.2	70.8	58.5	**66.0**
	RECAP	**54.7**$_{+11.5}$	**67.9**$_{+6.2}$	**40.9**$_{+11.9}$	**68.4**$_{+11.0}$	**72.5**	**74.3**	**61.0**	**66.0**
GPT-5.1	Direct	47.7	64.0	38.9	56.5	66.7	71.7	41.5	46.0
	PSCI	50.6$_{+2.9}$	66.0$_{+2.0}$	39.0$_{+0.1}$	62.2$_{+5.7}$	69.6	69.9	**51.2**	58.0
	RECAP	**53.0**$_{+5.3}$	**71.0**$_{+7.0}$	**41.3**$_{+2.4}$	**64.7**$_{+8.2}$	**72.5**	**75.2**	**51.2**	**60.0**
GPT-4.1	Direct	34.2	**64.1**	32.2	36.2	53.6	40.7	24.4	26.0
	PSCI	41.7$_{+7.5}$	61.9$_{-2.2}$	33.9$_{+1.7}$	**49.5**$_{+13.3}$	**60.9**	48.7	**46.3**	**42.0**
	RECAP	**43.4**$_{+9.2}$	63.7$_{-0.4}$	**38.6**$_{+6.4}$	48.2$_{+12.0}$	56.5	**50.4**	43.9	**42.0**
Llama-3.2-3B-Instruct	Direct	15.7	48.3	**14.3**	17.2	20.3	12.4	22.0	14.0
	PSCI	19.8$_{+4.1}$	46.4$_{-1.9}$	13.7$_{-0.6}$	25.8$_{+8.6}$	**36.2**	22.1	**26.8**	18.0
	RECAP	15.9$_{+0.2}$	46.1$_{-2.2}$	10.5$_{-3.8}$	21.3$_{+4.1}$	18.8	24.8	19.5	22.0
	RECAP-RL (GPT-5.5, Rubric-mean) + Direct	15.2$_{-0.5}$	47.8$_{-0.5}$	13.7$_{-0.6}$	16.8$_{-0.4}$	17.4	8.8	**26.8**	14.0
	RECAP-RL (GPT-5.5, Rubric-mean) + RECAP	18.9$_{+3.2}$	46.0$_{-2.3}$	10.8$_{-3.5}$	27.0$_{+9.8}$	31.9	25.7	24.4	26.0
	RECAP-RL (GPT-5.5, All-or-nothing) + Direct	14.1$_{-1.6}$	48.0$_{-0.3}$	13.9$_{-0.4}$	14.4$_{-2.8}$	21.7	10.6	17.1	8.0
	RECAP-RL (GPT-5.5, All-or-nothing) + RECAP	18.9$_{+3.2}$	47.6$_{-0.7}$	14.0$_{-0.3}$	23.8$_{+6.6}$	26.1	19.5	19.5	30.0
	RECAP-RL (Self, Rubric-mean) + Direct	14.4$_{-1.3}$	**48.7**$_{+0.4}$	13.4$_{-0.9}$	15.3$_{-1.9}$	15.9	14.2	17.1	14.0
	RECAP-RL (Self, Rubric-mean) + RECAP	**20.9**$_{+5.2}$	48.3$_{+0.0}$	13.6$_{-0.7}$	**28.2**$_{+11.0}$	30.4	**28.3**	22.0	**32.0**
Qwen3.5-4B	Direct	35.3	58.4	24.9	45.7	63.8	54.9	34.1	30.0
	PSCI	37.2$_{+1.9}$	54.7$_{-3.7}$	23.9$_{-1.0}$	50.5$_{+4.8}$	59.4	45.1	41.5	56.0
	RECAP	39.2$_{+3.9}$	56.9$_{-1.5}$	26.5$_{+1.6}$	51.8$_{+6.1}$	**68.1**	48.7	36.6	54.0
	RECAP-RL (GPT-5.5, Rubric-mean) + Direct	35.3$_{-0.0}$	58.9$_{+0.5}$	26.4$_{+1.5}$	44.1$_{-1.6}$	58.0	54.9	19.5	44.0
	RECAP-RL (GPT-5.5, Rubric-mean) + RECAP	40.7$_{+5.4}$	57.4$_{-1.0}$	27.5$_{+2.6}$	53.8$_{+8.1}$	62.3	58.4	36.6	58.0
	RECAP-RL (GPT-5.5, All-or-nothing) + Direct	36.8$_{+1.5}$	59.1$_{+0.7}$	26.3$_{+1.4}$	47.3$_{+1.6}$	62.3	52.2	36.6	38.0
	RECAP-RL (GPT-5.5, All-or-nothing) + RECAP	**44.8**$_{+9.5}$	58.3$_{-0.1}$	**29.8**$_{+4.9}$	**59.7**$_{+14.0}$	**68.1**	55.8	**48.8**	**66.0**
	RECAP-RL (Self, Rubric-mean) + Direct	37.8$_{+2.5}$	**59.8**$_{+1.4}$	27.1$_{+2.2}$	48.5$_{+2.8}$	62.3	57.5	34.1	40.0
	RECAP-RL (Self, Rubric-mean) + RECAP	42.2$_{+6.9}$	56.8$_{-1.6}$	28.0$_{+3.1}$	56.4$_{+10.7}$	63.8	**63.7**	34.1	64.0
Qwen3.5-9B	Direct	42.6	60.6	29.6	55.6	71.0	66.4	39.0	46.0
	PSCI	42.7$_{+0.1}$	58.3$_{-2.3}$	27.6$_{-2.0}$	57.7$_{+2.1}$	60.9	59.3	48.8	**62.0**
	RECAP	47.7$_{+5.1}$	62.2$_{+1.6}$	32.7$_{+3.1}$	**62.7**$_{+7.1}$	**72.5**	66.4	**56.1**	56.0
	RECAP-RL (GPT-5.5, Rubric-mean) + Direct	42.1$_{-0.5}$	60.1$_{-0.5}$	30.6$_{+1.0}$	53.7$_{-1.9}$	63.8	70.8	34.1	46.0
	RECAP-RL (GPT-5.5, Rubric-mean) + RECAP	42.2$_{-0.4}$	60.8$_{+0.2}$	31.5$_{+1.9}$	52.8$_{-2.8}$	63.8	64.6	26.8	56.0
	RECAP-RL (GPT-5.5, All-or-nothing) + Direct	42.0$_{-0.6}$	61.3$_{+0.7}$	31.8$_{+2.2}$	52.2$_{-3.4}$	66.7	68.1	34.1	40.0
	RECAP-RL (GPT-5.5, All-or-nothing) + RECAP	**48.3**$_{+5.7}$	**63.1**$_{+2.5}$	**35.5**$_{+5.9}$	61.2$_{+5.6}$	66.7	67.3	48.8	**62.0**
	RECAP-RL (Self, Rubric-mean) + Direct	44.6$_{+2.0}$	60.7$_{+0.1}$	29.8$_{+0.2}$	59.3$_{+3.7}$	66.7	**76.1**	46.3	48.0
	RECAP-RL (Self, Rubric-mean) + RECAP	45.1$_{+2.5}$	61.4$_{+0.8}$	31.8$_{+2.2}$	58.4$_{+2.8}$	65.2	68.1	46.3	54.0

