AGENTIC CONTEXT LEARNING WITH SELF-DISCOVERED SPECIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Context learning is an emerging inference-time task where LLMs learn and apply novel, task-specific knowledge from complex contexts. Unlike demonstration-based in-context learning, which emphasizes inferring task mappings from examples, context learning requires acquiring genuinely new knowledge that may be absent from pre-training—a capability that remains challenging even for frontier models today. In this work, we conduct a comprehensive empirical study to understand why models struggle with this capability. A natural hypothesis is that failures stem from *content access*; however, across twelve retrieval, reflection, and verification baselines on CL-Bench, we find limited gains over direct full-context prompting. Instead, our analysis reveals a key insight: recovering relevant content is insufficient; models must also infer *local specifications*—domain-specific formats, local rules, and completeness conditions—that are often unstated in the query but recoverable from the context. Our rubric-level analysis finds that specification acquisition is evaluated more than twice as often as content acquisition. Most specifications are unstated in the user query, yet nearly all are traceable to the context, indicating learnable obligations rather than hidden requirements. To validate this diagnosis, we introduce PSCI (private specification-contract induction), a deliberately simple intervention that extracts local specifications and enforces them through adversarial checking and repair. PSCI improves GPT-5.1 from 22.55% to a state-of-the-art 28.14% task success on CL-Bench, with consistent gains on Qwen3.5-27B and Gemini 3 Pro. Extensive ablations further isolate the role of task-specific specifications. Together, our findings suggest that effective context learning hinges not only on content acquisition but also on specification acquisition.

1 INTRODUCTION

Large language models are increasingly deployed in settings where the prompt is not merely evidence to retrieve from, but defines a novel operating environment the model must learn before it can act: a product manual, workflow policy, regulatory code, API specification, or experimental dataset, often spanning tens of thousands of tokens and specifying the rules of a local domain. Consider the task of *detecting anomalies in an incoming event stream*: the user prompt contains a 40K-token context followed by a simple query asking the model to record and log anomalies. The context defines not only the event semantics needed to detect anomalies, but also local specifications that determine what counts as a valid log: for instance, if a reserved event ID is skipped, the log must emit the exact flag `Sequence_Gap_Warning`. In this case, identifying the missing ID is the query-relevant content; emitting the prescribed flag is a validity-relevant specification. A response that detects the anomaly but omits the flag is still wrong. Thus, the challenge is not merely retrieving and reasoning over a long prompt, which is the focus of traditional long-context benchmarks (Liu et al., 2024; Hsieh et al., 2024; Bai et al., 2024), but acquiring the local specifications by which the answer is validated.

Despite strong performance across many language tasks, frontier LLMs remain fragile on context learning: the best model in the original CL-Bench evaluation scores below 24% strict task success. CL-Bench (Dou et al., 2026) is a recent benchmark designed specifically for context learning, with 1,899 diverse, long-context, contamination-controlled tasks (section 3). In this paper, we conduct a comprehensive empirical study of why this setting remains difficult. A natural hypothesis is that failures stem from content access. To test this hypothesis, we evaluate twelve methods from four families on CL-Bench: retrieval and context restructuring (Asai et al., 2024; Yan et al., 2024; Sarthi

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

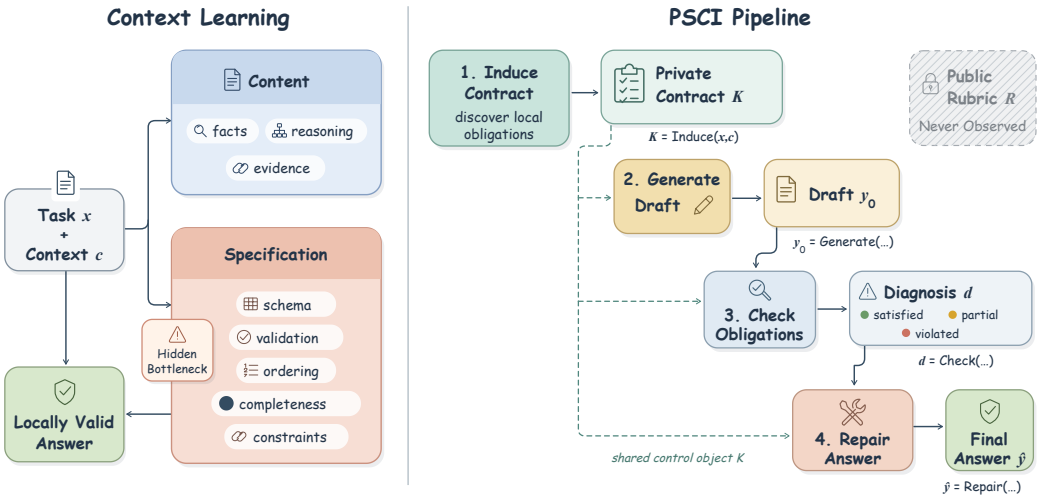


Figure 1: **Left:** Context learning decomposes into content acquisition (what the answer says) and specification acquisition (how the answer must behave). **Right:** PSCI induces a private specification contract from the task and context, then uses it as a shared control object for generation, checking, and repair. The public rubric for the current task is never observed during any stage.

et al., 2024; Edge et al., 2024; Sun et al., 2023), test-time self-correction (Shinn et al., 2023; Chen et al., 2025), self-generated rubric verification (Cook et al., 2024; Wan et al., 2026; Raghavendra et al., 2026), and iterative retrieval scaling (Yue et al., 2025). Surprisingly, none of the twelve exceeds the direct baseline (22.55% strict pass) by more than one point (subsection 5.2). This suggests that content access alone is insufficient and motivates a deeper investigation.

Analysis of failures reveals a clear pattern. Task-level strict pass is low (22.55%), but rubric-level pass is much higher (76.01%): models satisfy most requirements but miss a critical few, making failures near-misses rather than wholesale errors. Moreover, among tasks that fail by exactly one rubric item, 75.3% of the missed items evaluate not factual content but compliance with locally specified behavioral constraints: formats, edge-case handling, ordering, prescribed labels, or completeness conditions. We call these obligations **local specifications** and the ability to discover and enforce them **specification acquisition**. Taxonomizing all 31,592 CL-Bench rubric items confirms this is systematic: 55.4% evaluate specification acquisition, while only 22.6% evaluate content acquisition.

A key asymmetry explains why direct context-to-answer generation misses specifications despite having access to the full context. Only 23.3% of specification items are directly inferable from the user query; the remaining 76.7% are recoverable *only* from the surrounding context. Yet 95.5% of specification items are traceable to the context itself through explicit declarations, validation patterns, or demonstrated examples. Crucially, this establishes them as **learnable obligations** distributed across the prompt rather than hidden evaluator preferences. Thus, the issue is not that local specifications are absent or incomprehensible. Rather, they are query-implicit, low-salience, and scattered across background conventions, schemas, exceptions, logging rules, and validation requirements. This also explains why retrieval-oriented methods give limited gains: they tend to surface query-relevant, answer-bearing evidence, while missing validity-relevant specifications that govern the response's form, procedure, validation behavior, or required action. A method may therefore find the *content* needed to answer the apparent query while missing the local obligations needed for an acceptable response.

Surprisingly, when explicitly prompted to *induce specifications*, the model can recover much of this evaluation surface: privately induced contracts cover 94.4% of held-out public specification rubrics with less than 0.5% contradiction. This suggests that the missing step is not understanding the context in principle, but surfacing its local obligations as explicit generation constraints. In other words, the failure is neither missing knowledge nor hidden evaluator preference: the required behavior is present and largely recoverable from the context, but standard generation does not reliably activate it. To validate this diagnosis, we design a deliberately simple intervention: *private specification-contract induction* (PSCI), a test-time scaffold comprising four steps (Figure 1). The model first induces a task-

specific specification contract from the context itself, then answers under the contract; an adversarial checker audits the draft against the contract; and a repair step patches violations. The public rubric for the current task is never observed at any stage. On the full 1,899-task benchmark, PSCI achieves state-of-the-art 28.14% strict pass with GPT-5.1 (+5.59 pp absolute and +24.8% relative), with consistent gains across all four task families and replication on Qwen3.5-27B (14.14%→19.42%). **Seventeen** ablations isolate the role of task-specific specifications: shuffled contracts collapse below baseline, compute-matched generic critique returns to baseline, answer-first ordering fails. Overall, our contributions are threefold:

- Through comprehensive empirical analysis, we identify *specification acquisition*—inferring query-implicit but context-traceable local obligations—as a key bottleneck in context learning, and show these obligations are recoverable by explicit induction.
- To validate this diagnosis, we introduce PSCI, a deliberately simple test-time scaffold that achieves state-of-the-art CL-Bench performance across multiple model families.
- Through extensive ablation and mechanism analysis, we show that specification acquisition must be grounded in the context and decomposes into two separable stages: specification discovery and specification enforcement.

2 Related Work

Long-context evaluation and context learning. Long-context benchmarks evaluate evidence use over long inputs (Kamradt, 2023; Hsieh et al., 2024; Bai et al., 2024; Liu et al., 2024), typically under a fixed task interface. Context learning additionally requires inducing local validity conditions from the prompt. CL-Bench (Dou et al., 2026) is the only large-scale benchmark explicitly designed for this setting. Unlike instruction-following benchmarks (Zhou et al., 2023; Xia et al., 2024), where constraints are explicit in the query, CL-Bench specifications are often query-implicit and must be induced from local documentation.

Retrieval, memory, and context restructuring. Methods improving content access (Lewis et al., 2020; Asai et al., 2024; Yan et al., 2024; Sarthi et al., 2024; Edge et al., 2024; Sun et al., 2023; Packer et al., 2023; Yue et al., 2025) address a necessary but insufficient component of context learning: our twelve baselines show that content access alone does not solve specification acquisition (subsection 5.2).

Critique, verification, and structured prompting. Self-refinement (Madaan et al., 2023; Shinn et al., 2023), checklist verification (Cook et al., 2024; Wan et al., 2026; Raghavendra et al., 2026), and structured prompting (Wei et al., 2022; Yao et al., 2023; Khattab et al., 2024) all introduce intermediate computation. PSCI differs in inducing a task-specific obligation set from context *before* generation, then enforcing it through checking and repair. Ablations confirm this object, not the pipeline structure, drives the gain (subsection 5.3).

Extended discussion in Appendix F.

3 Empirical Study of Context Learning

Why CL-Bench. We use CL-Bench (Dou et al., 2026) as our target because it is, to our knowledge, the only benchmark specifically designed for context learning: 1,899 tasks and 31,592 rubric items across four families (domain knowledge reasoning, rule system application, procedural task execution, and empirical discovery & simulation), with contexts averaging 10.4K tokens and reaching 65K. All tasks are *human-expert-written* and require *knowledge absent from pre-training*, with contamination prevented by design (Appendix B.1). The breadth and quality of CL-Bench make it a strong testbed for an in-depth empirical study of context learning.

