Not All Instructions Are Forgotten Equal

Abstract. Large language models lose adherence to user-stated preferences over extended coding conversations, even when system prompt instructions remain stable. We investigate this per-instruction retention heterogeneity using a Bayesian ordered logistic model fitted to 244 compliance observations across 5 models, 12 decision types, and 3 real-world codebases. We find that treatment effects vary by over an order of magnitude: some preferences are retained perfectly when stated, others are ignored, and one is actively harmed by reinforcement. These results establish the empirical prerequisite for selective reinforcement: the per-instruction heterogeneity that a Bayesian Knowledge Tracing system would need to exploit. A posterior-derived reinforcement ranking identifies that 3 of 12 instruction types benefit from reinforcement, 1 should never be reinforced, and 8 require no intervention.

Keywords: instruction compliance , context rot , Bayesian inference , selective reinforcement , LLM evaluation

No Todas las Instrucciones se Olvidan Igual

Abstract. Los modelos de lenguaje grandes pierden adherencia a las preferencias del usuario a lo largo de conversaciones extendidas de programación, incluso cuando las instrucciones del system prompt se mantienen estables. Investigamos esta heterogeneidad en la retención por tipo de instrucción usando un modelo logístico ordinal bayesiano ajustado a 244 observaciones de cumplimiento en 5 modelos, 12 tipos de decisión y 3 codebases reales. Encontramos que los efectos del tratamiento varían en más de un orden de magnitud: algunas preferencias se retienen perfectamente al ser establecidas, otras se ignoran, y una empeora activamente con el refuerzo. Estos resultados establecen el prerrequisito empírico para el refuerzo selectivo: la heterogeneidad por tipo de instrucción que un sistema basado en Bayesian Knowledge Tracing necesitaría explotar.

Keywords: cumplimiento de instrucciones , degradación de contexto , inferencia bayesiana , refuerzo selectivo , evaluación de LLMs

1 Introduction

Large language models lose adherence to instructions over extended conversations, a phenomenon we refer to as *context rot*. Laban et al. (2025) measured compliance drops of up to 39% across 15 models in multi-turn settings. The cause is architectural: attention is not uniformly distributed across the context window, and information in middle positions receives less weight than information at the boundaries (Liu et al., 2024). System prompt instructions sit at position zero, where causal attention gives them a logarithmic compounding advantage (Chowdhury, 2026). User-stated preferences, introduced mid-conversation, get no such benefit. They compete for attention against all the context that accumulates after them.

This matters because every reinforcement token costs money, and weakened safety guardrails accumulate without detection over long sessions (Mu et al., 2025). The current fix is to repeat all instructions every turn. This multiplies prompt cost and still fails to distinguish between instructions the model has retained and those it has forgotten.

No prior work has measured this heterogeneity at the level of individual instructions. Existing studies report mean compliance across instruction sets (He et al., 2024; Laban et al., 2025) or propose uniform mitigation strategies such as periodic reminders (Dongre et al., 2025). But if each instruction decays at a different rate, the mean hides the structure that matters. An instruction that is already retained does not need reinforcement. An instruction that is harmed by repetition should not receive it. Without per-instruction estimates, any reinforcement policy is flying blind.

Bayesian Knowledge Tracing (Corbett & Anderson, 1994) offers a framework for exactly this kind of estimation. Originally developed in education, BKT estimates the probability that a student has mastered a concept based on a sequence of observed responses. The conceptual inversion we propose is to treat the LLM as the student and its instructions as the concepts. A BKT-based reinforcement system would only be useful if instruction retention is in fact heterogeneous. We test this using a Bayesian ordered logistic model with hierarchical effects per decision type, fitted to compliance scores collected from realistic multi-turn coding conversations across three open-source codebases.