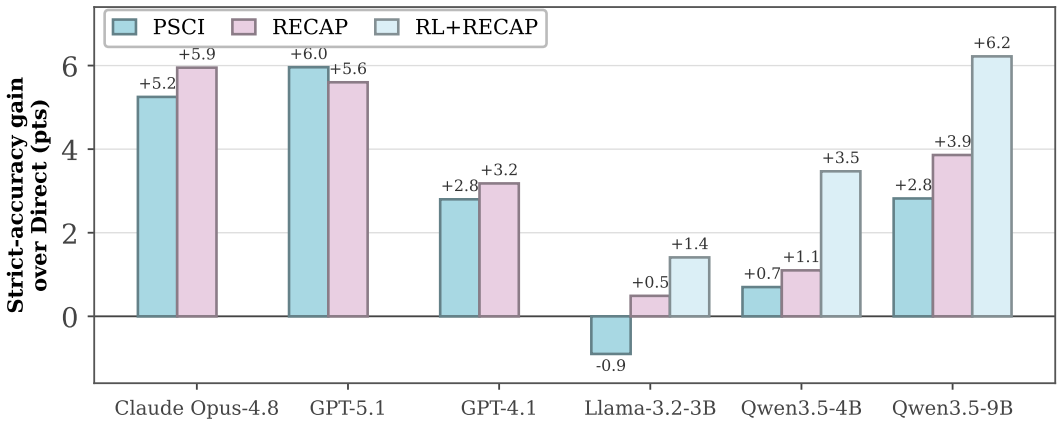

Figure 8: CL-bench strict-accuracy gain over same-backbone Direct.

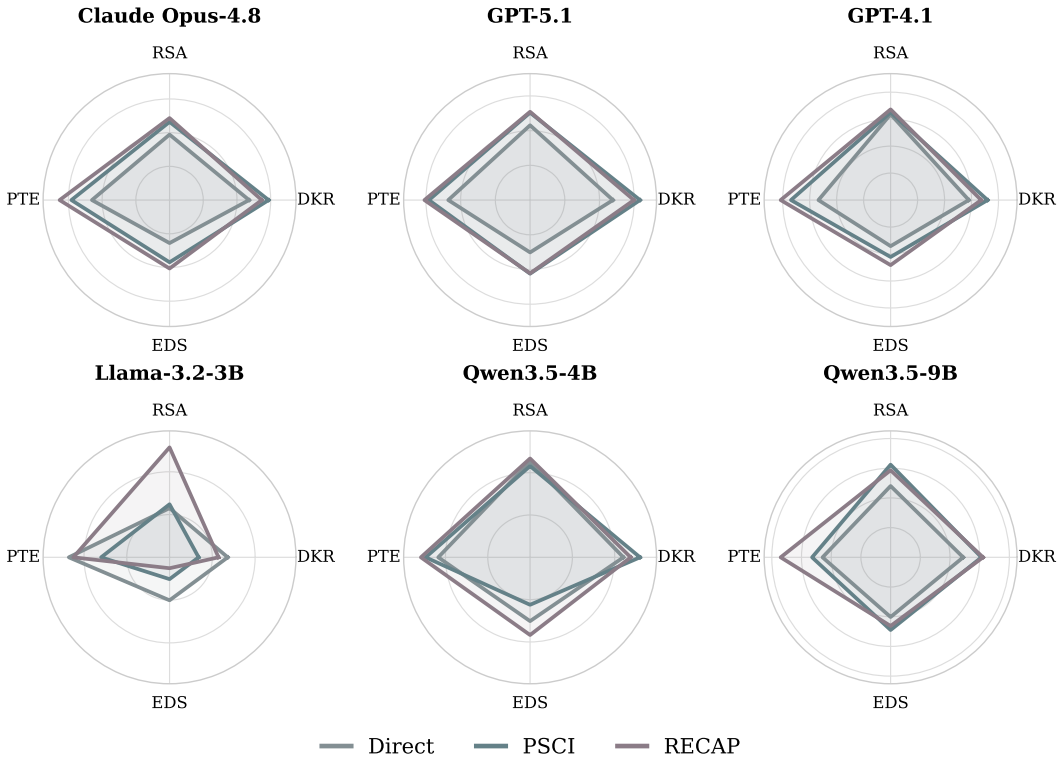

Figure 9: Per-family CL-bench strict accuracy, with one panel per backbone.

clipped, gradient-detached entropy bonus to the group-relative advantage,

$$\tilde{A}_{i,t} = A_i + \min\left(\alpha\, H_{i,t}^{\text{detach}},\ \frac{|A_i|}{\kappa}\right),$$

where $H_{i,t}$ is the policy entropy at token t; we fix $\kappa = 2$ and, for GRPO, set $\alpha = 0.4$. Table 9 summarizes the full set of training hyperparameters.

Training-corpus construction and source. RECAP-RL is trained on 9,410 context–task instances—9,410 tasks over 3,737 self-contained contexts—carrying 129,782 atomic binary rubrics in total (13.8 per task on average, range 9–24), with a further 192-instance validation split. The contexts, tasks, and rubrics are synthesized by GPT-5.1 (medium reasoning effort) across the four

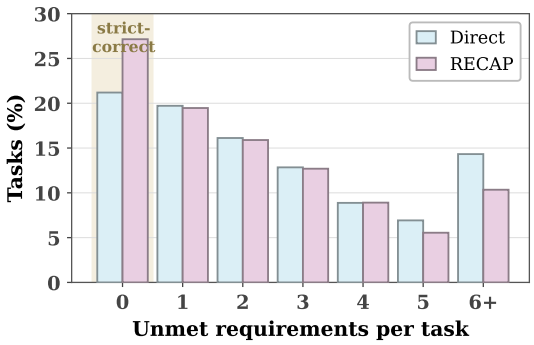

Figure 10: Unmet scored requirements per task, Direct vs. RECAP.