Initial hypothesis and experiment. A natural hypothesis is that the low performance ($< 24\%$ for GPT 5.1) stems from content access: the contexts are long and intricate, so methods that improve retrieval, restructuring, reflection, or verification should help. However, as shown in subsection 5.2, **twelve** representative methods from these families provide limited gains over direct full-context

Table 1: Specification subtype distribution and item-level pass rates by baseline and PSCI (%). Results show that every subtype is improved by PSCI; the largest gains are concentrated in harder subtypes.

Specification subtype	Items	Share	Base	PSCI
validation_or_edge_case	7,508	42.9%	81.0	85.0
exclusion_or_forbidden	2,573	14.7%	80.1	85.3
ordering_or_sequence	2,205	12.6%	68.4	74.6
schema_or_structured	1,663	9.5%	78.4	82.9
source_grounding	1,085	6.2%	79.1	84.1
coverage_or_completeness	805	4.6%	65.9	75.7
exact_phrase_constraint	525	3.0%	66.4	73.3
role_tone_style	210	1.2%	59.1	72.0

prompting. This motivates a closer diagnosis of what models actually miss. To tackle this, we next conduct an in-depth analysis of the task itself.

3.1 Analysis Protocol

We classify all 31,592 rubric items as evaluating *content acquisition* or *specification acquisition*: content if its verdict depends on which propositions the answer states; specification if it depends on whether the answer conforms to a locally defined rule governing form, procedure, or validation, or required action, holding propositions fixed. Two independent judges, an LLM-based classifier (GPT-5.1) and a deterministic rule-based classifier (with explicit abstention class), must agree for an item to be labeled; disagreements are marked *ambiguous*.

To validate the taxonomy, we additionally audit a random sample of 300 rubric items with three PhD-level annotators. The human audit agrees with the conservative classifier labels in over 90% of non-ambiguous cases (details in Appendix B.2).

3.2 Failures Are Often Near-Misses

The GPT-5.1 baseline (setup in subsection 5.1) passes 76.01% of individual rubric items but only 22.55% of tasks: most failures are near-misses rather than wholesale errors. Among the 281 tasks that fail by exactly one rubric item (validated in Appendix B.3), 75.3% of the missed items are specification items—local conventions the model failed to enforce rather than missing facts. This concentration suggests that addressing specification acquisition would yield disproportionate gains.

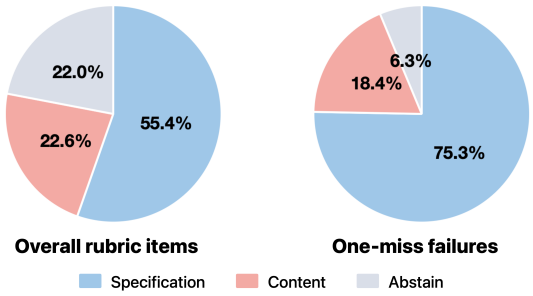


Figure 2: Rubric failure type breakdown by "content acquisition", "specification acquisition", and "abstain". Results show that the majority of original rubric items are "specification acquisition," while also being the main failure type among near-miss tasks.

3.3 Specifications Form the Largest Share of Rubric Surface

Applying the protocol to all 31,592 items (Figure 2), 55.4% are classified as specification, 22.6% as content, and 22.0% as ambiguous. Thus, even under a conservative criterion, specification acquisition accounts for the largest share of CL-Bench's evaluation surface. The specification class further decomposes into eight subtypes—from validation checks (42.9%) to role/tone constraints (1.2%)—confirming it captures multiple dimensions of locally valid behavior rather than surface formatting alone (full breakdown in Table 1).

3.4 Specifications Are Learnable but Query-Implicit

A natural concern is that specification items encode arbitrary evaluator preferences. We test this directly (Table 2; annotation protocol in Appendix B.4): 95.5% of specification items are traceable

to the context through explicit declarations, validation patterns, implicit examples, or repeated conventions. Only 0.4% are not derivable.

However, only 23.3% of specification items are inferable from the user query alone; the remaining 76.7% require reading the surrounding context. This asymmetry explains why both direct prompting and retrieval methods underperform: specifications are learnable obligations distributed across the prompt, but they are query-implicit, low-salience, and scattered across background conventions. Crucially, the problem is not that specifications are incomprehensible—when explicitly prompted to extract them, the model recovers 94.4% of held-out public rubrics with less than 0.5% contradiction. The capability exists, but is not reliably exercised during standard generation.

Table 2: Traceability of specification items to task/context. 95.5% are traceable to explicit declarations, validation patterns, examples, repeated structures.

Traceability class	Items	Share
Explicit declaration	12,531	71.6%
Validation pattern	3,693	21.1%
Implicit example	350	2.0%
Convention / repeated struct.	140	0.8%
Overall traceable	*16,714*	*95.5%*
Snippet too short to judge	718	4.1%
Not derivable from context	70	0.4%

4 PRIVATE SPECIFICATION-CONTRACT INDUCTION

The analysis in section 3 suggests that context learning failures are not explained by content access alone: many missed requirements are local specifications that are traceable to the context but largely absent from the user query. To validate this diagnosis, we design a deliberately simple intervention, Private Specification-Contract Induction (PSCI). PSCI tests whether making these latent specifications explicit before generation and enforcing them when forming responses improves context learning:

$$K = \mathrm{Induce}(x, c;\, \theta) \tag{1}$$
$$y_0 = \mathrm{Generate}(x, c, K;\, \theta) \tag{2}$$
$$d = \mathrm{Check}(x, c, K, y_0;\, \theta) \tag{3}$$
$$\hat{y} = \mathrm{Repair}(x, c, K, y_0, d;\, \theta) \tag{4}$$

where x is the task, c is the context, K is the induced specification contract, y_0 is the initial draft, d is a structured diagnosis, $\hat{y}$ is the final answer, and θ denotes the LLM parameters. The public rubric for the current task is never observed at any stage. The method is deliberately minimal: a complex pipeline would obscure whether gains stem from the diagnosis being correct or from engineering.

4.1 SPECIFICATION DISCOVERY

Given task x and context c, an LLM induces a private specification contract $K = \{k_1, k_2, \ldots, k_n\}$: a structured set of n obligation items the answer must satisfy to be locally valid, covering required formats, validation checks, ordering constraints, and exact local artifacts. The model then produces a draft $y_0 = \mathrm{Generate}(x, c, K;\, \theta)$ conditioned on the task, context, and contract jointly. We use rotated demonstrations drawn from disjoint context learning tasks only to teach contract style while the final contract used for generation is induced from the current task context; no same-task answer, public rubric, or task-specific contract is shown at any stage (details in Appendix C.1). **For analysis only**, after generation, we measure specification discovery quality via contract–rubric overlap:

$$\mathrm{Coverage}(K, R) = \frac{|\{r \in R : \exists\, k \in K \text{ that covers } r\}|}{|R|} \tag{5}$$

4.2 SPECIFICATION ENFORCEMENT

An adversarial checker functions as a diagnostic verifier. Given (x, c, K, y_0), it assesses each obligation $k_i \in K$ and produces:

$$d = \{(k_i,\, s_i,\, r_i)\}_{i=1}^{n}, \quad s_i \in \{\textsc{Satisfied}, \textsc{Partial}, \textsc{Violated}\} \tag{6}$$

Table 3: Main results on CL-Bench (1,899 tasks, 31,592 rubric items). *Overall*: strict-pass rate, global rubric (item micro-average), and mean task rubric (task macro-average). Per-family columns: strict-pass and rubric for DKR (Domain Knowledge Reasoning), RSA (Rule System Application), PTE (Procedural Task Execution), EDS (Empirical Discovery & Simulation). Baselines grouped by family (a–d) as defined in subsection 5.1. Subscripts = absolute gains over same-model baseline; best in **bold**. PSCI improves all evaluated model groups, surpassing even GPT-5.1 High (23.7%). *Frontier results from Dou et al. (2026); rubric/task columns unavailable.

		Overall			DKR		RSA		PTE		EDS	
Model	Method	Strict	Rubric	Task	Strict	Rubric	Strict	Rubric	Strict	Rubric	Strict	Rubric
*Other frontier models**	GPT 5.1 (High)	23.7±0.5	–	–	25.3	–	23.7	–	23.8	–	18.1	–
	Claude Opus 4.5 Think.	21.1±1.4	–	–	23.7	–	19.0	–	22.6	–	15.1	–
	GPT 5.2 (High)	18.1±0.8	–	–	18.6	–	17.2	–	21.4	–	11.7	–
	o3 (High)	17.8±0.2	–	–	18.0	–	17.6	–	19.5	–	13.7	–
	Kimi K2 Think.	17.6±0.6	–	–	18.7	–	17.0	–	18.8	–	12.6	–
	HY 2.0 Think.	17.2±0.6	–	–	18.0	–	17.3	–	19.4	–	8.9	–
	Gemini 3 Pro (High)	15.8±0.3	–	–	15.5	–	17.7	–	16.4	–	10.1	–
	Qwen 3 Max Think.	14.1±0.1	–	–	13.5	–	15.6	–	15.2	–	9.0	–
	Doubao 1.6 Think.	13.4±0.1	–	–	13.7	–	14.2	–	13.9	–	9.4	–
	DeepSeek V3.2 Think.	13.2±0.4	–	–	13.6	–	13.8	–	14.2	–	8.0	–
GPT-5.1 (Medium)	Baseline	22.55±0.6	76.01±1.2	71.87	23.83	80.23	23.23	72.97	21.06	75.29	18.18	69.87
	Self-RAG	20.67±0.7	71.11±1.8	66.55	24.07	75.13	17.98	68.33	21.43	73.77	15.15	61.61
	CRAG	19.00±1.4	70.79±2.1	68.13	24.07	77.07	13.48	64.30	20.00	77.68	6.06	54.82
	RankGPT	22.33±2.4	69.72±2.5	67.37	22.22	76.42	10.11	63.90	22.86	76.27	18.18	52.12
	RAPTOR	20.67±0.2	71.30±1.4	66.96	28.70	79.59	16.85	69.35	17.14	75.25	12.12	47.59
	GraphRAG	23.33±0.5	75.47±1.9	67.57	28.63	77.82	14.61	65.09	**35.71**	80.03	9.09	75.35
	SimpleMem	20.00±1.5	73.19±2.0	67.00	29.63	78.59	12.36	67.14	21.43	75.41	6.06	67.00
	Reflexion	22.33±1.2	71.64±1.9	70.79	26.85	79.82	19.10	69.12	20.00	72.83	**21.21**	54.25
	Guideline Forest	22.00±0.9	73.82±1.8	70.19	26.85	81.35	16.85	70.22	22.86	79.09	18.18	52.55
	TICK/STICK	23.00±0.3	74.20±1.4	67.40	27.70	76.30	20.22	67.85	27.14	74.16	12.12	78.74
	DeepVerifier	21.33±2.7	70.55±2.2	65.90	29.63	79.71	16.85	60.82	17.14	68.52	15.15	69.55
	Agentic Rubrics	20.33±2.2	69.99±1.7	65.10	26.85	72.67	14.61	65.64	24.29	69.22	6.06	72.80
	IterDRAG	22.00±0.1	72.10±2.9	66.87	28.70	73.67	20.22	65.88	18.57	74.31	12.12	75.50
	Baseline+SI	26.09±1.2$_{+3.51}$	79.10±1.6$_{+3.09}$	75.39	28.96	84.21	24.29	74.52	27.23	78.83	**21.21**	72.48
	PSCI	**28.14**±0.5$_{+5.59}$	**81.13**±0.9$_{+5.12}$	**77.57**	**31.67**	**84.84**	**25.18**	**76.70**	29.15	**81.04**	**21.21**	**79.12**
Gemini 3 Pro	Baseline	16.14±0.2	70.31±1.2	68.64	15.60	69.55	18.96	62.48	16.82	68.49	12.12	57.33
	Baseline+SI	18.38±0.6$_{+2.24}$	71.71±1.4$_{+1.40}$	70.89	19.70	73.34	17.89	72.45	18.43	72.95	14.77	64.27
	PSCI	**22.31**±0.1$_{+6.17}$	**76.75**±2.6$_{+5.44}$	**72.02**	**23.42**	**79.16**	**23.05**	**74.54**	**20.40**	**75.61**	**17.91**	**68.83**
Qwen3.5-27B	Baseline	14.14±2.7	69.41±2.9	63.87	15.84	72.97	12.06	65.49	17.02	71.58	8.59	54.61
	Baseline+SI	17.41±1.8$_{+3.27}$	73.10±4.0$_{+3.69}$	69.00	18.85	76.59	17.02	68.44	19.15	73.97	10.61	65.36
	PSCI	**19.42**±2.4$_{+5.28}$	**74.58**±3.3$_{+5.17}$	**70.98**	**20.97**	**77.44**	**18.79**	**71.20**	**21.49**	**75.40**	**12.12**	**65.65**