We report three findings. First, retention heterogeneity across instruction types is large, with treatment effects on the log-odds scale ranging from -2.4 to $+5.4$ (group-level standard deviation $\sigma_\beta = 2.1$). Second, one instruction type (`dependencies_no_new`) is actively harmed by reinforcement, with compliance dropping when the preference is stated. Third, neither model architecture nor codebase size explains meaningful variance in retention; the variation is almost entirely in the instruction type itself. These results establish that selective reinforcement is necessary, and that a Bayesian model can identify which instructions to reinforce, which to leave alone, and which to never mention again.

2 Related work

Instruction compliance degradation. Several recent studies have established that LLMs lose adherence to instructions as conversations grow. Laban et al. (2025) documented an average 39% compliance drop across 15 models in multi-turn settings over 200K conversations. He et al. (2024) showed that accuracy on multi-instruction tasks drops from 87.7% to 70.7% within three turns, even when instructions are given simultaneously. Du et al. (2025) demonstrated that increasing the context window alone does not resolve this degradation. All three studies report aggregate metrics across instruction types. None examines whether different instructions degrade at different rates.

Positional attention and prompt saturation. The mechanism behind compliance degradation has roots in the attention architecture itself. Liu et al. (2024) identified a U-shaped attention curve: information at the beginning and end of the context receives more weight, while middle positions are underweighted. This positional bias explains why system prompt instructions, which occupy position zero, remain more stable than user-stated preferences buried mid-conversation. Mu et al. (2025) showed a related saturation effect: as the number of guardrails in a system prompt increases, compliance with any individual rule drops sharply. Together, these results suggest that instruction adherence is not a single scalar that degrades uniformly, but a structured phenomenon shaped by position and quantity.

Uniform mitigation strategies. Existing approaches to maintaining compliance treat all instructions equally. Dongre et al. (2025) modeled instruction drift as a bounded stochastic process and proposed periodic reminders to reduce divergence. Leviathan et al. (2025) studied the effect of prompt repetition on instruction stability. Both approaches reinforce all instructions at each intervention, regardless of whether an individual instruction has been retained or forgotten. This uniform strategy becomes expensive at scale and, as our results show, can be counterproductive for instructions where repetition degrades compliance.

Bayesian models of LLM behavior. Zhang et al. (2025) demonstrated that LLMs behave as discounted Bayesian filters with a discount factor $\gamma < 1$, meaning that prior beliefs are systematically downweighted relative to recent observations. They measured this using controlled probabilistic probes (biased dice, Gaussian mean estimation) where ground-truth parameters shift and the model must adapt. Their work established that LLM forgetting has Bayesian structure, but did not measure whether γ varies across instruction types. In education, Bayesian Knowledge Tracing (Corbett & Anderson, 1994) has tracked per-concept mastery probabilities for over 30 years. We apply this same logic in reverse: rather than tracking what a student has learned, we track what an LLM has forgotten, and we show that the per-instruction heterogeneity that makes BKT useful in education also appears in LLM instruction compliance.

3 Method

3.1 Decision planting protocol

We simulate an AI coding assistant working on a real open-source codebase over 25 turns. At each turn, the assistant receives a user message and may call tools (read files, write code, search the codebase). Real source files are injected into the conversation at turns 0–19, building up context pressure as the conversation progresses.

In the **treatment** condition, 12 coding preferences are embedded as casual inline requests within the user's messages at specific turns. Six are planted early (turns 1–7) and six late (turns 14–19). Each preference targets a different coding practice: code style (numpy broadcasting, list comprehensions), architecture (subclassing vs standalone functions), testing (parametrize, approximate assertions), naming (underscore prefix, no abbreviations), dependencies (no new imports, module-level constants), and documentation (NumPy-style docstrings, regex comments). The instructions are phrased as natural user preferences, not system prompt rules (e.g., "Important: when writing array operations for this project, always use numpy broadcasting instead of for loops").

In the **baseline** condition, the conversation follows the same structure with the same file injections and the same task, but no preferences are planted. This allows us to measure what the model does by default for each decision type.

