RECAP: Regression Evaluation for Continual Adaptation of Prompts

Anonymous ACL submission

Abstract

Production agentic systems routinely face evolving constraints and must comply from the very next interaction. Scenarios like a tool-call notification changing a compliance threshold or a policy update adding disclosure requirements fit this criteria, having close to no room for errors in production. This *proactive* adaptation setting is common in deployment, but absent from current benchmarks, which assume either static constraint sets or reactive protocols with evaluation feedback. We introduce RECAP, a benchmark that measures continual-learning phenomena (forgetting, regression, forward transfer) at the constraint level under a strictly proactive adapt-then-test protocol: prompt optimization methods receive only the constraint specification and must generalize before seeing any test data. Evaluating six methods across four LLMs and three schedules with evolving constraints, we find that these methods show no significant improvement in performance, even after incurring a higher latency. These methods, designed for offline or reactive settings, are inadequate for the proactive paradigm. Our work emphasizes the growing need for designing proactive prompt adaptation methods, where the models must remain robust to evolving needs in deployment.

1 Introduction

Agentic systems in production operate under constraints that evolve continuously: a tool-call response tightens a length limit, a policy update adds a disclosure requirement, or a user preference changes the expected tone. The system must satisfy new constraints immediately while continuing to respect all prior ones. Further, these constraints are often not centrally documented: they accumulate from individual tool-call responses, user interactions, and personalization settings across many users (Ye et al., 2026), making it unrealistic to collect the full active set and optimize jointly each time one changes (Banerjee et al., 2025).

Aligned with the EMNLP 2026 theme of rethinking evaluation beyond static benchmarks, we argue that progress in deployed agents must account for longitudinal behavior under evolving specifications. We focus on *proactive* adaptation, where the system receives only a constraint specification and must comply before seeing any real test data or feedback, with minimal latency overhead. The setting is ubiquitous in deployment, yet entirely absent from current evaluation paradigms. Instruction-following benchmarks (Zhou et al., 2023; Guo et al., 2026; Jiang et al., 2024; Qin et al., 2024) present a fixed constraint set and measure single-shot success, with no mechanism for constraints to evolve over time. Prompt optimization methods (Yuksekgonul et al., 2025; Yang et al., 2024; Khattab et al., 2024; Opsahl-Ong et al., 2024) do address iterative improvement, but assume access to representative evaluation data and multiple rounds of feedback, which are not available in the proactive setting. Reactive protocols like ACE (Zhang et al., 2026) allow iterative debugging on observed failures, but again require test-time feedback to drive adaptation. The natural framework for evolving constraints is Continual learning (CL), which studies how systems adapt to new tasks without forgetting prior ones (De Lange et al., 2021; Shi et al., 2024; Wu et al., 2024). However, existing CL operate on model weights through regularization (Kirkpatrick et al., 2016), replay (Lopez-Paz and Ranzato, 2017), or prompt embeddings (Wang et al., 2022b,a) and do not address prompt-level text constraints where the model weights are frozen and adaptation must happen entirely through the input. None evaluates the proactive case where methods must generalize from specification alone.

We present **RECAP**: **R**egression **E**valuation for **C**ontinual **A**daptation of **P**rompts, a benchmark that extends the evaluation of constraint satisfaction beyond static settings. RECAP performs continual evaluation on schedules subject to evolving

constraints with *add*, *edit*, and *delete* operations. This enables a rigorous evaluation of recent prompt adaptation methods under a proactive protocol. We summarize our contributions below:

1. We design a **constraint-level CL benchmark** that converts static instruction-following datasets into temporal evaluation streams via typed operations under a proactive protocol, where methods receive only the constraint specification and must generalize immediately (§2).
2. We develop a **decomposed metric suite** measuring constraint satisfaction, regression, edit uptake (*Are modified constraints adopted?*), unlearning fidelity (*Are deleted constraints forgotten?*), and efficiency (§2).
3. We provide **empirical evidence** that existing prompt adaptation methods are structurally inadequate with the proactive paradigm, discussing failure modes to guide future work (§3 and §4).

2 Methodology

Source Data: In the proactive setting, constraints evolve independently of the base task. This requires source data where constraints are separated from base instructions. We build on RECAST-30K (Guo et al., 2026): which is based on Tulu 3 Persona IF (Lambert et al., 2025). The data contains base instructions (e.g., '*Write a cover letter for a data-science role*') paired with one or more constraints (e.g., '*keep under 200 words*' or '*mention Python at least 3 times*'). Constraints are grouped into semantic *types* (Length, Keyword, Format, Tone, etc.), each with one or more concrete values (e.g. maximum length can be 200 or 300 words). We have 21 constraints in total, out of which 8 have deterministic rule-based validators while 13 require LLM judgement (Appendix B).

Operations and Shadow Evaluation: RECAP transforms this static dataset of instructions and constraints into a temporal evaluation stream by defining *schedules* of evolving constraints. At each step in a schedule, we apply one out of three operations: 1) ADD introduces a new constraint type, 2) EDIT replaces the concrete value of an existing type, and 3) DELETE removes a constraint type entirely. A *schedule* consists of a sequence of operations over 15-20 steps (Appendix C), controlling which constraints are introduced, modified, or removed, and in what order. A key question is whether adapting to a new or modified constraint causes interference with previously satisfied ones. To measure this after edits and deletions, we retain the old constraint as a *shadow* in the evaluation set at all subsequent steps: responses continue to be checked against the replaced or removed specification even though the LLM no longer sees it. This enables tracking of edit persistence (*does the model revert to old behavior over time?*) and unlearning rebound (*does a deleted constraint resurface?*).

RECAP Protocol: We adopt the adapt-then-test protocol from CL evaluation (Lopez-Paz and Ranzato, 2017; Chaudhry et al., 2018, 2019; De Lange et al., 2021). At each step, the method first adapts to the new constraint operation, then is evaluated on all active constraints (see Figure 1, pseudocode is in Appendix A). Adaptation is *proactive*: `adapt()` receives only the constraint specification (e.g., "edit Length: Keep under 500 words") but no test prompts and no feedback from evaluation. Methods may use internal self-play during adaptation (generating and judging synthetic responses), but they never observe real test data or evaluation results from prior steps. The no-adaptation baseline (Base LLM) skips adaptation entirely and receives only the current active constraints appended to each user prompt at test time, making it a pure test of the LLM's instruction-following ability.