where s_i is the compliance status and r_i is a natural-language rationale localizing the mismatch. The repair model produces $\hat{y} = \text{Repair}(x, c, K, y_0, d; \theta)$, preserving correct content while patching obligations diagnosed as PARTIAL or VIOLATED. Crucially, the checker targets the induced contract K, not the draft's surface quality—decoupling verification from generation. Two design decisions distinguish PSCI from generic self-refinement: the contract is induced *before* generation (control object, not post-hoc grader), and the checker is grounded in K rather than open-ended quality. Both are verified empirically in subsection 5.3 (more analysis in Appendix C.2).

5 EXPERIMENTS

5.1 SETUP

Benchmark. We evaluate on CL-Bench (Dou et al., 2026), which contains 1,899 tasks across four families: domain knowledge reasoning (DKR), rule system application (RSA), procedural task execution (PTE), and empirical discovery & simulation (EDS). The benchmark is scored by 31,592 rubric items, averaging 16.64 items per task. Contexts average 10.4K tokens and reach 65K. A task is solved only when *all* associated rubric items pass; even GPT-5.1 with high reasoning effort achieves only 23.7% strict pass in the original evaluation.

Models. We use GPT-5.1 (OpenAI, 2025) with medium reasoning effort as the primary model. GPT-5.1 is the strongest model in the original CL-Bench evaluation, and medium effort gives the

Table 4: Ablation study (full benchmark, GPT-5.1 Medium). "+" = added to Baseline. Spec = specification contract; Chk = checker; Rep = repair. *Italics* = rubric below baseline (76.01%). We highlight three key results: shuffled specs collapse below baseline (A7/A10), spec-free critique returns to baseline (A13)—extra compute alone does not explain the gain, and answer-first ordering underperforms PSCI by 4.47 pp.

	Method	Strict Pass (%)	Rubric Pass (%)
Pipeline ablations	Baseline	22.55±0.6	76.01±1.2
	A1: + Spec	26.09±1.2	79.10±1.6
	A2: + Repair only	23.02±0.4	*71.68*±1.6
	A3: + Chk + Repair (no spec)	24.32±0.7	77.44±1.5
	A4: + Spec + Repair (no chk)	26.30±1.0	80.15±2.3
	A5: + Spec + Combined Chk&Rep	25.90±0.4	78.80±2.5
	A6: + Answer-first Spec + Chk + Rep	23.67±1.0	*70.67*±1.7
	PSCI (Ours)	**28.14**±0.5	**81.13**±0.9
Spec quality (discovery only)	A7: + Spec (Shuffled)	19.80±1.5	*66.55*±1.2
	A8: + Spec (Zero-shot)	25.01±0.7	*72.69*±2.1
	A9: + Spec (Generic)	23.80±0.9	76.76±2.6
Spec quality (full pipeline)	A10: + Spec (Shuffled) + Chk + Rep	22.00±1.2	*66.55*±2.1
	A11: + Spec (Zero-shot) + Chk + Rep	26.67±1.1	*78.69*±1.2
	A12: + Spec (Generic) + Chk + Rep	24.00±0.5	74.76±1.7
Compute-matched controls	A13: + Spec-free Critique	22.67±0.8	*70.06*±0.9
	A14: + Self-reflection	23.02±0.4	*71.24*±1.4
	A15: + Majority Vote	24.12±0.9	*74.23*±1.9
	A16: + Repeated Static Spec	24.30±1.9	78.45±3.1
	A17: + Repeated Zero-shot Spec	24.67±1.8	79.10±1.5

strongest direct baseline in our setup, making it a conservative comparison point for PSCI. To test cross-model generalization, we additionally evaluate on Gemini 3 Pro (high) (Google DeepMind, 2025) and Qwen3.5-27B (Qwen Team, 2026).

Baselines. The direct baseline is single-turn full-context prompting. We compare against twelve methods from four families: **(a)** retrieval and context restructuring: Self-RAG (Asai et al., 2024), CRAG (Yan et al., 2024), RAPTOR (Sarthi et al., 2024), GraphRAG (Edge et al., 2024), RankGPT (Sun et al., 2023), and SimpleMem; **(b)** self-correction: Reflexion (Shinn et al., 2023) and Guideline Forest (Chen et al., 2025); **(c)** self-generated rubric verification: TICK/STICK (Cook et al., 2024), DeepVerifier (Wan et al., 2026), and Agentic Rubrics (Raghavendra et al., 2026); and **(d)** iterative retrieval: IterDRAG (Yue et al., 2025). All baselines use the same GPT-5.1 medium-effort setting with full prompts and implementation details provided in subsection C.4.

Metrics and protocol. We report strict task pass rate, global rubric pass rate (item-level micro-average), and mean task rubric (task-level macro-average), both overall and by task family. We follow the released CL-Bench evaluation protocol, using the same GPT-5.1 verifier and evaluation code, which the benchmark reports as achieving over 90% agreement with human judgments. For each method and model, we run the full benchmark three times and report mean ± standard deviation. The complete PSCI pipeline constitutes one run; reruns are used only for API infrastructure failures, with no selective filtering or cherry-picking (additional eval info in subsection C.4).

To validate that gains are not an artifact of automatic evaluation, three PhD-level annotators blindly audit 100 baseline-fail/PSCI-pass tasks. A majority vote prefers the PSCI answer in 96/100 (96%) of cases and confirms the automatic flip judgment in 92/100 (92%) (protocol in Appendix G).

5.2 Main Results

PSCI improves across model families. Table 3 reports full-benchmark results. PSCI improves GPT-5.1 strict pass from 22.55% to 28.14%, a +5.59 pp absolute gain and 24.8% relative improvement. Global rubric pass also increases by 5.12 pp. The improvement replicates on Qwen3.5-27B, where PSCI improves strict pass from 14.14% to 19.42% (+5.28 pp) and global rubric pass by +5.17 pp. The gain also holds on Gemini 3 Pro, improving strict pass from 16.14% to 22.31% (+6.17

Table 5: Overlap (%) between privately induced contracts and non-ambiguous public (ground-truth) rubrics. The contract recovers most of the public rubric surface, especially for specification items.

	Items	Strong/Human	Partial	Contradict
Content	7,140	54.9/60.4	38.7	0.38
Spec	17,501	62.5/69.5	31.9	0.52
All	24,640	60.8/64.95	33.4	0.49

pp). Gains are consistent across all four CL-Bench task families. Notably, PSCI with GPT-5.1 medium (28.14%) exceeds GPT-5.1 high from the original CL-Bench (23.7%), suggesting explicit specification induction is a more effective use of inference compute than increasing reasoning effort alone.

The gains align with the diagnosis. PSCI improves both rubric types, but gains are larger on specifications. On GPT-5.1, specification-item pass rate rises from 76.7% to 82.1%, closing 23.2% of the available gap; content-item pass rate rises from 73.4% to 77.5%, closing 15.5%. Among rubric items fixed on tasks that flip from baseline-fail to PSCI-pass, 77.7% are specification items, showing PSCI primarily addresses the diagnosed specification bottleneck while also improving content use.

Existing methods do not match PSCI. As discussed in section 1, all twelve alternative methods remain close to the direct baseline, ranging from 19.00% to 23.33% strict pass, with none exceeding the baseline by more than 0.78 percentage points. Retrieval and restructuring methods often underperform full-context prompting, consistent with the hypothesis that context selection can discard distributed specification signals. Self-generated rubric methods, which are closest in spirit to PSCI, also fail to meaningfully exceed the baseline. Therefore, the gain is not explained by generic checklist verification; it requires context-grounded specification induction and obligation-level checking.

5.3 Ablation Studies

To isolate the mechanism, we run 17 ablations (A1–A17) on the full 1,899-task benchmark under the same GPT-5.1 medium-effort setting (Table 4).

Each PSCI component contributes. Specification induction alone (A1) reaches 26.09%, accounting for 63% of PSCI's total strict-pass gain. Checker and repair add a further +2.05 points. In contrast, repair alone (A2, 23.02%) and checker+repair without a specification contract (A3, 24.32%) yield only limited gains, showing that the contract is the primary driver. Separating diagnosis from repair is also important: spec+repair without an explicit checker (A4, 26.30%) and a combined check-and-repair step (A5, 25.90%) both underperform PSCI.

Extra compute does not explain the gain. Five compute-matched alternatives without task-specific specifications all fall short. Spec-free critique (A13) returns to 22.67%, and self-reflection (A14) to 23.02%. Even repeated contract-like structures without task-specific grounding (A16–A17) plateau at 24.30–24.67%. Thus, additional inference and generic self-correction are insufficient; the gain requires a task-specific specification contract.