At turns 20–25, the user requests code generation tasks designed to elicit each decision. Each test turn evaluates exactly two decisions (one early-planted, one late-planted), and the model's code output is scored by deterministic checkers.

3.2 Compliance measurement

Each of the 12 decision types is evaluated by a deterministic checker that parses the model's code output using regular expressions and pattern matching. Checkers are fully automated with no LLM-in-the-loop evaluation. This makes results reproducible across runs. Each checker produces an ordinal score on a 0–3 scale:

- **0**: Completely ignored or violated
- **1**: Minimal or inconsistent compliance
- **2**: Mostly followed with minor deviations
- **3**: Fully followed

For example, the `parametrize` checker searches for `pytest.mark.parametrize` decorators and penalizes separate test functions. The `broadcasting` checker detects `for` loops and rewards numpy operations. The `no_new_imports` checker counts `import` statements in generated code.

3.3 Bayesian model

We model the ordinal compliance scores using an ordered logistic regression (cumulative link model) with hierarchical effects per decision type. The ordered logistic is appropriate because the 0–3 scores have unequal intervals: the gap between 0 (ignored) and 1 (minimal compliance) is qualitatively different from the gap between 2 (mostly followed) and 3 (fully followed). A linear model would incorrectly assume equal spacing.

For each observation i, the latent compliance η_i is:

$$\eta_i = \alpha_{d[i]} + \beta_{d[i]} \cdot \text{treatment}_i \tag{1}$$

where $d[i]$ indexes the decision type, α_d is the baseline intercept (compliance without being told), and β_d is the treatment effect (how much telling improves compliance). The observed score is determined by where η falls relative to three ordered cutpoints $c_1 < c_2 < c_3$.

Both α and β are given hierarchical priors with non-centered parameterization:

$$\alpha_d = \mu_\alpha + \sigma_\alpha \cdot \tilde{\alpha}_d, \quad \tilde{\alpha}_d \sim \text{Normal}(0, 1) \tag{2}$$

$$\beta_d = \mu_\beta + \sigma_\beta \cdot \tilde{\beta}_d, \quad \tilde{\beta}_d \sim \text{Normal}(0, 1) \tag{3}$$

The hyperpriors are weakly informative on the log-odds scale: $\mu_\alpha \sim \text{Normal}(0, 1.5)$, $\mu_\beta \sim \text{Normal}(0, 1)$ (centered at zero, no prior assumption about direction), $\sigma_\alpha, \sigma_\beta \sim \text{Exp}(1)$. Cutpoints use an offset parameterization: $c_1 \sim \text{Normal}(0, 1.5)$ with positive increments $\Delta c \sim \text{Exp}(1)$ to guarantee ordering.

The key scientific parameter is σ_β: the group-level standard deviation of treatment effects. A large σ_β indicates that different instructions respond differently to being stated, which is the premise required for selective reinforcement.

We initially fit a full model including per-model and per-codebase intercepts. Both had posterior group-level standard deviations near zero ($\sigma_{\text{model}} \approx 0.4$, $\sigma_{\text{codebase}} \approx 0.3$) with all individual effects having credible intervals crossing zero, and the model produced 267 divergent transitions. We removed these terms and confirmed via the simplified model's clean diagnostics (zero divergences, all $\hat{R} \leq 1.01$, minimum ESS > 1400) that the data does not support meaningful model or codebase effects.

The model was implemented in PyMC (Abril-Pla et al., 2023). Posterior samples were drawn using the No-U-Turn Sampler (NUTS; Hoffman & Gelman, 2014) via the nutpie sampler, a Rust-based NUTS implementation (Seyboldt, 2024), with 4 chains of 1000 draws each following 1000 tuning steps. Prior predictive checks confirmed that the priors produce plausible score distributions before fitting. Posterior predictive checks verified that the fitted model reproduces the observed data.

Table 1. Models and experimental conditions. Only Qwen was tested in both baseline and treatment conditions.