Metrics: Our primary metric is $\overline{\text{sat}}$: Mean constraint satisfaction rate across all types and steps (formula in Appendix D). However, $\overline{\text{sat}}$ by itself can mask important dynamics. A method might maintain mean satisfaction while silently regressing on prior constraints – We detect this with *peak forgetting* (Chaudhry et al., 2018) (the maximum drop from a type's prior peak) and *collateral damage* (mean drop in non-targeted types after an operation). For edits, we define *edit switch*: the fraction of samples satisfying the new specification but not the old. For deletions, we ask whether the model appropriately stops satisfying a removed constraint: *Unlearning Fidelity* measures how quickly satisfaction reverts to its unconstrained default rate. We also report latency (Appendix E).

Methods: We evaluate 6 prompt adaptation methods, spanning few-shot (ICL), memory-based (Dynamic Cheatsheet (Suzgun et al., 2026)), and optimization-based (ACE, GEPA, MIPROv2 (Opsahl-Ong et al., 2024)) paradigms. The optimization methods use *self-play* during `adapt()`: the LLM generates a response to a synthetic prompt, then a second call judges whether the target constraint is satisfied. They differ in search strategy: ACE uses a generate–evaluate–reflect–

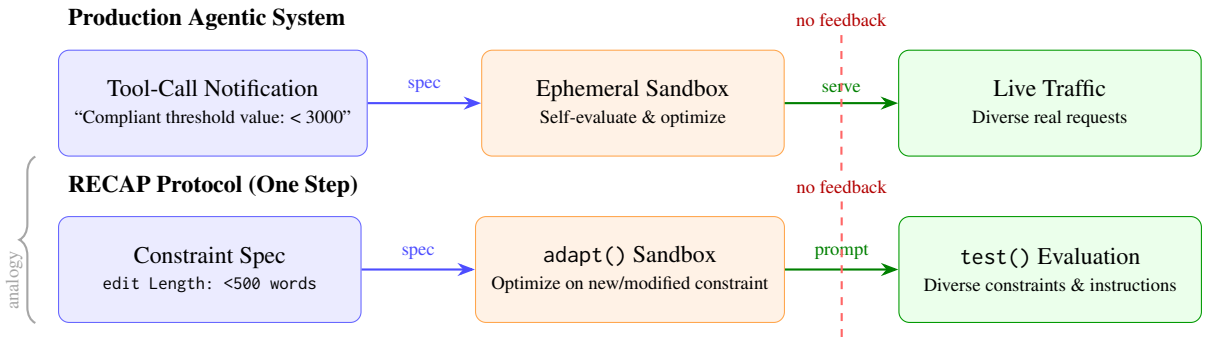

Figure 1: The RECAP protocol mirrors production deployment: a constraint specification arrives, the method adapts without access to test data, then is evaluated on all active constraints. This repeats over a multi-step schedule.

curate pipeline (4 LLM calls/step); GEPA evolves a population via mutation and fitness selection (17 calls/step); MIPROv2 proposes diverse candidates informed by a score history (∼11 calls/step). To mimic dynamic realistic settings, all methods are asked to optimize for the single new constraint at each step without access to other active constraints.

3 Experimental Setup

We show aggregated results on 50 base prompts taken from RECAST using 3 schedules, each designed to assess a different aspect of continual adaptation. **Interleaved-20** mixes 11 adds, 5 edits, and 4 deletes across 20 steps, testing whether methods handle concurrent accumulation and revision. **Clustered-20** applies the same operations but phased (ADD→EDIT→DELETE blocks), testing whether batched operations amplify forgetting. **Rule-Only-15** uses 6 rule-based types with deterministic validators in an interleaved structure, isolating genuine forgetting from any LLM judge noise (more details in Appendix C). We use 4 backbone LLMs: Llama-3.1-8B, Llama-3.3-70B, GPT-OSS-20B, and GPT-OSS-120B (Grattafiori et al., 2024; Agarwal et al., 2025). Claude Sonnet 4.5 (Anthropic, 2025) serves as the LLM judge for qualitative constraints. In total, we have 72 conditions (4 backbones × 6 methods × 3 schedules). Detailed hyperparameters are in Appendix I.

4 Results

Figure 2 reports results for Llama-3.3-70B and GPT-OSS-120B. Results for other models and efficiency comparisons are in Appendix E. The central finding is that no adaptation method achieves a significant improvement over the no-adaptation baseline (Base LLM) on any metric, across any backbone LLM. On GPT-OSS models, adaptation is actively harmful (up to −0.176 mean satisfaction). On Llama models, methods converge within noise of the baseline while consuming upto 1.7× the latency. This points to a structural misalignment with the proactive setting. These findings hold on the rule-only schedule, which uses purely deterministic validators and no LLM judge, confirming they are not artifacts of judge noise (Appendix F). Peak forgetting confirms the pattern: adaptation methods increase forgetting by up to 84% on GPT-OSS models (ACE: 0.330 vs. base 0.179 on 120B) due to context accumulation—as prompt artifacts grow (187→9K chars for ACE), earlier constraint signals are diluted in the context window, causing previously-satisfied types to regress (Figure 3).

Scale dominates method. Model size determines satisfaction scores more than the adaptation strategy. Llama-70B achieves ∼0.60 regardless of method (all 6 fall within 0.592–0.603), while Llama-8B plateaus near 0.49 (Figure 2). The best method on 70B only shows negligible gains over Base LLM (GEPA, +0.008), yet costs 1.5× the latency (189s vs. 126s per step).

Self-play fitness does not transfer. The three optimization methods report high self-play pass rates during adaptation, yet actual test satisfaction shows no gain over Base LLM. Self-play optimization for the newly added constraint on a fixed synthetic task is not robust to regression on existing constraints embedded in the evaluation prompts. The cost is substantial: GEPA's 17 extra calls per step add 64s of latency on 70B for no clear performance gain.

More self-play compute does not help. Scaling the self-play budget for ACE to 3× and 5× *degrades* quality further: satisfaction drops from 0.454 to 0.340 as the playbook grows faster and the model's refusal rates rise 4 times (Appendix J). The failure is structural, not computational.