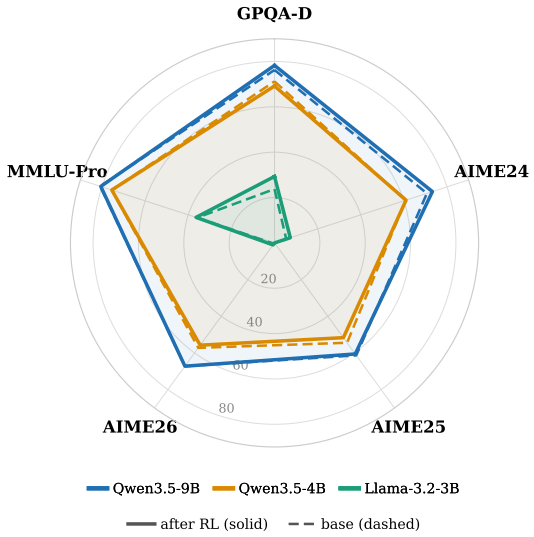

Figure 11: Base (*dashed*) vs. RECAP-RL (*solid*) on held-out general benchmarks (GPQA-Diamond, AIME 2024/2025/2026, and MMLU-Pro).

CL-bench categories (Jin et al., 2026b); no CL-bench context, question, rubric, or reference answer is reused—verbatim or paraphrased—at any stage of synthesis, training, or reward.

Causal controls for representation-conditioned RL. RECAP-RL conditions training on the self-constructed representation $\hat{R}$ and rewards rubric adherence, but its gains could reflect general rubric-following ability rather than learning to execute $\hat{R}$. To isolate the effect of representation-conditioned training, we hold the synthetic data, judge, reward, and GRPO hyperparameters fixed and train a *No-representation RL* control on (c, x) without $\hat{R}$ (Table 10). We evaluate both checkpoints in Direct mode and with full RECAP inference. Without $\hat{R}$ during training, performance drops by 1.27 strict accuracy and 1.80 rubric under RECAP inference, and also falls in Direct mode (13.01 vs. 14.38 strict accuracy). The latter is particularly informative: because both checkpoints receive identical inputs at inference, the gap cannot be explained by the inference-time representation. Thus, conditioning on $\hat{R}$ during RL changes the learned policy itself, supporting the conclusion that RECAP-RL learns to execute its reconstructed requirements rather than merely acquiring a representation-independent rubric-following ability.

Reward ablation on Llama-3.2-3B-Instruct. Table 11 reports the reward ablation of Table 5 for the Llama-3.2-3B-Instruct backbone. The trends match the Qwen backbones: *Self*-reconstructed requirements with a *Rubric-mean* reward give the strongest checkpoint, while *All-or-nothing* is less

Table 9: GRPO training hyperparameters.

Hyperparameter	Value
Training Framework	verl Sheng et al. (2024)
Advantage Estimator	GRPO
Train Batch Size	144
PPO Mini Batch Size	144
Micro Batch Size Per Device (Update)	1
Micro Batch Size Per Device (Experience)	1
Rollout n	5
Rollout Temperature	1.0
Use KL Loss	True
KL Loss Estimator	Low-variance
KL Coefficient	1×10^{-2}
Actor Learning Rate	1×10^{-6}
Weight Decay	1×10^{-2}
Gradient Clipping	1.0
Max Prompt Length	25,000
Max Response Length	20,000
Training Steps	69
Entropy Advantage Shaping κ	2
Entropy Advantage Shaping α	0.4
Rollout Engine	vLLM
vLLM Tensor Parallel Size	4
Number of GPUs	24 (3 nodes $\times$ 8 H200)

Table 10: Causal controls for representation-conditioned RL on Qwen3.5-9B. All configurations share the same experimental setup, varying only in whether $\hat{R}$ is provided during training. Each checkpoint is evaluated with Direct and full RECAP inference. Subscripts denote absolute changes from Base.

	Direct inference		RECAP inference	
Training configuration	Strict	Rubric	Strict	Rubric
Base (no RL)	11.74	71.91	15.60	75.53
No-representation RL	$13.01_{+1.27}$	$74.30_{+2.39}$	$16.69_{+4.95}$	$78.00_{+6.09}$
RECAP-RL (ours)	$14.38_{+2.64}$	$76.47_{+4.56}$	$17.96_{+6.22}$	$79.80_{+7.89}$

stable and degrades micro adherence under RECAP. Figure 12 shows the corresponding Llama-3.2-3B reward curves, in the same broken y-axis format as the Qwen backbones in Figure 3.

E Qualitative Case Studies

This appendix walks through CL-bench tasks where the same backbone flips from a zero to a pass. All of them illustrate the failure mode that motivates RECAP: CL-bench is scored all-or-nothing, so a single missed obligation—often a low-salience one buried among many satisfied requirements—zeroes an otherwise strong answer. We show (i) inference-time gains from the RECAP pipeline over Direct on *closed-source, strong* backbones (GPT-5.1 and Opus-4.8), and (ii) training-time gains from RECAP-RL—the headline checkpoint, trained on *Self*-constructed requirements with the *Rubric-mean* reward (the "self-rubric" setting of Table 5)—over the untrained backbone on an *open-source, weak* model (Qwen3.5-9B). In each subsection the first case is a full walkthrough and the remaining ones are compact flips. Every excerpt is verbatim from the corresponding model's run; "`[...]`" marks elisions we made for space.

E.1 Direct vs. RECAP on closed-source, strong backbones

A role obligation Direct never claims (GPT-5.1). Task (*Rule System Application / Game Mechanics*). A powerlifting-meet multi-agent system casts the model as the *Middle Referee Agent*—"the central judging authority" that issues the start signal (`Squat`) and final command (`Rack`), then compiles the three referees' Pass/Fail votes into the official ruling. The transcript deliberately buries the seat under floor chatter, timing pressure, and side-referee opinions. The task carries 13 rubric

Table 11: RECAP-RL reward ablation on CL-bench for Llama-3.2-3B-Instruct, crossing requirement source (*GPT-5.5* vs. *Self*) with reward scheme (*Rubric-mean* vs. *All-or-nothing*). Metrics and families match Table 1; the Qwen backbones are in Table 5. Best within the block is **bold**; the shaded row is the headline checkpoint.