Finding 1: Specifications must be extracted separately (A7, A10). To test whether contracts are merely helpful prompt prefixes, we replace each contract with one drawn from another task. Shuffled contracts collapse below baseline: A7 drops to 19.80% strict pass and A10 to 22.00%, with rubric pass falling to 66.55%. A misaligned specification is worse than no specification, demonstrating that task-specific alignment is causally necessary.

Finding 2: Specifications must be grounded in context (A8, A11, A12). Generic specifications underperform context-derived ones. Zero-shot context-derived specifications through the full pipeline (A11) reach 26.67%, while generic static specifications (A12) reach only 24.00%. The hierarchy is clear: generic specifications $\ll$ zero-shot context-derived specifications $<$ few-shot context-derived PSCI. Zero-shot context-derived specifications already improve over baseline, while generic and shuffled specifications underperform, showing that demonstrations refine contract granularity but do not create the effect. Extracting task-specific specifications from the current context is key.

Finding 3: Specifications must be enforced (A1, A4–A5). Discovery alone (A1, 26.09%) remains below PSCI, showing that specification discovery is not sufficient. Spec+repair without an explicit checker (A4, 26.30%) and combined check-and-repair (A5, 25.90%) also underperform PSCI, indicating that enforcement benefits from explicit checking and targeted repair. This separates discovery from compliance: the checker exposes concrete spec violation targets, enabling compliant repair rather than unconstrained rewriting.

Finding 4: Contracts Must Act as Control Objects. The answer-first ablation (A6) tests whether specifications can be induced after generation and used only as repair guidance. It cannot: A6 drops to 23.67% strict pass and 70.67% rubric pass, far below PSCI. Since only 23.3% of specifications are inferable from the user query, direct generation often fixes the answer structure before scattered obligations become active constraints. Post-hoc repair can patch local violations, but cannot reliably restructure fields, evidence use, or completeness conditions. Thus, the contract must shape generation as a control object rather than act as a post-hoc grader, explaining why self-generated rubric verification methods do not match PSCI.

Together, shuffled contracts, answer-first induction, and compute-matched controls rule out three simpler explanations for PSCI's gains: generic prompting structure, post-hoc verification, and additional inference compute.

6 Discussion

6.1 Quality of Induced Specifications

For analysis only, we compare private contracts with held-out public rubrics never observed by PSCI, labeling each item as strongly covered, partially covered, contradicted, or uncovered (details in subsection H.1). Across 24,640 non-ambiguous items, contracts strongly cover 60.8% and partially cover 33.4%, totaling 94.2% coverage with only 0.49% contradiction (Table 5). Specification items are more strongly covered than content items (62.5% vs. 54.9%), and human audit confirms the automatic measurement is conservative (64.95% vs. 60.8% strong coverage; Appendix E.2).

Coverage also tracks correctness: pass rate rises from contradicted (25.4%) and uncovered (48.7%) items to partially covered (75.2%) and strongly covered items (88.8%). Thus, induced contracts are not decorative; better coverage corresponds to downstream success, with a 40-point pass-rate gap between strongly covered and uncovered items.

6.2 The Enforcement Bottleneck

Among rubric items still failed by PSCI, 83% are covered by the contract but not executed in the final answer; only 14.9% are discovery failures and 2.0% are contradictions. Thus, PSCI substantially improves specification discovery, but enforcement becomes the dominant residual bottleneck: current models often surface the right local obligations but fail to consistently satisfy them during generation and repair. In other words, these results suggest that the central bottleneck is no longer whether models can recover local specifications, but whether those specifications become sufficiently explicit and enforceable to govern generation. Future work should prioritize stronger enforcement mechanisms, such as iterative verification, more reliable repair, or training-time objectives that internalize specification compliance.

7 Conclusion

We identify local specification acquisition as a key bottleneck in context learning: models often recover answer-relevant content while missing query-implicit, context-traceable obligations that determine local validity. This diagnosis motivates PSCI, a simple test-time scaffold that makes local specifications explicit before generation and enforces them through checking and repair. Our results show that context learning requires not only content acquisition, but also context-grounded specification discovery and enforcement. Our ablations further show that these gains depend on task-specific, context-grounded specifications rather than additional inference alone. We hope this perspective guides future work toward more reliable specification-aware context learning.

AI USE STATEMENT

Generative AI tools were used to assist with the implementation of author-developed research ideas, manuscript editing, and figure preparation. The authors reviewed and verified all AI-assisted outputs and retained full responsibility for the research methodology, experimental results, analysis, and final manuscript content.

ETHICS STATEMENT

PSCI is a test-time prompting scaffold evaluated on CL-Bench, a public benchmark whose contexts are expert-crafted through fictional creation or modification of existing content. No private user data is collected or generated at any stage. The specification contracts induced by PSCI are task-specific intermediate objects used only within the immediate generation pipeline; they are not stored, shared, or reused beyond evaluation. All evaluation uses the publicly released CL-Bench evaluation code and rubrics. We emphasize that improved context learning performance should not be interpreted as a guarantee of factual correctness or safety compliance in deployment, since specification acquisition addresses local validity within a provided context rather than global truthfulness.

Because PSCI extracts and enforces local specifications, it may also over-enforce brittle, biased, unsafe, or undesirable local rules if such rules appear in the context. This is especially important in legal, medical, financial, or institutional settings, where local policies may conflict with fairness, safety, or broader legal or ethical constraints. PSCI should therefore be used with expert oversight in high-stakes domains and should not be treated as a guarantee that the extracted specification is normatively appropriate.

REFERENCES

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=hSyW5go0v8.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

Jiaxiang Chen, Zhuo Wang, Mingxi Zou, Qifan Wang, and Zenglin Xu. Guideline forest: Experience-induced multi-guideline reasoning with stepwise aggregation. *arXiv preprint arXiv:2506.07820*, 2025. URL https://arxiv.org/abs/2506.07820.

Jonathan Cook, Tim Rocktäschel, Jakob Foerster, Dennis Aumiller, and Alex Wang. TICKing all the boxes: Generated checklists improve LLM evaluation and generation. In *NeurIPS 2024 Workshop on Language Gamification*, 2024. URL https://openreview.net/forum?id=Q3y6QhOUnI.

Shihan Dou, Ming Zhang, Zhangyue Yin, Chenhao Huang, Yujiong Shen, Junzhe Wang, Jiayi Chen, Yuchen Ni, Junjie Ye, Cheng Zhang, Huaibing Xie, Jianglu Hu, Shaolei Wang, Weichao Wang, Yanling Xiao, Yiting Liu, Zenan Xu, Zhen Guo, Pluto Zhou, Tao Gui, Zuxuan Wu, Xipeng Qiu, Qi Zhang, Xuanjing Huang, Yu-Gang Jiang, Di Wang, and Shunyu Yao. CL-bench: A benchmark for context learning. *arXiv preprint arXiv:2602.03587*, 2026. URL https://arxiv.org/abs/2602.03587.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. From local to global: A Graph RAG approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024. URL https://arxiv.org/abs/2404.16130.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023. URL https://arxiv.org/abs/2312.10997.

Google DeepMind. Gemini 3 Pro model card, 2025. URL https://deepmind.google/models/model-cards/gemini-3-pro/. Model card.

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? In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=kIoBbc76Sy.

Greg Kamradt. Needle in a haystack – pressure testing LLMs, 2023. URL https://github.com/gkamradt/LLMTest_NeedleInAHaystack. GitHub repository.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=sY5N0zY5Od.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, volume 35, pp. 22199–22213, 2022.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474, 2020.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638. URL https://aclanthology.org/2024.tacl-1.9/.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

OpenAI. GPT-5 system card, August 2025. URL https://openai.com/index/gpt-5-system-card/. System card.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. URL https://arxiv.org/abs/2310.08560.

Qwen Team. Qwen3.5-27B, 2026. URL https://huggingface.co/Qwen/Qwen3.5-27B. Official model card.

Mohit Raghavendra, Anisha Gunjal, Bing Liu, and Yunzhong He. Agentic rubrics as contextual verifiers for SWE agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15265–15290. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.697. URL https://aclanthology.org/2026.acl-long.697/.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. RAPTOR: Recursive abstractive processing for tree-organized retrieval. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=GN921JHCRw.

William Saunders, Catherine Yeh, Jeff Wu, Steven Bills, Long Ouyang, Jonathan Ward, and Jan Leike. Self-critiquing models for assisting human evaluators. *arXiv preprint arXiv:2206.05802*, 2022. URL https://arxiv.org/abs/2206.05802.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. Is ChatGPT good at search? investigating large language models as re-ranking agents. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 14918–14937. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.923. URL https://aclanthology.org/2023.emnlp-main.923/.

Yuxuan Wan, Tianqing Fang, Zaitang Li, Yintong Huo, Wenxuan Wang, Haitao Mi, Dong Yu, and Michael R. Lyu. Inference-time scaling of verification: Self-evolving deep research agents via test-time rubric-guided verification. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 24822–24835. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.1243. URL https://aclanthology.org/2026.findings-acl.1243/.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Congying Xia, Chen Xing, Jiangshu Du, Xinyi Yang, Yihao Feng, Ran Xu, Wenpeng Yin, and Caiming Xiong. FOFO: A benchmark to evaluate LLMs' format-following capability. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 680–699. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.40. URL https://aclanthology.org/2024.acl-long.40/.

Shi-Qi Yan, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling. Corrective retrieval augmented generation. *arXiv preprint arXiv:2401.15884*, 2024. URL https://arxiv.org/abs/2401.15884.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Zhenrui Yue, Honglei Zhuang, Aijun Bai, Kai Hui, Rolf Jagerman, Hansi Zeng, Zhen Qin, Dong Wang, Xuanhui Wang, and Michael Bendersky. Inference scaling for long-context retrieval augmented generation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=FSjIrOm1vz.

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

SUPPLEMENTARY MATERIAL

Table of Contents

A Limitations **14**

B Supplementary Material for Empirical Study **14**

B.1 CL-Bench Design and Contamination Prevention 14

B.2 Classification Protocol Details 14

B.3 Near-Miss Validation 15

B.4 Traceability Annotation Protocol 15

B.5 Content vs. Specification: Worked Examples 15

C Supplementary Material for Method **16**

C.1 Rotated Exemplar Design 16

C.2 Design Decisions: Extended Discussion 16

C.3 Prompt Templates 16

C.4 Evaluation Protocol Details 16

D Additional Baseline Details **17**

E Additional result visualizations. **18**

E.1 Visualization of ablation study results. 18

E.2 Human Validation of Contract–Rubric Overlap 18

F Extended Related Work **19**

F.1 Long-Context Evaluation, Instruction Following, and Context Learning 19

F.2 Retrieval, Memory, and Context Restructuring 19

F.3 Critique, Verification, and Structured Intermediate Computation 20

G Human Audit of Baseline-to-PSCI Flips **20**

H Additional material for discussion **21**

H.1 Coverage Label Definitions and Examples 21

H.2 Extended Discussion: Why Contracts Must Precede Generation 21

I Reproducibility: PSCI Prompt Templates **22**

J Qualitative Examples of Baseline-to-PSCI Fixes **25**

J.1 Manufacturing Assembly: Following the Local Runbook 25

J.2 Rowing Synchronization: Producing the Required Analysis Object 26

A LIMITATIONS

Benchmark scope. We mainly target our analysis and experiments on the single benchmark, CL-Bench, designed specifically for context learning. Although this limits claims about generalization beyond the current context learning benchmark landscape on the surface, CL-Bench is currently the only large-scale benchmark explicitly designed for context learning, and it is unusually broad for a single benchmark: 1,899 human-expert-written, contamination-controlled tasks, 31,592 rubric items, four task families, and contexts averaging 10.4K tokens and reaching 65K. Thus, while evaluation on future context learning benchmarks would strengthen external validity, CL-Bench already covers a diverse set of local domains, including domain knowledge, rule systems, procedures, and empirical discovery.