Method	$\overline{\text{sat}}$	Forg.↓	Coll.↓	Sw.↑	UF↑
Llama 3.3 70B					
Base LLM	0.595	0.235	0.048	0.295	0.642
ICL	0.592	**0.188**	**0.033**	**0.301**	0.650
Dyn. Ch.	0.600	0.204	0.048	0.300	**0.693**
ACE	0.602	0.212	0.035	0.290	0.683
GEPA	**0.603**	0.198	0.048	0.291	0.643
MIPROv2	0.595	0.209	0.034	0.282	0.672
GPT-OSS 120B					
Base LLM	**0.630**	**0.179**	0.053	0.309	0.674
ICL	0.598	0.211	**0.040**	**0.317**	**0.730**
Dyn. Ch.	0.610	0.224	0.050	0.315	0.709
ACE	0.454	0.330	0.050	0.289	0.610
GEPA	0.506	0.268	0.049	0.301	0.648
MIPROv2	0.571	0.222	0.053	0.256	0.635

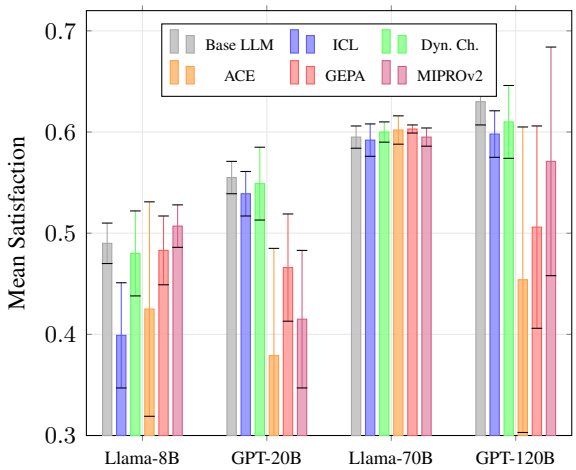

Figure 2: **(Left)** Results on RECAP. $\overline{\text{sat}}$: Mean Satisfaction, Forg.: Peak Forgetting, Coll.: Collateral Damage, Sw.: Edit Switch, UF: Unlearning Fidelity. Bold: best, Underline: second best. Full results with std. dev. in Appendix E. **(Right)** Mean satisfaction ($\pm$1 SD) across all four backbone LLMs.

Figure 3: Examples of observed failures. Each panel shows a single step within a schedule: Blue: All active constraints at that step, Red: Callouts show the model output with the failure cause highlighted.

Closing the information gap yields expensive neutrality. Providing the full constraint set to `adapt()` (not just the delta) recovers most of the deficit on GPT-OSS-120B (0.454 → 0.614, vs. base 0.630), but at 2.8× the token cost for near zero net improvement over Base LLM (Appendix K).

Accumulated artifacts introduce spurious contradictions. Adaptive methods accumulate stale values and may introduce tighter constraints than specified (e.g., adding a sentence-count restriction when optimizing for word-count). When a new constraint conflicts with these outdated or hallucinated values, the LLM mistakenly refuses to generate. On GPT-OSS-120B, ACE's playbook grows to 9K characters over 20 steps, producing a 14% refusal rate. MIPROv2's compiled prompt locks early values and is never revised, causing 31% of responses on GPT-OSS-20B to follow stale directives. Llama models are more robust, treating system prompts as guidance rather than hard directives.

Failure mode taxonomy. We identify 6 systematic failure modes that show structural limitations of the adaptation methods when tested under a proactive protocol (Table 10 in Appendix G); Figure 3 shows 3 of them as examples: (a) A locked prompt causes the model to follow stale directives from earlier steps, (b) Stale context triggers refusal on non-conflicting constraints, and (c) An evolved prompt injects a prefix that violates formatting constraints.

5 Conclusion

We formalize proactive adaptation – how agents handle evolving constraints without feedback or history, and introduce RECAP to evaluate it. Across 72 conditions, no adaptation method beats a baseline that simply appends active constraints to the prompt, despite incurring higher costs. Because real-world constraints are often heterogeneous and non-enumerable, this gap highlights the urgent need for building efficient, regression-free adaptation methods for production agentic systems.

Limitations

We note two limitations. First, all methods are prompt-level (no fine-tuning). Adaptation methods that explore model training specifically for this challenging continual learning setup is one potential future direction for progress in the future. Second, a comparison with reactive methods along with real-time feedback from a specialized oracle would further quantify the proactive gap. We encourage this experiment for future work, noting that self-play evidence presented in this work (near-perfect internal fitness vs. flat test performance) already demonstrates the gap is binding.

Ethical considerations

Our work was approved by the established internal review procedure. We carefully verified the licensing information associated with all the datasets and LLMs used in this work, ensuring that their use was within their intended scope.

Our benchmark uses publicly available instruction-following data (Tulu 3 Persona IF prompts) that does not contain personal, sensitive, or harmful content. No human subjects were involved; all evaluation is automated. We note that proactive adaptation methods, if made effective in future work, could be misused to silently inject constraints that alter system behavior without user awareness (e.g., suppressing certain topics). We believe that benchmarking and understanding these methods transparently is a necessary step toward responsible deployment and appropriate safeguards.

Finally, we note that for ensuring reproducibility, all the code as well as CL evaluation data used in RECAP will be released upon acceptance.

References

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, and 1 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.

Anthropic. 2025. Claude sonnet 4.5. Large Language Model.

Debangshu Banerjee, Tarun Suresh, Shubham Ugare, Sasa Misailovic, and Gagandeep Singh. 2025. CRANE: Reasoning with constrained LLM generation. In *Forty-second International Conference on Machine Learning*.

Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. 2018. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European conference on computer vision (ECCV)*, pages 532–547.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. 2019. Efficient lifelong learning with a-GEM. In *International Conference on Learning Representations*.

Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. 2021. A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Zhengkang Guo, Wenhao Liu, Mingchen Xie, Jingwen Xu, Zisu Huang, Muzhao Tian, Jianhan Xu, Yuanzhe Shen, Qi Qian, Muling Wu, and 1 others. 2026. Recast: Expanding the boundaries of llms' complex instruction following with multi-constraint data. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. 2024. FollowBench: A multi-level fine-grained constraints following benchmark for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4667–4688, Bangkok, Thailand. Association for Computational Linguistics.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2024. Dspy: Compiling declarative language model calls into self-improving pipelines. In *The Twelfth International Conference on Learning Representations*.