Model	Parameters	Baseline	Treatment
Qwen 3.5	27B	✓	✓
Gemini 3.1 Flash Lite	–		✓
GPT-5.4-nano	–		✓
Nemotron 3 Super	120B		✓
Gemma 4	26B		✓

4 Experimental setup

4.1 Codebases

We selected three open-source Python libraries from the Bayesian statistics ecosystem, chosen to span a range of codebase sizes and thus context pressure levels:

- **Bambi** (∼10K lines): a high-level interface for Bayesian regression. By turn 20, injected files produce approximately 98K tokens of context.
- **ArviZ** (∼25K lines): a library for exploratory analysis of Bayesian models. Context reaches approximately 160K tokens by turn 20.
- **PyMC** (∼57K lines): a probabilistic programming framework. Context reaches approximately 203K tokens by turn 20.

All repositories were cloned at fixed commits: Bambi (`5110615`, 2026-03-17), ArviZ (`arviz-base` at `374cd28` and `arviz-stats` at `8d6316a`, 2026-03-26/29), and PyMC (`bca2b1e`, 2026-03-27, branch v6). Source files were injected into the conversation at turns 0–19 via the tool execution system, which serves real file contents from the cloned repositories.

4.2 Models

We evaluated five instruction-following models across two experiment phases. Table 1 summarizes the models and conditions.

Qwen served as the primary model with both baseline and treatment conditions across all three codebases (18 conversations). The remaining four models were tested in the treatment condition only, to verify that the retention patterns generalize across architectures. Gemma was tested on Bambi only due to budget constraints.

4.3 Dataset

We ran 28 conversations in total: 9 baseline and 19 treatment. Models were accessed through OpenRouter (Qwen, Gemini, Gemma, Nemotron) and the OpenAI API (GPT-5.4-nano). The final dataset consists of 244 compliance

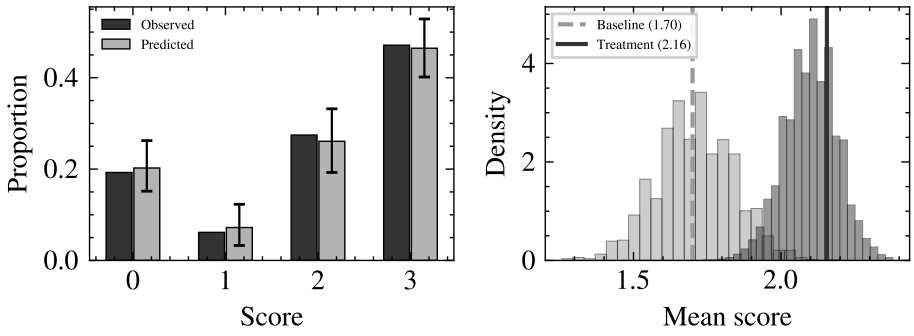

Fig. 1. Posterior predictive check. Left: the model reproduces the observed score distribution across all four ordinal categories. Right: the model captures the difference in mean compliance between baseline (1.70) and treatment (2.16) conditions.

observations: each observation is one decision type scored at one test turn in one conversation. Observations span 5 models, 12 decision types, 3 codebases, and 2 conditions. The baseline condition contributes 70 observations (all from Qwen), and the treatment condition contributes 174 observations across all five models. All conversation logs, checker code, and analysis notebooks are available in the supplementary material.

5 Results

The model converged cleanly: zero divergences, $\hat{R} \leq 1.01$ for all parameters, and minimum effective sample size above 1400. Posterior predictive checks confirm that the model reproduces the observed score distribution (Figure 1).

5.1 Treatment effect heterogeneity

The group-level standard deviation of treatment effects is $\sigma_\beta = 2.11$ (94% HDI: $[1.06, 3.28]$), indicating that different instruction types respond very differently to being stated. Individual treatment effects on the log-odds scale range from -2.36 to $+5.44$ (Figure 2).