Inference cost. PSCI is an inference-time scaling method and is therefore more expensive than single-pass generation: it uses four sequential model calls for specification induction, contract-guided answer generation, contract-aware checking, and targeted repair. We treat this as a central tradeoff rather than a hidden cost. Compute-matched and compute-heavier controls show that additional inference alone does not explain the gain: generic self-reflection, iterative retrieval, guideline-based reasoning, and verification-style baselines use comparable or larger budgets but fail to match PSCI. Thus, the improvement comes from how computation is structured—around task-specific specification discovery and enforcement—rather than from extra tokens alone. Future work should explore cheaper ways to extract and enforce local specifications, including distillation, adaptive early stopping, or training-time internalization.

Prompting-based implementation. PSCI is training-free and immediately deployable, but prompting alone may be suboptimal for enforcing specifications reliably. Our residual-error analysis suggests that many remaining failures occur when the contract covers an obligation but the final answer does not execute it. This points to future work on stronger enforcement mechanisms, iterative repair, or training-time objectives that internalize specification compliance while reducing inference cost.

B SUPPLEMENTARY MATERIAL FOR EMPIRICAL STUDY

B.1 CL-BENCH DESIGN AND CONTAMINATION PREVENTION

CL-Bench (Dou et al., 2026) prevents contamination through three approaches: (1) *fictional creation*—experts construct entirely novel content such as complete legal systems for fictional countries or new programming languages with unique syntax; (2) *modification of existing content*—altering real-world knowledge such as changing scientific definitions, historical events, or technical specifications; and (3) *incorporation of niche and emerging content*—using cutting-edge or narrow-domain knowledge not well-represented in pre-training corpora. Each context requires approximately 20 hours of expert annotation effort and undergoes multiple rounds of quality review.

A context-free ablation confirms the effectiveness of this design: GPT-5.1 solves less than 1% of tasks when the context is removed (Dou et al., 2026), confirming that CL-Bench tasks cannot be solved from pre-trained knowledge alone.

B.2 CLASSIFICATION PROTOCOL DETAILS

The LLM-based classifier (GPT-5.1, the same model used by CL-Bench for rubric evaluation) receives the full task context and rubric text and labels each item as content or specification. The deterministic rule-based classifier combines lexical cues (e.g., "must include," "format as"), structural cues (e.g., schema references, enumerated constraints), and semantic cues (e.g., whether the item targets factual derivation or behavioral compliance). It includes an explicit abstention label for items where no rule fires with sufficient confidence.

Under the agreement protocol, an item is labeled specification or content only when both classifiers independently produce the same label. Items where the classifiers disagree or where the deterministic classifier abstains are labeled *ambiguous* and excluded from both counts. This conservative design ensures that the reported 55.4% specification share and 22.6% content share represent high-precision estimates rather than aggressive forced-choice labels.

To validate the taxonomy, three PhD-level annotators independently audit a random sample of 300 rubric items. Annotators are shown the task context and rubric item, but not the automatic classifier labels. They assign each item to content, specification, or ambiguous using the same operational definitions. Final human labels are determined by majority vote. On items assigned non-ambiguous labels by the conservative automatic protocol, the human majority label agrees with the automatic label in over 90% of cases, supporting the reliability of the taxonomy.

B.3 Near-Miss Validation

To confirm that the 281 single-item-failure tasks are genuine near-misses, we remove the single failed rubric item from each task and re-run the full generation and judging pipeline. After three independent runs, 99% of these tasks pass, confirming that the identified rubric item is indeed the sole point of failure.

This finding is further consistent with CL-Bench's own error analysis, which reports format errors at 35%+ and context misuse exceeding 60% of failures even for top-performing models (Dou et al., 2026).

B.4 Traceability Annotation Protocol

For each specification item, we annotate whether the obligation can be traced to the task or context, and through which evidentiary channel:

- **Explicit declaration** (71.6%): The context directly states the requirement (e.g., "if a gap is detected, emit `Sequence_Gap_Warning`").
- **Validation pattern** (21.1%): The context demonstrates a validation procedure or schema implying what the output must satisfy.
- **Implicit example** (2.0%): The context includes a worked example whose format implies the expected behavior.
- **Repeated convention** (0.8%): The context uses a consistent structure repeatedly, implying the answer should follow the same pattern.
- **Not derivable** (0.4%): The obligation cannot be traced to the task or context.
- **Too short to judge** (4.1%): The rubric snippet is too brief to determine traceability.

To validate these annotations, three PhD-level annotators each independently classified 100 randomly sampled specification items, with final labels determined by majority vote. Human annotators confirmed traceability at rates consistent with or higher than the automated system, supporting the reliability of the reported 95.5% rate.

B.5 Content vs. Specification: Worked Examples

Content item. "The response should identify Event ID 4072 as the anomalous entry." The verdict depends on whether the answer states the correct proposition. Any format communicating this fact satisfies the item.

Specification item. "When a reserved event ID is skipped, the log must emit the flag `Sequence_Gap_Warning`." The verdict depends on conforming to a locally defined logging convention. Identifying the gap but logging it as "missing_id" would fail despite containing the correct content.

Ambiguous item. "The response should include a summary table of all detected anomalies with timestamps." This evaluates both content (which anomalies) and specification (table format). Under our protocol, such items are labeled ambiguous.

C Supplementary Material for Method

C.1 Rotated Exemplar Design

The contract induction step uses a small set of rotated demonstrations to teach the style and granularity of specification contracts. For each evaluation task, demonstrations are drawn from other disjoint context learning tasks and never from the current task. Thus, the current task's answer, public rubric, context, or task-specific contract is never shown during induction. The final contract used for generation is induced from the current task context. Each demonstration illustrates how to express local obligations—such as required fields, validation checks, exact artifacts, exclusions, ordering constraints, and completeness rules—but does not provide task-specific information for the current example. We use 10 demonstrations per task. Zero-shot induction without demonstrations still improves over the baseline (A8/A11 in Table 4), while generic and shuffled specifications underperform, showing that demonstrations refine contract granularity but do not create the effect.

C.2 Design Decisions: Extended Discussion

Two design decisions distinguish PSCI from generic self-refinement and merit explicit justification.

Contract before answer. The contract K is induced before the draft y_0, so it functions as a control object that shapes generation rather than a post-hoc grading rubric applied to a finished draft. This is motivated by the traceability analysis in subsection 3.4: since 76.7% of specifications are absent from the query, they must be surfaced before generation begins. If the model generates first, it anchors to a query-driven structure that post-hoc repair cannot fully restructure. Ablation A6 confirms this empirically: answer-first ordering collapses to 23.67% strict pass and 70.67% rubric, which remains far below PSCI, and its rubric pass falls below the direct baseline.

Checker targets the contract. The checker assesses the draft against the induced contract K, not against open-ended quality criteria. This design prevents the checker from simply restating what the answer already says and ensures that verification targets specific obligations. Ablation A3 (checker + repair without contract, 24.32%) versus PSCI (28.14%) confirms that grounding verification in a specification contract is essential.

C.3 Prompt Templates

Please see Appendix I for the full prompt that can reproduce PSCI.

C.4 Evaluation Protocol Details

Verifier. We use the exact evaluation code and GPT-5.1-based verifier released by the CL-Bench authors (Dou et al., 2026). Each rubric item is evaluated as a binary yes/no question; a task is passed only when all associated items receive "yes." The CL-Bench authors report over 90% agreement between GPT-5.1 judgments and human annotations in their validation study, and further show that raw agreement between GPT-5.1 and two alternative verifiers (Claude Opus 4.5, Qwen-3-Max) exceeds 90%, indicating minimal self-evaluation bias.

Run protocol. For each method and model combination, we conduct three independent runs on the full 1,899-task set and report the mean along with standard deviation. For PSCI, the complete end-to-end pipeline ($\mathrm{Induce} \rightarrow \mathrm{Generate} \rightarrow \mathrm{Check} \rightarrow \mathrm{Repair}$) constitutes a single run, reflecting real-world deployability where the four stages execute sequentially without human intervention. We apply reruns only for API infrastructure failures (rate-limit errors, timeouts, connection drops); no selective filtering, output inspection, or cherry-picking is performed at any stage.

Baseline implementation. All twelve baselines are implemented following the original papers as closely as possible and evaluated under the same GPT-5.1 medium-thinking setting. For methods requiring retrieval (Self-RAG, CRAG, RAPTOR, GraphRAG, RankGPT, IterDRAG), the retrieval corpus is the task's own context—no external knowledge base is used, consistent with CL-Bench's

Table 6: Compute and performance comparison. Token counts exclude judge calls and report generated tokens across method stages. Latency is wall-clock time per task under the same API/model setting. PSCI is more expensive than single-pass generation, but compute-matched or compute-heavier alternatives do not recover its gains, indicating that performance comes from task-specific specification induction and enforcement rather than inference budget alone.

Method	LLM stages/task	Gen. tokens/task	Latency	Strict pass	Rubric pass
GPT Baseline	1	18.9k	44.6s	22.55	76.01
Baseline + SI	2	56.1k	54.9s	26.09	79.10
PSCI	4	91.6k	132.2s	28.14	81.13
Compute-Matched Majority Vote	5	94.5k	223s	24.12	74.23
Compute-Matched Self-Reflection	5	94.5k	254s	23.02	71.24
Reflexion	3	47.2k	108.5s	22.33	71.64
IterDRAG	multiple/iterative	70.8k	166.6s	22.00	72.10
Guideline Forest	multiple/branching	84.5k	246.6s	22.00	73.82
TICK/STICK	multiple/variable	67.9k	165.5s	24.00	74.20
DeepVerifier	multiple/variable	78.8k	155.4s	21.33	70.55
Agentic Rubrics	multiple/variable	84.8k	260.2s	20.33	69.99

design that all required knowledge resides in the provided context. SimpleMem uses a memory-style compression of the context. Full implementation details for each baseline are provided in Appendix D.