James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2016. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114:3521 – 3526.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James Validad Miranda, Alisa Liu, Nouha Dziri, Xinxi Lyu, Yuling Gu, Saumya Malik, Victoria

Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Christopher Wilhelm, Luca Soldaini, and 4 others. 2025. Tulu 3: Pushing frontiers in open language model post-training. In *Second Conference on Language Modeling*.

David Lopez-Paz and Marc'Aurelio Ranzato. 2017. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. 2024. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 9340–9366, Miami, Florida, USA. Association for Computational Linguistics.

Yiwei Qin, Kaiqiang Song, Yebowen Hu, Wenlin Yao, Sangwoo Cho, Xiaoyang Wang, Xuansheng Wu, Fei Liu, Pengfei Liu, and Dong Yu. 2024. InFoBench: Evaluating instruction following ability in large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 13025–13048, Bangkok, Thailand. Association for Computational Linguistics.

Haizhou Shi, Zihao Xu, Hengyi Wang, Weiyi Qin, Wenyuan Wang, Yibin Wang, Zifeng Wang, and Hao Wang. 2024. Continual learning of large language models: A comprehensive survey. *ACM Computing Surveys*, 58:1 – 42.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. 2026. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7080–7106, Rabat, Morocco. Association for Computational Linguistics.

Zifeng Wang, Zizhao Zhang, Sayna Ebrahimi, Ruoxi Sun, Han Zhang, Chen-Yu Lee, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and 1 others. 2022a. Dualprompt: Complementary prompting for rehearsal-free continual learning. In *European conference on computer vision*, pages 631–648. Springer.

Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. 2022b. Learning to prompt for continual learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 139–149.

Tongtong Wu, Linhao Luo, Yuan-Fang Li, Shirui Pan, Thuy-Trang Vu, and Gholamreza Haffari. 2024. Continual learning for large language models: A survey. *arXiv preprint arXiv:2402.01364*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. In *International Conference on Learning Representations*, volume 2024, pages 12028–12068.

Junjie Ye, Guoqiang Zhang, Wenjie Fu, Tao Gui, Qi Zhang, and Xuanjing Huang. 2026. Cctu: A benchmark for tool use under complex constraints. *CoRR*, abs/2603.15309.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. 2025. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639:609 – 616.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. 2026. Agentic context engineering: Evolving contexts for self-improving language models. In *The Fourteenth International Conference on Learning Representations*.

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

A Protocol Pseudocode

Algorithm 1 RECAP Evaluation Protocol

Require: Schedule $\{(k, \text{op}_k)\}_{k=0}^{K}$, method $\mathcal{M}$, model θ
1: $\mathcal{M}$.reset()
2: **for** $k = 0 \ldots K$ **do**
3: ctx $\leftarrow$ StepContext(op$_k$, reg_meta) ▷ see App. I
4: $\mathcal{M}$.adapt(θ, ctx) ▷ proactive; no test results
5: Evaluate on all $\mathcal{C}_{0:k}$ + shadow-eval edited/deleted
6: **end for**

Here θ denotes the frozen LLM backbone, `reg_meta` is the regression metadata (which types are new, retained, edited, or deleted), and StepContext bundles the operation and metadata into the adaptation interface.

B Constraint Taxonomy

Table 1 lists all 21 constraint types.

C Schedule Configuration

Interleaved-20. Edits and deletes interleaved with adds throughout 20 steps. Types span rule-based and LLM-judged. Features: edit-on-edit (Length edited at steps 5 and 16), delete-after-edit (Style edited then deleted), variable post-deletion windows (12, 8, 5, 0 steps).

Clustered-20. Identical operations but phased: ADDs first (steps 0–10), then EDITs (steps 11–15), then DELETEs (steps 16–19). Clustered produces 12% higher forgetting than interleaved (0.251 vs. 0.223), consistent with classical CL findings.

Table 1: Full constraint taxonomy. Rule-based types use deterministic validators; LLM-based types use an LLM judge.

Type	Eval Method	Example
`Length`	Rule: `evaluate_word_length` / `evaluate_sentence_length`	"Keep your answer around 200 words"
`Keyword`	Rule: `evaluate_keyword`	"Include 'sustainability' 3 times"
`Format`	Rule: `evaluate_format`	"Respond in JSON format"
`Start_With`	Rule: `evaluate_start_with`	"Begin with the word 'Dear' "
`End_With`	Rule: `evaluate_end_with`	"End with 'Sincerely' "
`All_Upper`	Rule: `check_english_uppercase`	"Write entirely in uppercase"
`All_Lower`	Rule: `check_english_lowercase`	"Write entirely in lowercase"
`No_Commas`	Rule: `contains_no_punctuation`	"Do not use any commas"
`Style`	LLM Judge	"Write in a formal academic style"
`Topic`	LLM Judge	"Focus on renewable energy"
`Tone`	LLM Judge	"Use an encouraging tone"
`Emotion`	LLM Judge	"Express gratitude"
`Example`	LLM Judge	"Include a real-world example"
`Factuality`	LLM Judge	"Ensure all facts are accurate"
`Helpfulness`	LLM Judge	"Provide actionable advice"
`Language`	LLM Judge	"Write in Spanish"
`Numerical Constraints`	LLM Judge	"List exactly 5 items"
`Role Playing`	LLM Judge	"Respond as a financial advisor"
`Situation`	LLM Judge	"Assume the reader is a beginner"
`Background Info`	LLM Judge	"Reference the 2024 climate report"
`Unclassified`	LLM Judge	Constraints not fitting above categories

Table 2: The three evaluation schedules. A/E/D = adds/edits/deletes.

Schedule	Steps	A/E/D	Types	Rationale
Interleaved-20	20	11/5/4	11	Mixed ops
Clustered-20	20	11/5/4	11	Phased A→E→D
Rule-Only-15	15	7/4/4	6	Determ. eval

Rule-Only-15. Only 6 rule-based types with deterministic validators. Confirms that findings are not artifacts of LLM judge noise: method rankings are preserved.