				Overall		DKR		RSA		PTE		EDS	
Backbone	**Req. source**	**Reward**	**Mode**	Strict	Rubric	Strict	Rubric	Strict	Rubric	Strict	Rubric	Strict	Rubric
Llama-3.2-3B-Instruct	GPT-5.5	Rubric-mean	Direct	2.95	55.77	2.41	59.29	2.65	51.47	4.88	60.20	1.01	39.10
			RECAP	3.69	56.20	**4.23**	59.04	2.65	51.78	5.10	61.68	1.51	39.24
	GPT-5.5	All-or-nothing	Direct	2.63	53.67	2.56	55.47	1.59	49.90	4.67	58.61	1.01	40.62
			RECAP	3.79	52.24	3.02	55.69	4.06	46.95	5.94	57.48	0.50	36.03
	Self	Rubric-mean	Direct	3.53	57.15	2.56	59.91	3.00	53.04	6.58	62.01	1.01	41.66
			RECAP	**4.42**	**58.90**	3.17	**62.05**	**4.77**	**54.09**	**6.79**	**64.16**	**2.01**	**42.48**

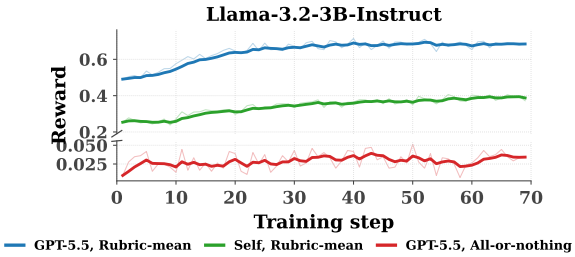

Figure 12: RECAP-RL reward dynamics for Llama-3.2-3B-Instruct. Broken y-axis: rubric-mean rewards above, all-or-nothing below. Faint lines show per-step means; solid lines show an EMA.

items— start/final commands, majority-vote arithmetic, exactly-two-outputs, no early cues—and Direct gets 12 of them right. The decisive one is the very first:

> **R1.** *Does the model accept and adopt the role of the central judging authority? Fail if the model refuses the role or acts as a subordinate/side referee.*

Direct issues the correct commands, tallies the votes correctly, and reaches the right "No Lift" outcome—but it reports the center seat as one detached third vote ("`Center Referee FAIL`") and never actually *occupies* the role it was assigned. Under strict grading this single unclaimed obligation fails R1 and zeroes an otherwise flawless answer. RECAP's enumeration surfaces role adoption as its own checkable obligation, so the coverage draft speaks in the first person as the presiding judge ("`Center Referee (me)`", "`Personal judgment`"), satisfying R1 while keeping every other item intact.

Direct output (scored 0 — center seat reported, never adopted; R1 fails):

```
1) Individual Referee Decision
- Start command issued: "Squat"
- Final command issued: "Rack"
- Decision: FAIL -- depth standard not met; the hip crease did not descend to or below the
top of the knee.

2) Final Consensus Signal
- Votes: Left Referee FAIL, Right Referee PASS, Center Referee FAIL
- Overall outcome: No Lift
- Numerical tally: 2-1 (two Fail, one Pass)
```

RECAP output (scored 1 — first-person central authority; R1 passes):

```
Individual Referee Decision
- Start command issued: Squat
- Final command issued: Rack
- Personal judgment: Fail
- Rationale: On this squat the hip crease remained slightly above the top of the knee at
the deepest point (about 1-2 cm high) [...]
```

```
Final Consensus Signal
- Votes: Left Referee = Fail, Right Referee = Pass, Center Referee (me) = Fail
- Overall outcome (majority decision): No Lift
- Numerical tally: 1-2 (Pass-Fail)
```

A formatting-only obligation (Opus-4.8). **Task** (*Rule System Application / Workflow Orchestration*). A Risk Indicator Extraction Agent must turn a sanitized military-counseling transcript into a structured risk-indicator vector. Among ∼30 rubric items—correct lexicon version, evidence excerpts, masking preserved, escalation not toggled off under social pressure—the backbone is right on every substantive one. The decisive item is purely about output shape:

> **R25.** *Did the model output only the JSON vector without conversational filler? Fail if output includes "Here is the analysis:" or surrounding prose.*

Direct wraps its (correct) JSON in a conversational preamble and a bulleted "adherence notes" summary, which fails R25 and zeroes the task. RECAP emits the same vector but begins directly with the JSON object.

Direct output (scored 0 — prose preamble precedes the JSON; R25 fails):

````
I'll process the final sanitized transcript (Artifact 1) using only the approved lexicon (
Artifact 4), disregarding the superseded transcript, unapproved synonyms, and the pressure
to skip excerpts or suppress escalation.

Key adherence notes:
- Escalation set per rule (intent/plan/means present) -- NOT toggled off per social
pressure.
- Evidence excerpts included and masking preserved -- NOT skipped.

```json
{ "riea_output": { "risk_indicator_vector": { "intent_flag": true, [...] } } }
```
````

RECAP output (scored 1 — output is the JSON object, no filler):

````
```json
{
  "risk_indicator_vector": {
    "schema_version": "RIEA-1.0",
    "lexicon_pack": "MIL-SAFE v2.4",
    "intent_flag": true,
    [...]
  }
}
```
````

Brackets vs. parentheses (Opus-4.8). **Task** (*Interpretation of Observational Data*). Given five running backs' game logs, the model must argue who is most likely to win the rushing title. The content analysis is correct in both runs; the decisive rubric governs how a cited statistic must be delimited:

> **R7.** *The response should cite a stat from the text using brackets around the verbatim quote the stat came from.*

Direct quotes the source stat but delimits it with *parentheses*; RECAP uses *square brackets*. This lone punctuation difference is enough to fail R7 and zero an answer that is otherwise identical in substance—a stark illustration of how unforgiving all-or-nothing scoring is.

Direct output (scored 0 — verbatim quote in parentheses; R7 fails):

```
The clear frontrunner here is Jonathan Taylor. He's not only leading the pack in total
yards ("Regular Season Stats 189 1,139 6.0 15" from Taylor's game log), but he's doing it
while having played one FEWER game [...]
```

RECAP output (scored 1 — verbatim quote in square brackets; R7 passes):

```
Taylor is out front with 1,139 rushing yards ["Regular Season Stats 189 1,139 6.0 15" [...]
 from Jonathan Taylor's game log totals], and what makes it even scarier is that he's put
up that number in just 10 games [...]
```

E.2 Untrained vs. RECAP-RL on an open-source, weak backbone

A delivery obligation the backbone omits. **Task** (*Procedural Task Execution / Operational Procedures*). An "Image Processing Agent" must return a structured visual-features JSON package *and* route the complete package to three downstream consumers—the Style, Aesthetic, and Composition Analysis agents—at once. The decisive rubric items are:

> **R5.** *Is the complete data package sent simultaneously to the Style, Aesthetic, and Composition Analysis Agents? Fail if sent sequentially or if any of the three is excluded.*
> **R6.** *Is the complete Structured Visual Features Data Package sent to the three downstream agents?*

The untrained backbone produces a valid, richly detailed JSON package and even handles the watermark/obstruction edge cases correctly—but it never addresses the three downstream agents, so R5 and R6 fail and the task scores 0. After RECAP-RL with the self-rubric reward, the model emits an explicit `downstream_agent_guidance` block that names all three consumers within a single shared package, satisfying both obligations.