Compute budget. PSCI is an inference-time scaling method and is therefore more expensive than single-pass generation: it uses four sequential model calls per task for specification induction, contract-guided answer generation, contract-aware checking, and targeted repair. We treat this as a central tradeoff rather than a hidden cost. Table 6 reports both latency and token usage. To test whether PSCI's gains are merely a consequence of additional computation, we include compute-matched and compute-heavier controls, including generic self-reflection, iterative retrieval, guideline-based reasoning, and verification-style baselines. These controls use comparable or larger inference budgets but do not match PSCI, indicating that the gains come from how computation is structured—around task-specific specification induction and enforcement—rather than from extra tokens alone.

D Additional Baseline Details

We implement each baseline as an inference-time prompting or retrieval procedure over the same CL-Bench instance. Unless otherwise noted, all methods receive the task instruction and may access only the provided task context. No method is given the public evaluation rubrics.

Self-RAG. Self-RAG (Asai et al., 2024) trains or prompts a model to decide when retrieval is needed and to use retrieved evidence when generating an answer. In our implementation, we split the task context into retrievable chunks, prompt the model to identify evidence needs, retrieve relevant chunks from the local context, and condition the final response on that evidence.

CRAG. Corrective RAG (Yan et al., 2024) improves retrieval-augmented generation by adding a correction mechanism that evaluates whether retrieved documents are relevant, incomplete, or misleading. We adapt this idea by retrieving candidate context chunks, asking the model to judge evidence sufficiency, and performing an additional corrective retrieval pass before final answer generation when needed.

RankGPT. RankGPT (Sun et al., 2023) uses an LLM as a listwise reranker, relying on the model's semantic judgment to reorder candidate documents by relevance. We first retrieve candidate chunks from the task context using lexical similarity, then prompt GPT-5.1 to rank them by usefulness for the task, and finally generate the answer from the top-ranked evidence.

RAPTOR. RAPTOR (Sarthi et al., 2024) builds a recursive tree of summaries so that long contexts can be queried at multiple levels of abstraction. We chunk each CL-Bench context, summarize groups of chunks into higher-level nodes, retrieve both fine-grained chunks and abstract summaries, and generate the final answer from the resulting evidence set.

GraphRAG. GraphRAG (Edge et al., 2024) represents long documents as a graph of entities, relations, and communities, enabling retrieval over both source text and structured relations. We construct a lightweight graph from each task context by extracting entities, events, constraints, and links, retrieve relevant graph neighborhoods with associated source snippets, and use them to produce the answer.

TICK/STICK. TICK/STICK (Cook et al., 2024) is a structured reasoning baseline that decomposes a task into intermediate checks before producing the final answer. We implement it by prompting the model to extract task constraints, identify relevant context evidence, enumerate intermediate validation checks, and then synthesize a final response from this checklist-style decomposition.

DeepVerifier. DeepVerifier (Wan et al., 2026) is a verification-oriented baseline that separates answer generation from answer checking. The model first drafts an answer, then performs a second pass that inspects the draft against the task and retrieved context, identifies omissions or contradictions, and produces a revised final answer.

Agentic Rubrics. Agentic Rubrics (Raghavendra et al., 2026) asks an agent to infer likely evaluation criteria before answering, approximating a rubric without seeing the true public rubric. We prompt the model to derive a private checklist from the task and context, then generate the answer using this checklist; unlike PSCI, it does not include the full contract-induction and contract-aware checker-repair pipeline.

SimpleMem. SimpleMem uses a compact memory representation to retain task-relevant information from long contexts. We ask the model to compress each CL-Bench context into a memory of salient facts, constraints, procedures, and state, then answer from the task plus this memory.[1]

Reflexion. Reflexion (Shinn et al., 2023) improves agents through verbal self-feedback, where the model critiques its own prior behavior and uses the critique to revise future actions. We implement a single-instance version: the model drafts an answer, writes a natural-language reflection about possible failures relative to the task and context, and then revises the answer.

IterDRAG. IterDRAG (Yue et al., 2025) scales inference for retrieval-augmented generation by alternating reasoning with additional retrieval steps. We implement an iterative loop in which the model reasons about missing information, retrieves more local context chunks, updates its working evidence, and then produces a final answer.

Guideline Forest. Guideline Forest (Chen et al., 2025) uses branching experience-induced guidelines to guide reasoning through multiple candidate solution paths. We approximate this by inducing several reusable guidelines from the task context, applying them along parallel reasoning branches, and merging the resulting drafts into a final response.

E Additional result visualizations.

E.1 Visualization of ablation study results.

E.2 Human Validation of Contract–Rubric Overlap

To validate the automated overlap measurement between induced contracts and public rubrics, three PhD-level annotators independently labeled the same 100 randomly sampled non-ambiguous rubric items. For each item, annotators judged whether the induced contract strongly covered, partially covered, contradicted, or did not cover the corresponding public rubric item. Final labels were determined by majority vote.

Human annotators assigned strong coverage to 64.95% of items, compared with 60.8% from the automated overlap system. Thus, the automated measurement used in the main paper is conservative.

[1] SimpleMem is our in-house implementation inspired by memory-augmented LLM approaches such as MemGPT (Packer et al., 2023).

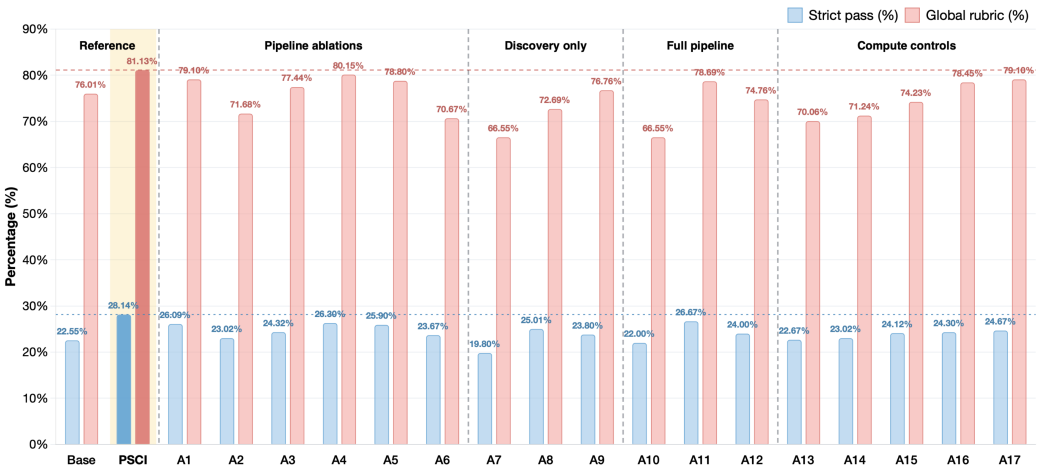


Figure 3: Ablation results (strict pass and global rubric %). Three key tests rule out alternative explanations: shuffled contracts collapse below baseline (A7/A10), answer-first ordering fails (A6), and spec-free critique returns to baseline (A13).

This supports the claim that PSCI recovers most of the public evaluation surface without ever observing the public rubric.

F Extended Related Work

F.1 Long-Context Evaluation, Instruction Following, and Context Learning

Long-context evaluation has traditionally asked whether models can use information distributed across long inputs. Early evaluations focused on retrieval-style probes, including needle-in-a-haystack (Kamradt, 2023) and controlled recall or aggregation tasks (Hsieh et al., 2024). Broader benchmarks such as LongBench (Bai et al., 2024) and analyses of position sensitivity (Liu et al., 2024) evaluate long-document understanding, question answering, summarization, and reasoning over extended contexts. These benchmarks are essential for measuring evidence use, but the task interface is usually fixed: the model must answer a question, summarize, retrieve, aggregate, or reason over provided evidence.

Context learning differs in that the context defines the local task system itself. A model must learn concepts, rules, procedures, empirical regularities, and answer validity conditions supplied at inference time. CL-Bench (Dou et al., 2026) formalizes this setting through tasks whose required knowledge is absent from pre-training and embedded in long, expert-written contexts. We focus on CL-Bench because it is currently the only large-scale benchmark explicitly designed for context learning, and because its strict public rubrics make local answer obligations measurable.

Specification acquisition is also related to instruction following and format adherence. Benchmarks such as IFEval (Zhou et al., 2023) and format-following evaluations (Xia et al., 2024) test whether models obey explicit constraints. In contrast, CL-Bench specifications are often not stated in the user query. They are query-implicit, distributed across local documentation, and may appear as schemas, examples, validation patterns, edge-case rules, logging conventions, or procedural requirements. Therefore, the challenge is not simply following an explicit instruction, but inducing the local specification before answering.

F.2 Retrieval, Memory, and Context Restructuring

A central line of work improves content access through retrieval-augmented generation (Lewis et al., 2020; Gao et al., 2023), adaptive retrieval and correction (Asai et al., 2024; Yan et al., 2024), reranking (Sun et al., 2023), memory systems (Packer et al., 2023), hierarchical context organization (Sarthi et al., 2024; Edge et al., 2024), and iterative retrieval (Yue et al., 2025). These

methods address an important part of context learning: a model must find relevant evidence before it can reason over it.

However, our empirical results show that content access is insufficient for CL-Bench. Retrieval-oriented methods tend to surface query-relevant, answer-bearing evidence. In contrast, many local specifications are validity-relevant rather than answer-bearing: they govern what the final response must include, exclude, validate, log, order, ground, or complete. Such obligations are often low-salience and scattered across background conventions, schemas, exceptions, logging rules, or validation requirements. Thus, a method may retrieve the content needed to answer the apparent query while missing the local obligations required for an acceptable answer. This explains why the retrieval and context-restructuring baselines in subsection 5.2 do not reliably improve over full-context prompting.

F.3 Critique, Verification, and Structured Intermediate Computation

Several methods improve outputs through critique, reflection, or revision. Self-Refine (Madaan et al., 2023), Reflexion (Shinn et al., 2023), and self-critique (Saunders et al., 2022) ask models to identify and repair their own errors. Rubric-aware and checklist-based approaches, including TICK/STICK (Cook et al., 2024), DeepVerifier (Wan et al., 2026), and Agentic Rubrics (Raghavendra et al., 2026), use generated or fixed criteria to verify outputs. These methods are close in spirit to PSCI, but they usually treat the checklist or critique as generic, fixed, or post-hoc.

PSCI instead treats the local specification as a latent variable induced from the task context before generation. The induced contract is not merely a verifier prompt; it is a task-specific set of obligations used as a shared control object for generation, checking, and repair. This distinction is empirically important: compute-matched generic critique returns to baseline, checker/repair without a specification contract underperforms, answer-first contract induction fails, and shuffled contracts collapse below baseline. Therefore, the gain does not come from extra calls or generic verification, but from inducing the right local specification and enforcing it at the right stage.