D Detailed Metric Definitions

Satisfaction. For type t at step k:

$$r_{\text{sat}}(t,k) = \frac{|\{x : t \text{ satisfied}\}|}{|\{x : t \text{ active}\}|}$$

We report mean satisfaction across types and steps.

Peak forgetting.

$$f_{\text{peak}}(t,k) = \max_{j<k} r_{\text{sat}}(t,j) - r_{\text{sat}}(t,k)$$

Clamped to $[0,\infty)$.

Collateral damage. When step k targets type t^*, let $\mathcal{R}_k = \mathcal{C}_{0:k} \setminus \{t^*\}$:

$$\text{coll}(k) = \frac{1}{|\mathcal{R}_k|} \sum_{t \in \mathcal{R}_k} \max\big(0,\ \Delta_t(k)\big)$$

Step ID	Operation	Type
0	add	Length
1	add	Keyword
2	add	Format
3	add	Style
4	add	Topic
5	edit	Length
6	add	Start_With
7	delete	Format
8	add	End_With
9	edit	Style
10	add	Tone
11	delete	Keyword
12	edit	Topic
13	add	Emotion
14	delete	Style
15	add	Helpfulness
16	edit	Length
17	add	Factuality
18	edit	Tone
19	delete	End_With

Table 3: Interleaved-20 schedule configuration.

where $\Delta_t(k) = r_{\text{sat}}(t,k{-}1) - r_{\text{sat}}(t,k)$.

Edit switch.

$$\text{switch} = \frac{\text{Adapted}}{\text{Both} + \text{Stale} + \text{Adapted}}$$

where Adapted = satisfies new but not old, Both = satisfies both, Stale = satisfies old but not new.

Unlearning Fidelity. When type d is deleted at step k:

$$\mathrm{UF}(d,k) = 1 - \frac{|r_{\text{post}}(d,k) - r_{\text{def}}(d)|}{\max\big(|r_{\text{pre}}(d) - r_{\text{def}}(d)|,\ \epsilon\big)}$$

where r_{pre} is frozen at deletion, r_{post} is shadow-evaluated at step k, r_{def} is the unconstrained default rate, and ϵ=0.05. Clamped to $[0,1]$. Three variants: immediate, sustained (reported), rebound.

Trajectory metrics. $\overline{\text{sat}}$ = mean across steps. Init. = satisfaction at first introduction (forward transfer). Final = last-step average. BWT = mean change between first appearance and final step per type.

Efficiency. Step latency = adapt time + generation time (seconds). Tokens = total LLM tokens consumed across adaptation and inference.

E Per-Backbone Results

Figure 2 in the main body reports a subset of metrics for Llama-3.3-70B and GPT-OSS-120B. Tables 4–7 below report all metrics for all four backbones. Table 8 consolidates efficiency metrics. All values are averaged over 3 schedules (n=3).

F Rule-Only Schedule Analysis

The Rule-Only-15 schedule uses exclusively deterministic constraint types (Length, Keyword, Format, Start_With, End_With, No_Commas), eliminating LLM judge noise entirely. Figure 4 shows that the same patterns observed on the full schedules persist under purely deterministic evaluation: Base LLM leads or ties on 3 of 4 backbones, all methods converge within 0.008 on Llama-70B, and adaptation is actively harmful on GPT-OSS-120B (MIPROv2 drops to 0.441, GEPA to 0.552 vs. base 0.654). This confirms our findings are not artifacts of LLM judge variance.

G Failure Mode Analysis

We identify six systematic failure modes across all 72 production runs. Each illustrates a structural limitation of prompt-level continual adaptation under the proactive protocol.

Failure 1: Specification Lock (MIPROv2, GPT-OSS-20B). MIPROv2's compiled prompt hard-codes constraint values from early steps and never revises them after edits. Self-validation reports perfect scores despite a -0.140 gap versus base.

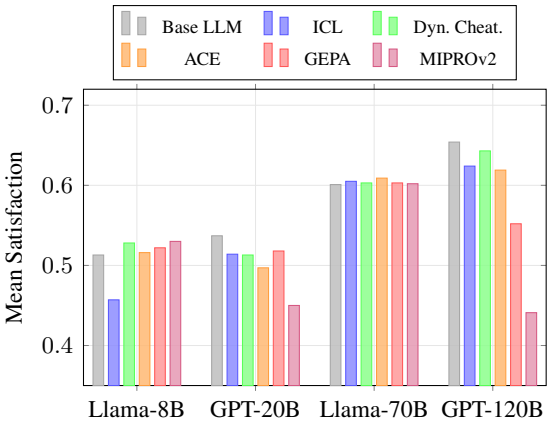

Figure 4: Rule-Only-15 schedule: mean satisfaction by backbone (deterministic evaluation only, no LLM judge). Same conclusion holds: Base LLM dominates on GPT-OSS models; all methods converge on Llama-70B.

> **Clustered Step 12, GPT-OSS-20B**: Current constraints require Start_With "To", keywords "Fair Trade Certification" and "transparent reports", topic about action movies. Compiled prompt locks stale values: topic=DACA, prefix="In addressing the legal challenges of DACA." Response follows the stale system prompt (1/12). Base LLM follows current constraints (5/12).

Failure 2: Progressive Refusal Cascade (ACE, GPT-OSS-120B). ACE's playbook grows from 187 to 9,085 characters over 20 steps. The model perceives conflicts between accumulated rules and current constraints, refusing 14% of samples.

> **Step 15, GPT-OSS-120B**: "I'm sorry, but I can't fulfill this request as it contains conflicting constraints that cannot be satisfied simultaneously." Base LLM satisfies 4/9 constraints—no actual conflict exists.

Failure 3: Demo Noise (ICL, Llama-3.1-8B). Synthetic demonstrations bloat context by 90–130% and introduce surface patterns (preambles, list punctuation) that conflict with active constraints.

> **Step 7, Llama-8B**: Active constraints include Start_With ("Google"), No_Commas, Length ($\leq$3 sentences). ICL response begins "Here are two classroom management tools...", uses commas, generates 107 words. Base LLM correctly starts "Google Classroom is a game-changer..." using dashes instead of commas (5/8 vs. ICL 0/8).