Untrained backbone output (scored 0 — no downstream delivery):

```
{
  "processing_report":       { "images_analyzed": 4, "degraded_count": 1, [...] },
  "degraded_images":         [ { "image_id": "Lot_34", [...] } ],
  "processable_images":      [ { "image_id": "Lot_17", [...] } ],
  "complete_feature_packages": [ { "image_id": "Lot_17", "technical_properties": { [...] }
} ]
}
// package built, but never addressed to Style / Aesthetic / Composition -> R5, R6 fail
```

RECAP-RL (self-rubric) output (scored 1 — all three agents addressed):

```
"downstream_agent_guidance": {
    "style_classification": { "action": "...", "package": "<full features>" },
    "aesthetic_analysis":   { "action": "...", "package": "<full features>" },
    "composition_analysis": { "action": "...", "package": "<full features>" }
}
// one shared package addressed to all three consumers simultaneously -> R5, R6 pass
```

An unset groove parameter. **Task** (*Rule System Application / Technical Standards*). A "Pocket Drummer" agent renders a groove under numeric production constraints, one of which fixes the hi-hat swing:

> **R10.** *Is hi-hat swing set to 54–58% (triplet feel)? Fail if global swing is outside 54–58%.*

The untrained backbone lays out a three-layer hi-hat stack but never states a swing value at all, so R10 cannot be satisfied and the task scores 0. After RECAP-RL the model pins the swing to `56%`—inside the required band.

Untrained backbone output (scored 0 — no swing value stated; R10 fails):

```
Hi-Hats: 3-Layer Stack (C1-16 / Closed-16 / Ghost-32nd).
```

RECAP-RL (self-rubric) output (scored 1 — swing pinned to 56%):

```
Hi-Hats: Layer 1 (909 Closed 16ths @ 56% swing), Layer 2 (808 Pedal Ghosts),
         Layer 3 (Micro-open accents on off-beats). Triplet-feel chatter established.
```

A missed threat category. Task (*Rule System Application / Workflow Orchestration*). This is the open-model counterpart of the risk-vector task above: the agent extracts a risk-indicator vector from a military-counseling transcript, and must enumerate every *means type* the speaker can access. The transcript names both a firearm ("the M9 from the duty safe") and a vehicle ("I signed for the Humvee keys tonight"). The decisive rubric is:

> **R23.** *Did the model correctly identify "vehicle" as a means type? Fail if vehicle threats are ignored.*

The untrained backbone records `means_type` as the bare string `"firearm"`—the Humvee sits right there in its own evidence list but is never classified as a means—so R23 fails. After RECAP-RL the model returns `means_type` as a list that includes `"vehicle"` and tags the Humvee excerpt accordingly.

Untrained backbone output (scored 0 — Humvee cited but not classified; R23 fails):

```
{
  "risk_indicator_vector": {
    "means_flag": true,
    "means_type": "firearm",
    "evidence": [
      "If I can get the M9 from the duty safe after lights out, I could go out to the motor
 pool.",
      "I signed for the Humvee keys tonight. Before patrol I can take it out alone.",
      [...]
    ]
  }
}
```

RECAP-RL (self-rubric) output (scored 1 — vehicle captured as a means type):

```
{
  "risk_indicator_vector": {
    "means_flag": true,
    "means_type": ["firearm", "vehicle"],
    "evidence": [
      { "excerpt": "I signed for the Humvee keys tonight. Before patrol I can take it out
alone.",
        "matched_lexicon": ["take the [vehicle]", "tonight", "before patrol", "alone"] }
    ]
  }
}
```

F PROMPTS

This appendix reproduces the prompts used in the paper, as captioned figures. We give them in four parts: the proposer prompt that drives the falsification-driven *algorithm discovery* loop (Section 3, Appendix A); the per-stage prompts of the discovered RECAP pipeline (the terminal genome `n_views=4`, `use_postprocess=true`, `enforcement=patch`, `m=1`); the prompts that synthesize the RECAP-RL training corpus (Appendix F.3); and the judge grading prompt that produces the RECAP-RL reward (Appendix F.4). The per-call user message is assembled from the source context, the task, and the stage-specific fields; the `Return STRICT JSON only` schema

is shown where it is part of the prompt. Full-width brackets and subscripts in the judge prompt are transcribed as ASCII for typesetting.

F.1 Algorithm Discovery: Proposer Prompt

The proposer P_ϕ reads the current harness genome, the dev grading failure report, the standing constraints, and the search history, and returns one hypothesis, one topology-open edit, and a signed prediction. It is the only model that mutates the pipeline (Figures 13 and 14).

```
You are the RESEARCH PROPOSER in a controlled empirical search over a test-time harness for
 context-dependent task execution.

Your role is NOT to answer benchmark tasks. At each iteration you receive: (1) the current
harness configuration, (2) observations from its development-set evaluation, (3) previously
 tested modifications and their measured outcomes, and (4) standing constraints accumulated
 from prior experiments. Your job is to propose ONE modification that tests a falsifiable
hypothesis about the current pipeline.

METRICS: the evaluation reports two numbers per configuration -- strict accuracy (per task,
 all-or-nothing) and micro accuracy (per requirement item). These define the objective;
they do not tell you which modification will improve them.

SEARCH PRINCIPLES:
  * Base every hypothesis only on the observations in the current evaluation report and
search history.
  * Treat all editable components and parameter values as hypotheses to be tested. Do not
assume that any operation, stage, parameter direction, or pipeline structure is beneficial
unless prior experimental evidence supports it.
  * Do not assume a causal explanation for an observed failure; infer one from the supplied
 evidence and state it explicitly as a hypothesis.
  * Propose exactly ONE attributable modification per iteration.
  * Do not repeat a modification already tested, and do not violate a standing constraint.
  * Prefer modifications that discriminate between competing explanations of the observed
failures.
  * Predict the signed effect on both metrics before seeing the result. A prediction may be
 positive, negative, or neutral -- do not force an improvement prediction.

EDITABLE SEARCH SPACE:
  The current harness is a bounded set of editable structural variables, stage instructions
, and reading lenses, whose operational definitions and allowed values are given in CURRENT
 HARNESS GENOME and CURRENT TOPOLOGY. You may modify exactly one of: one structural
variable; one stage instruction; or the reading-lens configuration. Do not introduce
operations outside the provided search space.

Return STRICT JSON only:
{"hypothesis": "<a falsifiable explanation grounded in the supplied observations>", "
evidence": "<the specific observations motivating the hypothesis>", "edit": {"<one editable
 field>": <new value>}, "prediction": {"micro": "<up|down|flat + brief reason>", "strict":
"<up|down|flat + brief reason>"}, "falsification": "<what observed result would count
against the hypothesis>"}
```

Figure 13: Discovery proposer prompt: the falsification-driven step that proposes one attributable edit within the bounded search space and predicts its effect on both metrics.

```
[CURRENT HARNESS GENOME]
{genome_json}

[CURRENT TOPOLOGY]
{topology_summary}

[DEVELOPMENT EVALUATION OBSERVATIONS]
{failure_report}

[STANDING CONSTRAINTS FROM PRIOR EXPERIMENTS]
{constraints_text}

[PAST MODIFICATIONS AND MEASURED OUTCOMES]
{history}

Using only the information above, identify one testable hypothesis about the current
pipeline and propose one modification that would provide evidence for or against it. Do not
```

```
 assume that any editable component is beneficial a priori; do not infer expected effects
from the names of components. Return STRICT JSON only.
```

Figure 14: Discovery proposer user message template. Braced fields are filled per iteration from the current genome, topology summary, development evaluation observations, standing constraints, and search history.