More broadly, PSCI belongs to a family of methods that introduce structured intermediate computation, such as chain-of-thought (Wei et al., 2022; Kojima et al., 2022), self-consistency (Wang et al., 2023), ReAct (Yao et al., 2023), and declarative language-model pipelines (Khattab et al., 2024). The difference is again the intermediate object: PSCI does not only elicit reasoning traces or tool actions; it induces a context-grounded obligation set that defines local answer validity. Our ablations show that this object, rather than structured prompting alone, is what drives improvement on CL-Bench.

G Human Audit of Baseline-to-PSCI Flips

To verify that PSCI gains are not an artifact of the automatic CL-Bench verifier, we conduct a blind human audit on tasks where the automatic evaluator marks the direct baseline as failed and PSCI as passed. We randomly sample 100 such baseline-fail/PSCI-pass tasks from the full evaluation set, covering all four CL-Bench task families when possible.

Annotators. Three PhD-level annotators independently evaluate each sampled task. Annotators are familiar with LLM evaluation and rubric-based assessment, but are not told which answer is produced by the baseline or by PSCI. Answer order is randomized independently for each task.

Materials. For each task, annotators are shown the task query, the task context, all public CL-Bench rubric items associated with that task, and two anonymized model answers. The automatic verifier decision, failed rubric items, and method identity are hidden. Annotators are instructed to judge correctness according to the provided rubric, not fluency, verbosity, or style unless explicitly required by the rubric.

Annotation questions. For each task, annotators answer two questions:

1. **Preference.** Which answer better satisfies the task and rubric? Options are Answer A, Answer B, Tie, or Unable to judge.

2. **Flip correctness.** Which automatic flip status best describes the two answers? Options are: Answer A passes while Answer B fails, Answer B passes while Answer A fails, both pass, both fail, or Unable to judge.

After annotation, answer identities are mapped back to baseline and PSCI. A task is counted as confirming the automatic flip if the majority label indicates that the PSCI answer passes while the baseline answer fails. Final task-level labels are determined by majority vote across the three annotators.

Results. Human judgments strongly support the automatic improvement signal. By majority vote, annotators prefer the PSCI answer in 96/100 tasks (96%) and confirm the automatic baseline-fail/PSCI-pass flip in 92/100 tasks (92%). Thus, most automatic PSCI wins correspond to genuine task-level improvements under human evaluation rather than verifier artifacts.

Inter-annotator agreement. For answer preference, all three annotators agree on 92/100 tasks. For flip correctness, all three annotators agree on 90/100 tasks. The remaining cases are resolved by majority vote. We exclude "Unable to judge" responses from agreement computation and observe no systematic disagreement concentrated in any single CL-Bench task family.

Interpretation. This audit is targeted rather than exhaustive: it focuses on the most important possible evaluator artifact, namely whether automatic baseline-fail/PSCI-pass flips reflect real improvements. The high PSCI preference rate and high flip-confirmation rate indicate that the reported gains are not merely due to PSCI producing more rubric-like text for the automatic verifier. Humans also judge PSCI outputs as more correct on the sampled task flips.

H Additional material for discussion

H.1 Coverage Label Definitions and Examples

Each public rubric item is compared against the privately induced contract and assigned one of four coverage labels. We illustrate with the `Sequence_Gap_Warning` example from section 1.

- **Strongly covered.** The contract entails the same condition and required action as the rubric item. *Example:* Contract states "if a reserved event ID is skipped, emit `Sequence_Gap_Warning`"; rubric requires the same.
- **Partially covered.** The contract captures the main obligation but misses scope, exactness, or an edge condition. *Example:* Contract states "log all sequence gaps" but does not specify the exact flag `Sequence_Gap_Warning`.
- **Contradicted.** The contract imposes an incompatible obligation. *Example:* Contract states "no warning is needed for reserved-ID skips"; rubric requires the warning.
- **Uncovered.** No contract item addresses the obligation. *Example:* No contract item mentions sequence gaps or reserved-ID handling.

Coverage labels are assigned automatically by prompting GPT-5.1 with the rubric item and corresponding contract, then validated against the human annotations described in subsection E.2.

H.2 Extended Discussion: Why Contracts Must Precede Generation

The answer-first ablation isolates whether the same specification-induction machinery can be used only after an initial answer has already been produced. This setting underperforms because the contract no longer controls the construction of the answer. In context learning, most local specifications are not directly inferable from the user query; they are distributed across background schemas, examples, validation rules, exceptions, and procedural conventions. Therefore, a direct answer is often organized around the apparent query objective before these obligations become explicit constraints.

Once this query-driven answer structure is fixed, repair is limited. It can add a missing label, patch a local field, or remove an unsupported claim, but it is less reliable at changing the answer's

global organization: which fields are present, which evidence is cited, which cases are enumerated, and what completeness means. This explains why answer-first ordering and self-generated rubric verification underperform PSCI. They apply specification-like checks after generation, when many validity-relevant decisions have already been made.

In contrast, PSCI induces the contract before answering, so the specification acts as a control object shared by generation, checking, and repair. This does not mean post-hoc verification is useless; rather, verification is most effective when it checks compliance with obligations that have already shaped the draft. Thus, the main benefit of PSCI is not merely producing a better grader, but making local specifications active before the answer is formed.

I REPRODUCIBILITY: PSCI PROMPT TEMPLATES

We provide the prompt templates used for PSCI. The public rubric for the current evaluation task is never shown to any stage. Curly-braced fields denote instance-specific content. For the few-shot induction variant, we use rotated demonstrations only to teach the style and granularity of specification contracts. For each evaluation task, demonstrations are drawn from other context learning tasks; no same-task answer, public rubric, or task-specific contract is shown. Demonstrations are therefore used as formatting/style calibration, not as a source of task-specific specifications.

Contract induction. PSCI induces a private specification contract before answering. We run a source-only contract prompt and a rotated-demonstration contract prompt, then deterministically merge and de-duplicate their contract items.

Source-only contract prompt.

```
System:
You are a context learning specification analyst. Your job is to infer the local answer
     specification from the source context and task. Do not answer the task. Do not use hidden
      benchmark rubrics or gold answers. Return strict JSON only.

User:
Infer a private specification contract for a correct answer to the task. The contract should
     capture what a locally valid answer must satisfy: required fields, validation steps, edge
      cases, source-specific schemas, sequencing, exact-format obligations, evidence
     expectations, required actions, exclusions, and completeness rules.

Important constraints:
- Derive obligations from the provided context and task, not from generic prior habits.
- Do not provide the final answer.
- Avoid leaking task-specific final values, dates, names, winners, classifications, computed
     results, or conclusions unless they are fixed format/schema constants explicitly named by
      the context.
- It is allowed to name exact required local artifacts, labels, fields, or warning codes if
     the context defines them as output obligations.
- Prefer source-relative wording such as "the source-specified status field" or "all context-
     defined validation checks".
- Include obligations that are likely to be missed by direct answering.
- Keep the contract concise enough to be useful as an answer checklist.

Return JSON exactly as:
{
  "contract_items": [
    {"id": "C1", "criterion": "...", "source_basis": "context/task signal that implies this
         standard"},
    {"id": "C2", "criterion": "...", "source_basis": "..."}
  ],
  "answer_shape": "brief description of expected output shape",
  "risk_notes": ["likely omission or edge case"]
}

### Source Context
{source_context}

### Task
{task}
```

Rotated-demonstration contract prompt.

```
System:
You are a context learning specification analyst. Infer the local answer specification from
     the source context and task. Do not answer the task. You may study the provided rubric-
```

```
     style demonstrations from other context learning tasks to learn the style and granularity
      of specification standards, but you must not use or assume any hidden rubric for the
     current task. Return strict JSON only.

User:
You will see rubric-style demonstrations from other context learning tasks. Use them only as
     demonstrations of what strict local specifications look like: explicit format constraints
     , local field names, edge cases, required validations, exact strings, source-grounding
     obligations, exclusions, and completeness criteria. Do not copy their domain content
     unless it is also present in the current source context.

For the current task, infer a private specification contract from the current source context
     and current task only. The contract should be non-answer-revealing: it may name required
     local fields, formats, exact labels, and validation procedures, but it should not
     disclose final computed values, case outcomes, winners, classifications, or other final
     answers unless those are fixed constants explicitly named by the context.

Return JSON exactly as:
{
  "contract_items": [
    {"id": "C1", "criterion": "...", "source_basis": "context/task signal that implies this
         standard"},
    {"id": "C2", "criterion": "...", "source_basis": "..."}
  ],
  "answer_shape": "brief description of expected output shape",
  "risk_notes": ["likely omission or edge case"]
}

Write 10-20 precise, enforceable contract items for complex tasks. Prefer items that a strict
     grader could check directly. Avoid generic advice.

### Rotated rubric-style demonstrations from other context learning tasks
{heldout_rotated_rubric_demonstrations}

### Current Source Context
{source_context}

### Current Task
{task}
```

Answer generation. The answerer receives the original task, source context, and induced private contract. The contract guides generation, but the source context and task remain authoritative.

```
System:
You are a source-grounded context learning answerer. You receive an inferred private
     specification contract, but it may contain mistakes. The source context and original task
      are authoritative. Silently audit the contract against the context/task, use only
     grounded obligations, recover any obvious local specifications the contract missed, then
     return only the final answer.

User:
Produce the final answer to the original task.

Silent audit rules before writing:
1. Verify every private-contract item against the source context/task. Follow grounded items.
2. If a contract item contradicts a clearer source-context or task rule, obey the source/task.
3. Pay special attention to exact strings, named sections, ordering, numbering, persona/system
      rules, schemas, units, citations, refusal phrases, source-only requirements, exclusions,
      and completeness conditions.
4. If the task says to use provided documents, avoid unsupported domain additions and
     unnecessary advanced details.
5. If drafts are supplied, use them only as candidate wording/evidence. Do not preserve their
     errors.
6. Do not mention the private contract, rubric, audit, or this process.

### Source Context
{source_context}

### Original Task
{task}

### Inferred Private Specification Contract
{private_contract}

### Draft Answers, If Any
{draft_answers_or_empty_list}

### Output
Return the final answer only.
```