Failure 4: Prefix Contamination (GEPA, GPT-OSS-20B). GEPA's evolved prompts inject a fixed prefix ("Here is what you asked for:") into 19/50 responses at affected steps, violating all Start_With constraints.

Table 4: Trajectory-level results for **Llama 3.1 8B** (averaged over 3 schedules). Bold = best; underline = second best.

Method	Satisfaction			Regression			Edit / Delete	
	$\overline{\text{sat}}$	Init.	Final	Forg.↓	Coll.↓	BWT↑	Switch↑	UF_{sus} ↑
Base LLM	0.490±0.020	**0.665±0.037**	0.536±0.036	**0.204±0.032**	0.058±0.021	-0.096±0.066	0.245±0.116	0.589±0.175
ICL	0.399±0.052	0.598±0.097	0.302±0.130	0.269±0.055	0.058±0.028	-0.148±0.044	0.252±0.156	0.623±0.200
Dyn. Cheat.	0.480±0.042	0.652±0.054	0.497±0.051	0.220±0.078	0.049±0.021	**0.016±0.152**	0.253±0.133	0.607±0.158
ACE	0.425±0.106	0.605±0.108	0.425±0.135	0.252±0.106	**0.043±0.019**	0.009±0.029	0.240±0.147	0.565±0.166
GEPA	0.483±0.034	0.573±0.116	0.492±0.039	0.209±0.045	0.050±0.020	-0.146±0.171	0.253±0.126	**0.639±0.190**
MIPROv2	**0.507±0.021**	0.647±0.081	**0.550±0.031**	0.212±0.013	0.052±0.019	-0.197±0.113	**0.257±0.145**	0.596±0.158

Table 5: Trajectory-level results for **GPT-OSS 20B** (averaged over 3 schedules). Bold = best; underline = second best.

Method	Satisfaction			Regression			Edit / Delete	
	$\overline{\text{sat}}$	Init.	Final	Forg.↓	Coll.↓	BWT↑	Switch↑	UF_{sus} ↑
Base LLM	**0.555±0.016**	0.625±0.123	**0.518±0.045**	**0.184±0.023**	0.056±0.045	**-0.153±0.252**	0.310±0.170	0.611±0.174
ICL	0.539±0.022	**0.754±0.026**	0.486±0.093	0.221±0.009	0.055±0.021	-0.172±0.194	0.298±0.166	0.644±0.161
Dyn. Cheat.	0.549±0.036	0.672±0.035	0.480±0.112	0.248±0.067	**0.049±0.016**	-0.274±0.148	**0.317±0.209**	0.612±0.173
ACE	0.379±0.106	0.627±0.065	0.277±0.120	0.315±0.110	0.065±0.030	-0.161±0.177	0.251±0.188	0.559±0.253
GEPA	0.466±0.053	0.658±0.063	0.496±0.165	0.284±0.018	0.071±0.034	-0.213±0.095	0.283±0.188	0.635±0.144
MIPROv2	0.415±0.068	0.662±0.056	0.340±0.146	0.281±0.044	0.062±0.012	-0.237±0.200	0.275±0.168	**0.752±0.248**

> **Step 14, GPT-OSS-20B**: Constraint "Start with 'Google' ". GEPA response starts "Here is what you asked for: I recommend Google Classroom…" (0/8). Base correctly starts "Google Classroom and Schoology are the tools…" (4/8).

Failure 5: Inert Cheatsheet Growth (Dynamic Cheatsheet, GPT-OSS-120B). The cheatsheet grows to 3,945 characters of general strategies, yet 62% of responses are identical to Base LLM. Net improvement: +0.005 (within noise).

> **Step 16, GPT-OSS-120B**: Cheatsheet advice "Include testimonials for credibility" introduces uppercase in a response requiring All_Lower, causing the cheatsheet to *harm* (5/8 vs. base 6/8).

Failure 6: Keyword Anti-Unlearning (all methods, GPT-OSS-120B). After Keyword deletion, target keywords persist in 20–40% of responses due to topic-keyword semantic correlation. UF peaks at 0.62, never reaching 1.0.

> **Step 14, GPT-OSS-120B**: Deleted keyword "ethnographic" appears naturally ("a recent ethnographic fieldwork project…") because it is semantically related to the topic. Structural constraints (End_With) achieve near-perfect unlearning (3.7% residual).

H Per-Method Behavioral Profiles

Base LLM succeeds by doing nothing harmful—constraint text in the user prompt is sufficient. **ICL** achieves 0% win rate; demos add 90–130% token overhead and introduce conflicting patterns. **Dynamic Cheatsheet** is the safest adaptive method (42% win rate, -0.008 delta) because it produces transferable strategies rather than specific values. **ACE** is the most polarized: +0.007 on 70B but -0.176 on GPT-OSS-120B due to progressive refusal cascades. **GEPA** spends 311 additional API calls per run for +0.008 on 70B and -0.124 on 120B. **MIPROv2** consistently helps Llama-8B (+0.017) but collapses on GPT-OSS-20B (-0.140) via specification lock. GPT-OSS models copy system-prompt patterns literally (84–94% prefix contamination); Llama models treat them as guidance.

I Implementation Details

Adaptation interface: The adaptation interface encapsulates all information available to a method at each step:

`step_id`: Integer step identifier.

`step_ops`: Operations at this step, including concrete constraint text (e.g., `[{"op": "edit", "type": "Length", "new_value": "Keep under 500 words"}]`).

`step_meta`: Step metadata (schedule name, operation counts, type inventory).

`regression_meta`: Which types are new, retained, removed, edited, plus cumulative deleted/edited sets.

`prev_state`: Opaque state from previous adaptation call; `None` for step 0.

Table 6: Trajectory-level results for **Llama 3.3 70B** (averaged over 3 schedules). Bold = best; underline = second best.