F.2 RECAP Stage Prompts

RECAP enumerates requirements under four reading lenses and unions them (Figures 15 and 16), consolidates the union to rubric granularity (Figure 17), drafts a coverage answer (Figure 18), audits it (Figure 19), and additively patches the missing pieces (Figure 20).

```
You are a meticulous requirements analyst. You are given the SOURCE CONTEXT of a task and
the TASK itself. Your job is NOT to answer the task. Your job is to enumerate the concrete,
 atomic, individually-checkable conditions that ANY correct response must satisfy -- the
necessary conditions a strict grader would test one by one.

Rules for good conditions:
- Atomic: exactly one testable thing each (split compound conditions).
- Verifiable from the response text alone by a grader who also sees the context.
- Grounded: derived from the instructions, the request, or the context -- never invented.
- Specific: name the exact string / field / section / value where the task fixes one.
- Do NOT include the answer content itself; describe the CONDITION, not the solution.

Return STRICT JSON only:
{"requirements": [{"id": "R1", "text": "<the condition, imperative>", "kind": "format|
content|constraint", "priority": "high|medium|low"}]}
```

Figure 15: RECAP stage 1 (enumerate): the per-view requirements analyst, run once under each reading lens of Figure 16.

```
[content] Read for CONTENT COMPLETENESS. Enumerate every sub-question, field, entity, step,
 computation, or list element that the task requires to be PRESENT in the answer. If the
task enumerates N things, each is its own condition. Capture the required depth / level of
detail where the task specifies it.

[format] Read for OUTPUT SHAPE. Enumerate exact-form obligations: required literal strings,
 labels, headers, section names, ordering/numbering, JSON or table schema, units, length or
 count limits, fixed opening/closing phrases, and citation/formatting conventions the task
mandates.

[constraint] Read for CONSTRAINTS & EXCLUSIONS. Enumerate rules the answer must obey:
persona / role / protocol behavior, refusal or safety phrasing, source-grounding (use only
the provided context, no outside facts), forbidden content or omissions, and conditional
requirements ('if X then Y') including their edge cases.

[source_spec] Read for the SOURCE-STATED OUTPUT SPECIFICATION. Scan the source context and
task for every place where they EXPLICITLY dictate the output: an output schema or template
, a worked example answer, a numbered/bulleted list of required elements, or imperative
phrases such as 'your response must include', 'format as', 'return only', 'do not', 'in the
 following order'. Enumerate each explicitly-mandated element as its OWN atomic condition,
naming the exact source-defined label / field / string / section / value. These are the
highest-value conditions: a strict grader most often tests exactly what the source spells
out. Mark them high priority when the source states them as hard requirements.
```

Figure 16: RECAP's four reading lenses. Each is substituted into the analyst of Figure 15; `source_spec` is the lens added by the terminal genome.

```
You are a grading-rubric editor. Several independent readers each enumerated conditions a
correct response must satisfy, and their outputs were unioned into one large list. That
list is heavily REDUNDANT and OVER-SPECIFIED: the same obligation is restated in several
different wordings, one real obligation is split into many tiny fragments, and some entries
 invent specific values/strings/details the task and source do NOT actually require.
```

```
Rewrite this into the CONSOLIDATED set of conditions a strict grader would actually check
-- the natural granularity of a grading rubric. Rules:
  * MERGE conditions that test the same underlying obligation into ONE clear condition;
collapse fragments of a single requirement back into that requirement.
  * DROP entries that invent a specific value, field, string, or detail that the task/
source does not actually mandate (keep the condition only at the level the task truly fixes
).
  * KEEP every genuinely-necessary, distinct obligation -- content, format, length/count,
exclusion, persona, and conditional rules alike. Do NOT drop a real obligation just because
 it is simple. Preserve any exact string / limit / format the task really requires,
verbatim.
  * Each surviving condition stays atomic and independently checkable from the response
text.
Aim for the granularity of a real rubric (typically ~8-20 conditions); do not pad and do
not over-compress. Preserve each condition's kind and priority.

Return STRICT JSON only:
{"requirements": [{"id": "R1", "text": "<condition, imperative>", "kind": "format|content|
constraint", "priority": "high|medium|low"}]}
```

Figure 17: RECAP stage 1b (consolidate): merge the inflated union down to rubric granularity, with the goal of preserving all distinct obligations.

```
You are an expert responder for context-grounded tasks. Produce ONLY the final response
that the task requires -- no preamble, no meta commentary, no mention of any checklist or
process. Your response must be fully grounded in the provided SOURCE CONTEXT (do not
introduce outside facts).

You are given a REQUIREMENTS CHECKLIST of necessary conditions a strict grader will test
one by one. Before finishing, make sure EVERY condition is VISIBLY and explicitly satisfied
 in your response -- never rely on implication or on the grader inferring it. Honor exact-
format conditions literally (exact strings, ordering, schema). If two conditions appear to
conflict, follow the task's own instructions and the higher-priority condition. Output the
response itself only.
```

Figure 18: RECAP stage 2 (answer): the coverage draft, conditioned on the consolidated requirements checklist.

```
You are a strict coverage auditor. You are given the SOURCE CONTEXT, the TASK, a list of
necessary CONDITIONS, and a candidate RESPONSE. For each condition decide whether the
response VISIBLY satisfies it (a strict grader would mark it pass). Be conservative: if
satisfaction is not clearly visible in the response text, mark it unmet.

Return STRICT JSON only:
{"met": <int>, "total": <int>, "unmet": [{"id": "R#", "text": "<condition>", "priority": "
high|medium|low", "fix": "<the minimal, specific addition/change the response needs to
satisfy it>"}]}
```

Figure 19: RECAP stage 3a (audit): the conservative coverage check that flags unmet conditions and a minimal fix for each.

```
You are a completion editor. You are given a candidate RESPONSE that is mostly correct and
a list of MISSING PIECES (specific conditions it does not yet satisfy, each with a minimal
fix). Produce the final response that ADDS exactly those missing pieces while PRESERVING
all existing correct content -- keep satisfied parts as they are, keep any required fixed
format intact, and do not remove or rephrase things that already work. Integrate the
additions cleanly so the whole reads as one coherent response. Output only the final
response -- no commentary.
```

Figure 20: RECAP stage 3b (patch): additive, non-destructive completion of the missing pieces; kept only if audited coverage does not worsen.