The posterior separates the 12 decision types into three groups:

Instructions that benefit from reinforcement. Three decision types have treatment effects with 94% HDI entirely above zero. `testing_parametrize` has the largest effect ($\beta = 5.44$, HDI: $[2.94, 8.10]$): models almost never use `pytest.mark.parametrize` unprompted (baseline mean: 0.00) but retain the preference when told (treatment mean: 2.79). `dependencies_module_constants` ($\beta = 2.76$, HDI: $[0.60, 5.00]$) and `architecture_standalone` ($\beta = 2.38$, HDI: $[0.29, 4.63]$) follow the same pattern: near-zero baseline, meaningful retention when stated.

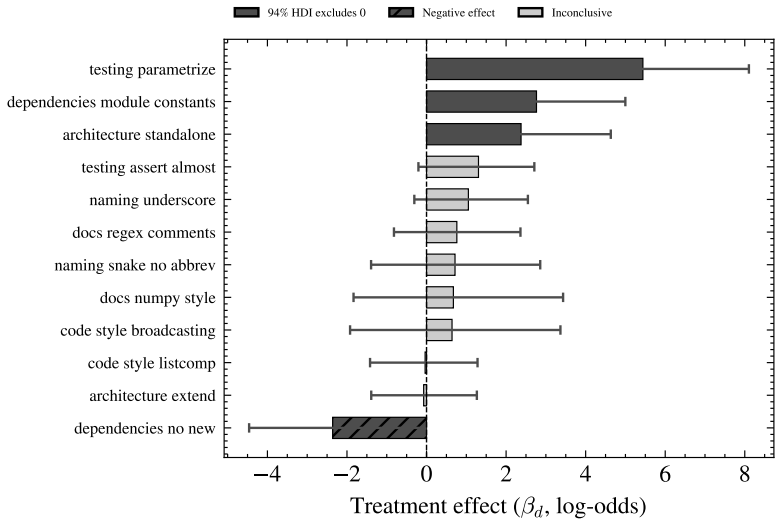

Fig. 2. Per-decision treatment effects (β_d) with 94% HDI. Dark bars: 94% HDI excludes zero. Hatched: negative effect. Light bars: inconclusive.

An instruction harmed by reinforcement. `dependencies_no_new` ($\beta = -2.36$, HDI: $[-4.46, -0.03]$) is the only decision type where telling the model reduces compliance. The baseline mean score is 2.25 (models naturally avoid adding imports), but stating "do not add new imports" drops the treatment mean to 0.77. We hypothesize that the explicit negation confuses the model's instruction following. Testing this would require isolating the negation from the instruction content, e.g., comparing "avoid new imports" with "use only existing imports."

Instructions that need no intervention. The remaining eight decision types show treatment effects with HDIs crossing zero. Several of these have high baseline compliance (`broadcasting`: 3.00, `numpy_style`: 3.00, `snake_no_abbrev`: 2.67), meaning models already follow these conventions without prompting. Reinforcing these would waste tokens.

5.2 Model and codebase effects

As described in Section 3.3, we initially fit a model with per-model and per-codebase intercepts. Both had posterior group-level standard deviations near zero ($\sigma_{\text{model}} \approx 0.4$, $\sigma_{\text{codebase}} \approx 0.3$) and produced 267 divergences. The simplified model without these terms fits cleanly.

The retention landscape is consistent across five model architectures (ranging from 26B to 120B parameters) and three levels of context pressure (98K to 203K tokens). What determines whether an instruction is retained is the instruction itself, not the model processing it or the amount of surrounding context.