	Satisfaction			Regression			Edit / Delete	
Method	$\overline{\text{sat}}$	Init.	Final	Forg.↓	Coll.↓	BWT↑	Switch↑	UF_{sus} ↑
Base LLM	0.595±0.011	0.653±0.042	0.638±0.013	0.235±0.030	0.048±0.025	-0.239±0.107	0.295±0.190	0.642±0.154
ICL	0.592±0.016	0.586±0.022	0.641±0.022	**0.188±0.026**	**0.033±0.012**	-0.015±0.113	**0.301±0.190**	0.650±0.150
Dyn. Cheat.	0.600±0.010	0.595±0.007	0.651±0.028	0.204±0.037	0.048±0.034	-0.029±0.121	0.300±0.170	**0.693±0.089**
ACE	0.602±0.014	0.629±0.040	**0.668±0.049**	0.212±0.005	0.035±0.021	-0.073±0.070	0.290±0.176	0.683±0.111
GEPA	**0.603±0.004**	**0.671±0.100**	0.668±0.023	0.198±0.006	0.048±0.026	**-0.013±0.181**	0.291±0.157	0.643±0.176
MIPROv2	0.595±0.009	0.618±0.029	0.634±0.042	0.209±0.033	0.034±0.012	-0.218±0.094	0.282±0.175	0.672±0.134

Table 7: Trajectory-level results for **GPT-OSS 120B** (averaged over 3 schedules). Bold = best; underline = second best.

	Satisfaction			Regression			Edit / Delete	
Method	$\overline{\text{sat}}$	Init.	Final	Forg.↓	Coll.↓	BWT↑	Switch↑	UF_{sus} ↑
Base LLM	**0.630±0.023**	**0.683±0.078**	**0.639±0.031**	**0.179±0.021**	0.053±0.027	0.033±0.131	0.309±0.192	0.674±0.121
ICL	0.598±0.023	0.682±0.043	0.575±0.056	0.211±0.015	**0.040±0.013**	**0.075±0.023**	**0.317±0.207**	**0.730±0.121**
Dyn. Cheat.	0.610±0.036	0.624±0.060	0.587±0.054	0.224±0.034	0.050±0.022	-0.094±0.179	0.315±0.212	0.709±0.108
ACE	0.454±0.151	0.639±0.078	0.351±0.222	0.330±0.095	0.050±0.014	-0.180±0.170	0.289±0.212	0.610±0.198
GEPA	0.506±0.100	0.663±0.177	0.524±0.153	0.268±0.072	0.049±0.028	-0.059±0.119	0.301±0.197	0.648±0.145
MIPROv2	0.571±0.113	0.629±0.025	0.620±0.025	0.222±0.082	0.053±0.014	-0.030±0.216	0.256±0.098	0.635±0.054

`num_samples`: Window size (number of samples per evaluation step).

Hyperparameters: ICL: num_demos=3, constraint-matched selection, pool cap 30. Dynamic Cheatsheet (Suzgun et al., 2026): max 4K tokens. ACE: 4K token playbook budget, 4-call self-play pipeline. GEPA: population size 4, 2 generations, elite size 2, 2K token budget (17 calls/step). MIPROv2: 3 proposals/step, max 20 history entries (∼11 calls/step). All self-play methods use the backbone as both generator and judge (2 calls per evaluation).

Efficiency metrics: We track per-step: adaptation wall-clock time, generation wall-clock time, constraint scoring time, tokens consumed during adaptation, tokens for test responses, LLM API calls during adaptation, and LLM API calls for test generation.

J Self-Play Budget Ablation

We test whether ACE's underperformance is due to insufficient optimization by scaling the self-play budget from 1× (4 LLM calls/step; default) to 3× (12 calls, 3 Gen→Eval→Reflect→Curate cycles) and 5× (20 calls, 5 cycles) on GPT-OSS-120B. Table 12 reports the results.

Key finding: more optimization accelerates failure. Mean satisfaction *decreases* monotonically with budget ($0.454 \rightarrow 0.421 \rightarrow 0.340$), widening the gap to Base LLM (0.630). The mechanism is the Refusal Cascade (§G): each cycle adds rules to the playbook, so 5× grows the playbook to 15.6K characters (vs. 9.1K at 1×), exceeding the model's tolerance. Refusal rates rise from 12% to 48%, directly explaining the satisfaction drop.

Schedule-dependent nuance: On the Rule-Only-15 schedule (6 rule-based types, deterministic evaluation), more cycles *marginally help*: mean satisfaction rises $0.619 \rightarrow 0.624 \rightarrow 0.636$ and refusal rates actually *decrease* ($10.9\% \rightarrow 9.1\% \rightarrow 6.8\%$). Rule-based constraints produce unambiguous self-play signal—binary pass/fail on concrete specifications (word counts, keywords)—so the Reflector generates actionable rules that the Curator can integrate without bloat. On LLM-judged schedules (Interleaved-20, Clustered-20), the same mechanism fails: vague constraint specifications ("adopt a persuasive narrative style") produce noisy self-play verdicts, leading to verbose, hedge-laden playbook entries that accumulate into perceived contradictions.

Implication: The failure of proactive adaptation on complex schedules is *structural*, not computational. The bottleneck is not insufficient self-play optimization but distribution mismatch between the fixed synthetic self-play task and the diverse real evaluation prompts. Scaling compute within this

Table 8: Efficiency metrics across all four backbones (averaged over 3 schedules). Step latency includes adaptation and generation time (seconds). Tokens is the total LLM tokens consumed per run.

	Llama 3.1 8B		GPT-OSS 20B		Llama 3.3 70B		GPT-OSS 120B	
Method	Lat.(s)	Tok.	Lat.(s)	Tok.	Lat.(s)	Tok.	Lat.(s)	Tok.
Base LLM	108.6	491K	61.3	643K	125.5	468K	97.2	796K
ICL	139.5	1,122K	68.2	1,045K	131.8	935K	98.4	1,103K
Dyn. Cheat.	114.5	918K	72.8	1,223K	128.7	767K	95.0	1,351K
ACE	129.7	1,035K	89.1	1,578K	136.0	819K	109.7	1,623K
GEPA	141.0	852K	106.0	1,362K	189.4	841K	135.5	1,459K
MIPROv2	127.3	764K	74.4	861K	136.0	719K	130.5	1,660K

Table 9: Full results for the Rule-Only-15 schedule (deterministic evaluation only).