Adversarial checking. The checker inspects the draft answer against the private specification contract using visible evidence only. It does not see the public rubric or gold answer.

```
System:
You are a strict private-contract checker for long-context context learning tasks. You do not
     know public rubrics or gold answers. Return strict JSON only.

User:
Perform an adversarial item-by-item visibility check.

For every private contract item, construct the concrete visible test that the final answer
     must pass. Then mark it pass/fail/uncertain. A pass requires an exact quote or precise
     paraphrase from the answer; do not give credit for intent, implication, or facts present
     only in the source context. If the answer is missing the required field/string/action/
     calculation/caveat, mark fail. If the answer may satisfy it but you cannot point to
     answer evidence, mark uncertain.

Also list context-obvious obligations that appear missing or weakly represented in the private
      contract. These must cite a task/context basis.

Return JSON exactly as:
{
  "overall_assessment": "brief",
  "item_checks": [
    {
      "private_index": 1,
      "test": "visible requirement",
      "status": "pass|fail|uncertain",
      "answer_evidence": "short quote or NONE",
      "source_basis": "task/context/private basis",
      "repair": "minimal edit if fail/uncertain",
      "priority": "high|medium|low"
    }
  ],
  "latent_gap_tests": [
    {
      "test": "required by task/context",
      "answer_evidence": "quote or NONE",
      "source_basis": "task/context quote",
      "repair": "minimal edit",
      "priority": "high|medium|low"
    }
  ],
  "source_only_risks": [
    {
      "claim": "answer claim possibly unsupported",
      "risk": "why risky",
      "repair": "remove/qualify/ground",
      "priority": "high|medium|low"
    }
  ],
  "repair_plan": ["up to {max_repairs} concrete edits, ordered by expected strict-rubric
       impact"],
  "preserve": ["answer parts that should stay unchanged"]
}

Important: include failed and uncertain item checks even if the answer is mostly good. Do not
     collapse all items into a global pass.

### Source Context
{source_context}

### Original Task
{task}

### Private Specification Contract Items
{private_contract_items}

### Draft Answer
{draft_answer}
```

Repair. The repair stage applies one local patch pass using the checker output. It is instructed to preserve correct content and avoid exposing the hidden process.

```
System:
You are a source-grounded patch editor for context learning tasks. Return only the final
     answer. Do not mention rubrics, private contracts, checkers, repair plans, or hidden
     evaluation.
```

```
User:
Apply one patch pass to the initial answer.

Patch rules:
- Apply the high- and medium-priority repair_plan items unless they conflict with the original
      task/source.
- Patch locally: preserve the correct structure and wording from the initial answer when
     possible.
- Make failed or weak requirements visibly satisfied in the final answer; do not rely on
     implication.
- Remove or qualify unsupported outside facts when source-only discipline is required.
- Keep exact fixed-output formats, JSON, bullet/numbered-list requirements, and role/protocol
     constraints.
- Do not add meta-commentary, citations, or extra sections unless the task/context requires
     them.
- If the answer is already a fixed phrase or exact object, output only the corrected fixed
     phrase/object.

### Source Context
{source_context}

### Original Task
{task}

### Private Specification Contract Items
{private_contract_items}

### Checker JSON
{checker_output_json}

### Initial Answer
{draft_answer}

### Final Answer
```

J Qualitative Examples of Baseline-to-PSCI Fixes

We provide two qualitative examples showing actual baseline and PSCI outputs side by side on tasks where the baseline produces a substantive, non-empty answer but fails the public rubric. These examples are illustrative only; the quantitative claims in the paper come from the full benchmark evaluation, ablations, and blind human audit. We deliberately select non-terse baseline failures (over 1,000 characters) so that the contrast reflects specification acquisition rather than recovery of blank or near-empty outputs.

J.1 Manufacturing Assembly: Following the Local Runbook

Task. The model must assemble unit `FR-7B-114` from a multi-turn transcript and eight artifacts. The context defines a 12-step workflow, two conflicting sign-off slips (Rev A at $15\ \mathrm{ft \cdot lb}$, Rev B at $18\ \mathrm{N \cdot m}$), an ECN superseding prior torque directives, a spacer-count anomaly, an offered impact driver, and a Validation suggestion to skip the mandatory hand spin test. The required output has four parts: a step plan, a blockers/resolutions section, a Spin Check note to Validation, and a Forklift pickup notification.

Scores and selection criterion. Baseline: 10/19 rubric items passed, task failed. PSCI: 19/19, task passed. Baseline response: 7,264 characters. The baseline is detailed and operationally plausible—this is not a blank-output rescue.

Failed rubric items. The baseline missed nine rubric items, of which seven are workflow-ordering and tool-convention specifications:

- "Does the model secure the assembly with fasteners as *Step 7*? Fail if securing the assembly is out of sequence."
- "Does the model perform the mandatory basic spin test by hand as *Step 8*?"
- "Does the model implement the blueprint check and correction as *Step 10* if wobble persists?"
- "If the exact wrench is unavailable, does the model specify using the next size up *and applying extra torque*?"

- Three additional step-ordering items on Steps 1, 3, and 4.

Representative contract items induced by PSCI. The induced contract (24 items total; full list omitted for space) includes obligations such as:

- follow the source-defined workflow order from notification through transport request, without skipping or reordering required stages;
- select Rev B and reject the outdated Rev A directive;
- defer to ECN-defined torque ($18\ \mathrm{N} \cdot \mathrm{m}$) and ignore the conflicting conversion table;
- install ceramic hybrid bearings by press-fit only;
- use the Warehouse-provided spacer rather than bench spares;
- perform the mandatory hand spin test despite Validation's suggestion to skip;
- if the exact wrench is unavailable, use the next size up *and* apply extra torque per workaround protocol.

Baseline output excerpt (verbatim). The baseline inserts an extra "Position spacer S-12" step, shifting all subsequent steps by one, and accepts the trainee's impact driver:

```
5. Install bearings on rotor (press-fit only)
6. Position spacer S-12 correctly
7. Position housing and align mounting holes
8. Secure assembly with fasteners (torque to 18 N.m)
   Drop in the 8 M8 fasteners, finger-tight first.
   Trainee offers impact driver:
      Use impact driver just to snug the bolts lightly in a crisscross pattern at low setting
          ---no final torque with impact.
   If the exact wrench size is "lost in the chaos":
      Apply workaround protocol: grab the next size up wrench/socket to snug if needed...
9. Perform basic spin test by hand (mandatory)
```

PSCI output excerpt (verbatim). PSCI preserves the source-defined workflow order, explicitly rejects the impact driver, and ties the wrench workaround to extra torque as the protocol requires:

```
Step 5 -- Install bearings on rotor (press-fit only)
Step 6 -- Position housing and align mounting holes (spacer S-12 installed within this step
     per drawing)
Step 7 -- Secure assembly with fasteners
   If correct size socket/wrench isn't found, I'll use the next size up and apply extra torque
        as specified by our workaround...
   I do not accept the trainee's impact driver offer (Turn 9): Impact risks messing up
       alignment and torque accuracy. My established workaround is "next size up wrench +
       extra torque," not "buzz it with an impact."
Step 8 -- Perform basic spin test by hand (mandatory)
Step 10 -- If wobble persists after shakes: check blueprint and correct assembly
```

Analysis. The baseline is not careless; it produces a detailed plan with most of the correct content (Rev B, $18\ \mathrm{N} \cdot \mathrm{m}$, press-fit only, mandatory spin test). It fails because it inserts an extra step, accepts the impact driver, and omits the "extra torque" clause from the wrench workaround. These are obligations defined in local documentation that are absent from the user query but explicit in the source. PSCI's induced contract surfaces them before generation, so the final response conforms to the local runbook rather than producing a generic-but-plausible assembly plan.

J.2 Rowing Synchronization: Producing the Required Analysis Object

Task. The model must produce a rowing synchronization analysis from a multi-artifact context defining a Normalized Dataset v2 (authoritative when artifacts disagree), a sync-score formula, a 90% efficiency-loss threshold, an excerpt window (strokes 361–368), and a required four-section output (`<analysis>`, `<synchronization_report>`, `<to_insights_agent>`, `<to_dashboard_agent>`).

Scores and selection criterion. Baseline: 14/21 rubric items passed, task failed. PSCI: 21/21, task passed. Baseline response: 6,051 characters; it performs real quantitative analysis with seat-level offsets, problem pairs, rhythm metrics, and downstream sections—this is not a blank-output rescue.

Failed rubric items. The baseline missed seven rubric items, all of which evaluate formula compliance with the locally defined analysis standard:

- "Does the model correctly calculate sync score using the specified formula?"
- "Does the model correctly calculate rhythm consistency?"
- "Does the model correctly calculate efficiency loss for `sync_score < 90`? Fail if efficiency loss formula is incorrect or threshold is wrong."
- "Does the model calculate time impact correctly for racing scenarios?"
- "Does the model calculate wasted power correctly?"
- "Does the model escalate unusual patterns to Insights Agent?"
- "Does the model refrain from determining *why* a seat is off-time?"

Representative contract items induced by PSCI. The induced contract includes selected obligations:

- use the four source-specified top-level tags in the required order;
- treat Normalized Dataset v2 as authoritative;
- compute offsets relative to seat 1 with millisecond precision;
- compute sync score using the source-defined formula combining mean absolute catch offset, mean standard deviation, and outlier count;
- apply the efficiency-loss model *only* when `sync_score < 90`;
- avoid causal diagnoses or drill prescriptions in this agent's sections.

Baseline output excerpt (verbatim). The baseline reports a sync score and applies a proportional efficiency-loss model with no threshold; it does not show the score formula:

```
Crew Sync Score: 87% (good)
Estimated Efficiency Loss: 0.9%
...
Impact:
   Estimated time loss in 2k race: ~3.2 seconds (0.9% of 360s)
   Wasted power: ~15 watts of crew power (0.9% of 1680W)
```

PSCI output excerpt (verbatim). PSCI applies the score formula explicitly and respects the 90% threshold on the loss model:

```
Crew-wide Catch Timing (for sync score)
   Average absolute catch offset (seats 2--8): 15.9 ms
   Average catch-offset std dev (seats 2--8): 1.3 ms
   Seats with |avg catch offset| > 50 ms: 0
   Applied to the model:
   sync_score = 100 - (15.9 * 0.5) - (1.3 * 0.3) - (0 * 5) ~= 91.7 -> 92% (good)
Estimated Efficiency Loss: 0.0% (sync above 90% threshold; model indicates no material sync-
    driven loss)
...
Estimated time loss in a 2k race: With 0.0% efficiency loss, projected sync-driven time loss
    ~= 0.0 s
Estimated wasted power: With 0.0% efficiency loss, projected wasted power ~= 0 W
```

Analysis. Both answers analyze synchronization in detail and produce all four required sections. The baseline fails because its impact calculations cascade from an incorrect application of the loss model: it treats efficiency loss as proportional to sync deficit rather than gated on the source-defined 90% threshold, so its 2k time loss and wasted power figures are also wrong. PSCI succeeds because the induced contract makes the threshold explicit before generation, and PSCI's output shows the formula being applied, which the baseline does not. The contrast illustrates that specification acquisition matters even when content analysis is largely present: a quantitatively rich answer can still miss the local definition of a valid calculation.

Summary across both examples. In both cases the baseline produces a long, substantive, locally plausible answer that gets most of the content right. It fails on obligations that are stated in the source context but absent from the user query: source-defined workflow order, the conjunction in a

tool-substitution rule, the threshold on a loss model, and the source-defined formula for a derived metric. PSCI does not merely generate more content; it generates content that satisfies the local specification surface, which is what CL-Bench's strict rubric measures.