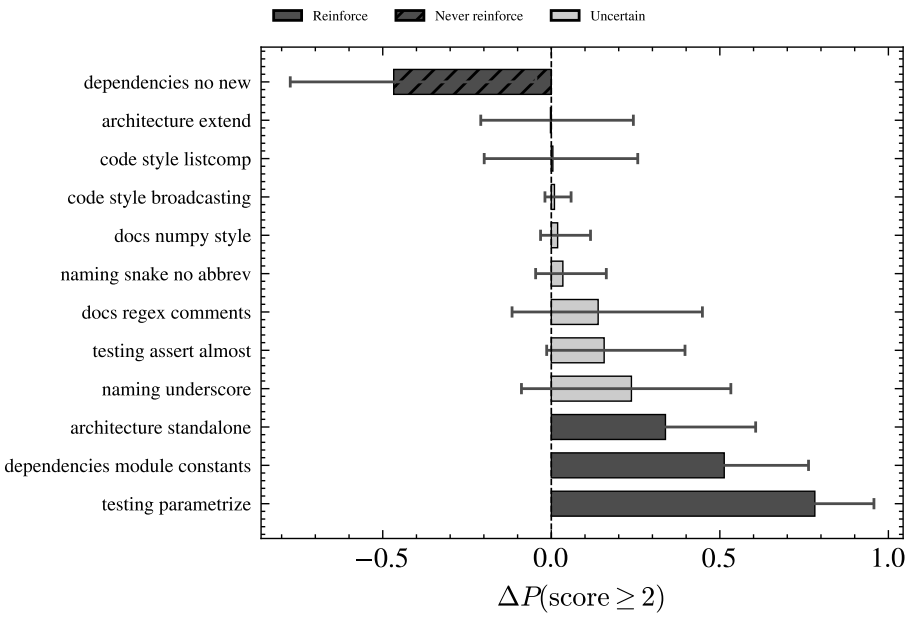

Fig. 3. Expected change in $P(\text{score} \geq 2)$ from reinforcing each decision, with 94% HDI. Dark bars: HDI excludes zero. Hatched: reinforcement is harmful.

5.3 Reinforcement priorities

We use the posterior to derive a reinforcement ranking. For each decision type d, we compute $P(\text{score} \geq 2 \mid \text{told})$ and $P(\text{score} \geq 2 \mid \text{not told})$ across all posterior samples, and define the reinforcement gain as their difference (Figure 3). This is not a closed-loop policy evaluation; it is a posterior-derived priority ranking that identifies which instructions would benefit from reinforcement.

Table 2 summarizes the ranking. Of 12 instruction types, only 3 have a reinforcement gain with 94% HDI entirely above zero. A uniform policy that reinforces all 12 would spend tokens on 8 instructions that need no help and 1 that is actively harmed by repetition.

6 Discussion

6.1 What determines retention

The treatment effects follow a pattern: instructions that ask for something the model would not do by default are retained when stated, while instructions that match existing behavior show no treatment effect. The three decisions with clear positive effects (`parametrize`, `module_constants`, `standalone`) all have baseline compliance near zero. The decisions with no treatment effect (`broadcasting`, `numpy_style`, `snake_no_abbrev`) have baseline compliance near 3.0. Models

Table 2. Reinforcement priorities derived from posterior estimates. Gain is $\Delta P(\text{score} \geq 2)$ from reinforcement.

Decision type	Gain	94% HDI	Action
`testing_parametrize`	$+0.78$	$[+0.46, +0.96]$	Reinforce
`module_constants`	$+0.51$	$[+0.18, +0.76]$	Reinforce
`arch_standalone`	$+0.34$	$[+0.06, +0.61]$	Reinforce
`naming_underscore`	$+0.24$	$[-0.09, +0.53]$	Uncertain
`testing_assert_almost`	$+0.16$	$[-0.01, +0.40]$	Uncertain
`docs_regex_comments`	$+0.14$	$[-0.12, +0.45]$	Uncertain
`6 other types`	$< +0.04$	crosses zero	Skip
`deps_no_new`	-0.47	$[-0.77, -0.05]$	Never reinforce

already follow these conventions without being told. Reinforcing them would be wasting tokens.