BB	Method	$\overline{\text{sat}}$	Forg.↓	Coll.↓	Sw.↑	UF↑
8B	Base LLM	0.513	0.206	0.035	0.379	0.791
	ICL	0.457	0.222	0.036	0.431	0.851
	Dyn. Ch.	**0.528**	0.181	**0.024**	0.406	0.789
	ACE	0.516	0.198	0.026	0.410	0.750
	GEPA	0.522	**0.162**	0.027	0.398	**0.857**
	MIPROv2	0.530	0.199	0.029	**0.424**	0.778
20B	Base LLM	**0.537**	**0.159**	**0.025**	0.506	0.812
	ICL	0.514	0.231	0.033	0.488	0.807
	Dyn. Ch.	0.513	0.219	0.031	**0.557**	0.811
	ACE	0.497	0.209	0.037	0.468	**0.851**
	GEPA	0.518	0.303	0.033	0.501	0.789
	MIPROv2	0.450	0.254	0.049	0.467	0.751
70B	Base LLM	0.601	0.269	0.020	0.514	0.817
	ICL	0.605	0.212	**0.019**	**0.519**	**0.819**
	Dyn. Ch.	0.603	0.242	0.020	0.493	0.790
	ACE	**0.609**	**0.212**	0.017	0.493	0.810
	GEPA	0.603	0.191	0.018	0.470	0.839
	MIPROv2	0.602	0.244	0.021	0.485	0.818
120B	Base LLM	**0.654**	**0.175**	**0.022**	0.531	0.794
	ICL	0.624	0.219	0.031	**0.556**	0.825
	Dyn. Ch.	0.643	0.199	0.025	0.560	0.823
	ACE	0.619	0.254	0.039	0.533	**0.834**
	GEPA	0.552	0.278	0.026	0.528	0.662
	MIPROv2	0.441	0.316	0.037	0.368	0.584

paradigm only accelerates artifact growth without improving transfer.

K Full-Constraints Information Ablation

We test whether ACE's failure is due to an *information gap*: in the standard protocol, `adapt()` sees only the current step's operation and a list of active type names. In this ablation, we additionally provide the full text of *all* currently active constraints, so the self-play pipeline evaluates against the same specifications the model will face at test time. We run ACE on GPT-OSS-120B (the worst-affected backbone; standard ACE: 0.454, base: 0.630) across 3 schedules.

Key finding: closing the information gap is necessary but not sufficient. Full-constraints ACE recovers 91% of the standard ACE deficit ($0.454 \rightarrow 0.614$; gap to base narrows from -0.176 to -0.016) and nearly eliminates excess forgetting ($0.330 \rightarrow 0.203$, vs. base 0.179). The step-by-step trajectory tracks Base LLM closely throughout all 20 steps, confirming that the fix is not merely a better initialization but sustained alignment. However, despite this recovery, full-constraints ACE does *not* exceed base on any primary metric—it achieves expensive neutrality at $2.8\times$ the token cost.

What drives the remaining gap: Refusal rates remain comparable to standard ACE ($\sim 13\%$ vs. $\sim 12\%$; base: 0%), indicating that the playbook representation itself—regardless of information quality—still causes occasional perceived contradictions. The Gen→Eval→Reflect→Curate pipeline introduces indirection: constraint knowledge must survive four LLM transformations before reaching the playbook, introducing noise and verbosity at each stage.

Implication: The information gap explains ACE's *catastrophic* failure on GPT-OSS-120B, but the fundamental limitation is architectural: for constraint satisfaction tasks where the specification is self-contained, no amount of meta-cognitive machinery (reflection, curation, playbook management) improves upon directly presenting the constraints in the user prompt. The constraints *are* the optimal prompt.

Table 10: Six failure modes observed across 72 production runs. Each represents a structural misalignment between the method's design assumptions and the proactive protocol. Δsat is relative to Base LLM on the worst-affected backbone.

#	Mode	Method	Backbone	Mechanism	Δsat
1	Spec. Lock	MIPROv2	20B	Compiled prompt hardcodes early constraint values (topic, prefix, keywords) and is never revised after edits. Model follows stale system prompt directives instead of current constraints. Model may even echo system prompt content.	-0.140
2	Refusal Cascade	ACE	120B	Playbook grows to 9K chars; model perceives phantom contradictions between accumulated rules and constraints. 14% refusal rate.	-0.176
3	Demo Noise	ICL	8B	Synthetic demos bloat context 90–130%; surface patterns (preambles, punctuation) conflict with active constraints.	-0.091
4	Prefix Contam.	GEPA	20B	Evolved prompt injects fixed prefix ("Here is what you asked for:") violating all Start_With constraints. 19% of samples affected.	-0.089
5	Inert Growth	Dyn. Cheat.	120B	Cheatsheet grows to 4K chars of generic advice; 62% of responses identical to base.	-0.020
6	Anti-Unlearn	All	120B	Topic-keyword semantic correlation causes deleted keywords to persist in 20–40% of responses.	UF$\leq$0.62

Table 11: Method behavioral profiles across 12 conditions (4 backbones $\times$ 3 schedules). Delta = mean difference from Base LLM.

Method	Delta	Win%	Best BB	Worst BB	Adapt(s)
Base LLM	—	—	120B	8B	0
ICL	-0.036	0%	70B	8B	24
Dyn. Cheat.	-0.008	42%	70B	120B	64
ACE	-0.103	17%	70B	20B/120B	224
GEPA	-0.053	25%	70B	120B	739
MIPROv2	-0.045	33%	8B	20B	442

Table 12: Self-play budget ablation for ACE on GPT-OSS-120B, averaged across 3 schedules. More optimization *degrades* quality.

Budget	$\overline{\text{sat}}$	Forg.$\downarrow$	Refusal%	Tok.
1$\times$ (4 calls)	0.454	0.330	12.4%	1.62M
3$\times$ (12 calls)	0.421	0.326	28.6%	2.39M
5$\times$ (20 calls)	0.340	0.390	47.8%	2.82M
Base LLM (0 calls)	0.630	0.179	0.0%	0.80M

Table 13: Full-constraints ablation for ACE on GPT-OSS-120B, averaged across 3 schedules. Closing the information gap recovers most of the deficit but yields zero net improvement over Base LLM.

Condition	$\overline{\text{sat}}$	Forg.$\downarrow$	BWT$\uparrow$	Switch$\uparrow$	Tok.
Base LLM	**0.630**	**0.179**	**0.033**	0.309	0.80M
ACE (full-ctx)	0.614	0.203	-0.032	**0.321**	2.26M
ACE (standard)	0.454	0.330	-0.180	0.289	1.62M