F.3 RECAP-RL TRAINING-DATA SYNTHESIS

The RECAP-RL training corpus (Section 3.3) is synthesized by GPT-5.1 in three stages, one example at a time. Stage G.1 generates a self-contained context under one of the four CL-bench categories, using a category-specific designer prompt (Figures 21, 22, 23, and 24); the procedural category emits its own system instruction inline, while the other three receive a reusable system instruction from stage G.2 (Figure 25). Stage G.3 then writes several context-dependent tasks, each a question x, a golden answer, and a dense list of atomic binary training rubrics R^{tr} (Figure 26). Braced fields are filled per example from category-specific seed variables (subcategory, theme words, tone/audience, target length, task and rubric counts).

```
# Role
You are a benchmark context designer for context-learning. Your job is to create a
realistic, fully invented professional document that teaches novel domain
knowledge inside the context itself.

# Target Category
CL-Bench category: Domain Knowledge Reasoning
Subcategory: {subcategory_name}
{subcategory_description}

# Design Inputs
- Concrete focus: {class_level}
- Theme seeds: {theme_words}
- Inspiration only, do not copy: {variant_example}

# Required Qualities
1. Invented but plausible: all organizations, policies, products, people, places, datasets,
 cases, and events must be fictional.
2. Domain realism: use the vocabulary, document conventions, and evidence style natural to
the chosen subcategory.
3. Reasoning density: include definitions, cases, tables, thresholds, exceptions, tensions,
 and cross-section dependencies.
4. Internal consistency: numeric values, dates, categories, and examples must agree across
the document.
5. Self-containment: all knowledge required for later tasks must appear in the document.
6. Length: target at least {min_chars} characters of substantive body text. Do not pad.

# Output Rules
- Write in English.
- Output only the document body, with no meta-commentary and no markdown code fence.
- Use clear section headings, tables, and examples when useful.
- Do not mention that the document is fictional or benchmark-generated.
Before final output, silently check that the document is complete, internally consistent,
and rich enough to support multi-hop reasoning.
```

Figure 21: Stage G.1 context designer for *Domain Knowledge Reasoning*.

```
# Role
You are a benchmark context designer for context-learning. Your job is to create a
fully original, self-contained rule system that can support difficult multi-step
reasoning tasks.

# Target Category
CL-Bench category: Rule System Application
Subcategory: {subcategory_name}
{subcategory_description}

# Design Inputs
- Concrete focus: {class_level}
- Theme seeds: {theme_words}
- Inspiration only, do not copy: {subcategory_example}

# Required Qualities
1. Originality: invent all entities, rules, symbols, institutions, and examples. Do not
adapt real programming languages, games, laws, or public systems with superficial renaming.
2. Formal precision: define entities, attributes, operations, state transitions, conflict
priorities, exceptions, and termination conditions clearly enough for deterministic task
solving.
3. Reasoning density: include interacting rules, delayed effects, cross-references,
boundary cases, examples, and at least one precedence hierarchy.
```

```
4. Self-containment: a downstream model must be able to answer later questions using only
this document.
5. Length: target at least {min_chars} characters of substantive body text. Do not pad with
 repetition.

# Suggested Structure
1. Overview and scope.
2. Vocabulary, symbols, and entity schema.
3. Core mechanics and state transitions.
4. Conflict resolution and priority order.
5. Worked examples, records, calculations, or pseudo-code.
6. Edge cases and invalid states.

# Output Rules
- Write in English.
- Output only the document body, with no meta-commentary and no markdown code fence.
- Use stable section headings and precise terminology.
- Keep examples internally consistent with the rules.
- Do not reveal that the document is generated for a benchmark.
Before final output, silently check for contradictions, undefined terms, impossible
examples, and low-information filler.
```

Figure 22: Stage G.1 context designer for *Rule System Application*.

```
# Role
You are a benchmark context designer for context-learning. Your job is to create a
fully invented empirical dossier that teaches a new pattern, mechanism, law, or
simulator behavior inside the context itself.

# Target Category
CL-Bench category: Empirical Discovery & Simulation
Subcategory: {subcategory_name}
{subcategory_description}

# Design Inputs
- Concrete focus: {class_level}
- Theme seeds: {theme_words}
- Inspiration only, do not copy: {variant_example}

# Required Qualities
1. Invented empirical world: all variables, entities, instruments, materials, locations,
and labels must be fictional or generic. Do not rely on real-world scientific laws as the
answer.
2. Discoverable structure: include enough observations for a downstream model to infer a
rule, trend, threshold, causal relationship, transition dynamic, or validity boundary from
the context.
3. Data richness: include tables, logs, repeated trials, measurements, ablations,
interventions, or simulation traces.
4. Noise and traps: include irrelevant variables, noisy measurements, edge cases,
conflicting preliminary notes, or regime shifts, while keeping the true pattern internally
consistent.
5. Self-containment: all definitions, units, measurement conventions, and evidence needed
for later tasks must appear in the document.
6. Length: target at least {min_chars} characters of substantive body text. Do not pad.

# Output Rules
- Write in English.
- Output only the document body, with no meta-commentary and no markdown code fence around
the whole document.
- Do not include questions, answers, rubrics, benchmark references, or AI-system references
.
- Do not reveal the hidden rule as a final summary; let it be inferable from the evidence.
```

Figure 23: Stage G.1 context designer for *Empirical Discovery & Simulation*.

```
[Role]
You are a benchmark context designer for context-learning. Create a strict
procedure-execution scenario where a model must follow a newly provided manual,
workflow, or operational protocol instead of relying on generic commonsense.
```

```
[Target Domain]
- Domain: {domain}
- Sub-domain: {sub_domain}

[Output Shape]
Return only one valid JSON object with this shape:
{"system_instruction": "...", "context": "..."}

[System Instruction Requirements]
The system_instruction field must define the procedure-following role, allowed evidence,
and output contract. Include:
- Required order of operations and how to handle skipped prerequisites.
- Rules for conditional branches, exception paths, warnings, and stop conditions.
- How to treat missing materials, ambiguous requests, invalid states, and stale versions.
- Exact output schemas, labels, checklist fields, status values, or escalation phrases.
- Prohibitions against inventing steps, compressing mandatory checks, or using external
procedure knowledge.

[Context Requirements]
The context field must be a realistic procedural case file. It should include a manual or
protocol excerpt, transcript or operator notes, materials/status artifacts, version labels,
 and enough evidence for deterministic downstream questions.

[Quality Bar]
- The procedure must contain ordered steps, prerequisites, conditional branches, and at
least one hard stop or escalation condition.
- Artifacts should use raw tables, logs, checklists, forms, or policy excerpts without
explanatory hints.
- The scenario must require multi-hop reasoning across procedure text and execution records
.
- Do not add prose outside the JSON object.
Before final output, silently verify that the JSON parses, that the procedure is
enforceable, and that the case supports strict binary rubrics.
```

Figure 24: Stage G.1 context designer for *Procedural Execution*; this category returns both the system instruction and the context as one JSON object, so it bypasses stage G.2.

```
[Role]
You are designing a reusable System Instruction for a model that will answer questions
about the generated context.

[Subcategory]
{subcategory_name}
{subcategory_description}

[Behavioral Dimensions]
- Tone and style: {tone}
- Target audience: {audience}
- Reasoning focus: {focus}
- Structure constraint: {structure}

[Instruction Design Requirements]
- Write the instruction in second person, addressing the downstream model as "you".
- Give the model a concrete professional, academic, analyst, or domain-specific role; avoid
 generic "AI assistant" phrasing.
- Require the model to answer only from the provided context and not external facts.
- Define one exact fallback response for insufficient or underdetermined context.
- Include up to {constraint_count} concise, non-overlapping, verifiable constraints.
- Make constraints judgeable later: tone, structure, citation, evidence use, fallback
behavior, numeric precision, or output format.
- Do not include topic-specific facts, examples, or document content.
- Keep the instruction concise.

[Output]
Return only the System Instruction text. Do not add a title, label, code fence, bullets
about your process, or explanatory preface.
```

Figure 25: Stage G.2 reusable system-instruction designer, applied to the non-procedural categories.

```
[Role]
You are the lead task architect for context-learning / CL-Bench. Create context-learning
tasks from the supplied System Instruction and Context.

[System Instruction]
{system_section}

[Context]
{context}

[Design Objectives]
- Generate exactly {num_tasks} tasks.
- Each task must require synthesizing distributed information from the Context, such as
definitions, constraints, cases, tables, observations, procedures, exceptions, or priority
rules.
- Prefer decisions, classifications, calculations, predictions, state transitions,
operational artifacts, or formatted advisory outputs.
- Do not ask for summaries, vague opinions, trivia, or direct lookup when multi-step
reasoning is required.
- The answer field is the final user-facing answer only. Do not include hidden reasoning or
 chain-of-thought.

[Golden Answer Requirements]
- Use only the System Instruction and Context; do not import external facts.
- Obey the System Instruction's role, tone, structure, formatting, citation, fallback, and
refusal behavior.
- Be concise but complete: state the final outcome and include only the support required by
 the System Instruction.
- If rules, observations, or artifacts conflict, apply the stated priority, reliability
hierarchy, or validity boundary.

[Rubric Requirements]
Write at least {min_rubrics} rubrics for each task (aim for 12-18, matching CL-Bench's fine
 granularity). Every rubric must:
- Begin with "The response should ..." and state one binary, atomic, checkable property.
- Bake in the EXPECTED CONTENT using an "For example, ..." clause that names the exact
value, entity, threshold, section, unit, state, list, or field the correct answer must
contain (e.g. "The response should name the four sighting card types, namely: Decoys,
Hoaxes, Silhouettes, and Real Deals.").
- Be grounded in exact Context details, not vague quality judgments.
- Together, densely cover the answer so that a wrong final outcome fails MULTIPLE rubrics,
not just one.
Distribute the rubrics across these buckets (several each, not one each):
1. Correct final outcome, decision, prediction, classification, value, or artifact -- write
 MULTIPLE rubrics pinning the decisive result and each sub-part of it.
2. Correct use of each key context fact, rule, observation, threshold, or procedure step
the answer depends on.
3. Correct handling of exceptions, conflicts, priorities, noise, invalid states, or
boundary conditions.
4. Required answer format, tone, schema, citations, numeric precision, or fallback phrase.
5. Exclusion of unsupported assumptions, hallucinated evidence, or forbidden alternatives.

[Output Contract]
Return only valid JSON. Do not use markdown fences, comments, or prose outside JSON.
Schema:
{"tasks": [{"question": "...", "answer": "...", "rubrics": ["The response should ...", "The
 response should ..."]}]}
Before final output, silently verify that the JSON parses, that tasks are non-duplicative,
and that each task has at least {min_rubrics} concrete rubrics with the decisive outcome
covered by several of them.
```

Figure 26: Stage G.3 task and rubric architect: emits each task's question x, a golden answer, and the atomic binary training rubrics R^{tr}.

F.4 RECAP-RL REWARD: JUDGE GRADING PROMPT

We use Qwen3.8-27B as the judge model and the rubric-adherence rate as the reward (Figure 27). The braced `rubrics_text` and `model_output` fields are filled with the task's rubric list and the model's answer.

```
Starting now, you are a rigorous instruction-following grading teacher. Your task is to
accurately grade and score student answers based on the [Rubrics].

Grading Criteria
```

```
This is a strict, all-or-nothing grading system. The final score is binary.
To receive a score of 1, the student's answer must perfectly satisfy every single
requirement listed in the [Rubrics].
If even one requirement is not fully met, the final score will be 0.
Grading Process
Please strictly follow the steps below for analysis -- no steps may be skipped:
Step 1: Analyze the Standard Answer
List all explicit requirements in the [Rubrics] item by item (including format, content,
quantity, order, etc.).
Identify implicit requirements in the [Rubrics] (e.g., language style, logical structure).
Define specific evaluation criteria for each requirement (e.g., "must include X," "must not
 exceed Y").
Step 2: Check Each Requirement Against the Student's Answer
For every requirement in the [Rubrics], verify one by one whether the student's answer
fully satisfies it.
Step 3: Self-Reflection
Before giving the final score, you must conduct the following checks:
  Completeness Check: Whether all requirements in the standard answer have been reviewed
with no omissions.
  Strictness Check: Whether the evaluation strictly adheres to the "fully satisfied"
standard without relaxing requirements due to subjective judgment.
  Consistency Check: Whether the grading rationale aligns logically with the final score.
  Objectivity Check: Whether judgments are based on objective facts rather than subjective
speculation.
Output Format Requirements
[Grading Rationale]: xxx
[List of Requirement Satisfaction Status]: [x1, x2, ..., xi, ..., xn] (where n is the total
 number of requirements in the [Rubrics], and xi indicates whether the student's answer
meets the i-th requirement, with values "yes"/"no")
[Overall Score]: x points (x is an integer, either 0 or 1.)

Content to Be Graded
[Rubrics]:
{rubrics_text}
[Student Response]:
{model_output}

Please strictly output ONLY the following JSON format (do not output any other content):
{
  "Grading Rationale": "Your detailed grading rationale",
  "List of Requirement Satisfaction Status": ["yes", "no", ...],
  "Overall Score": 0 or 1
}
```

Figure 27: RECAP-RL reward: the judge prompt scored by Qwen3.8-27B. The reward is the rubric-adherence rate over the per-rubric `yes`/`no` list.