The negative effect on `dependencies_no_new` ($P(\beta < 0) = 0.98$) deserves separate attention. The instruction "do not add new imports" is the only one phrased as a negation. All other instructions tell the model what to do, not what to avoid. This raises the question of whether the negative treatment effect reflects something about the content of the instruction (import management) or its framing (negation). The answer matters for practitioners: if negation is the problem, reformulating as "use only existing imports" might recover the positive effect. If import management is inherently hard to retain, no framing will help. Distinguishing these hypotheses requires a controlled comparison that holds semantic content constant while varying the negation frame.

6.2 Robustness checks

To address the concern that baseline data comes from Qwen only, we fit the same model restricted to Qwen observations (132 of 244, both conditions). The per-decision treatment effects correlate at $r = 0.80$ with the full-model estimates, and the group-level heterogeneity parameter remains stable ($\sigma_\beta = 2.35$ vs. 2.11 in the full model). The three strongest effects replicate: `parametrize` ($\beta = 6.13$), `module_constants` ($\beta = 2.25$), and `dependencies_no_new` ($\beta = -1.07$, still negative though with wider intervals).

One decision, `architecture_standalone`, does not replicate in the Qwen-only analysis ($\beta = 2.38$ in the full model vs. $\beta = -0.71$ with Qwen alone). Its treatment effect in the full model appears driven by the other four models. We reduce the count of clearly beneficial decisions from three to two when considering only Qwen data.

Leave-one-out cross-validation shows no problematic observations (all Pareto $\hat{k} < 0.7$, ELPD $= -228.8 \pm 14.5$). Per-decision posterior predictive checks confirm that the model reproduces the observed mean score for all 12 decision types within the 90% predictive interval (Figure 4).

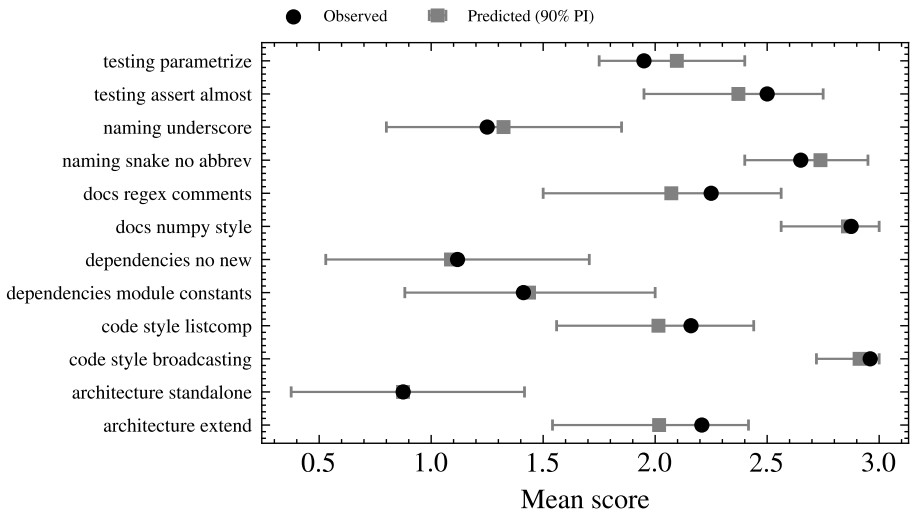

Fig. 4. Per-decision posterior predictive check. Observed means (dots) fall within the 90% posterior predictive interval for all 12 decision types.

6.3 Limitations

Baseline data comes from one model only (Qwen). The sensitivity analysis ($r = 0.80$) mitigates but does not eliminate this concern. Future replications should include baseline conditions for every model tested.

The dataset contains 244 observations, with some decision-by-condition cells containing as few as 4 observations. The hierarchical model handles sparsity through partial pooling, and the wide credible intervals for low-data decisions reflect this uncertainty honestly. However, the reinforcement ranking for the "uncertain" category (Table 2) should be treated as preliminary.

Compliance was measured at turns 20–25 only. This gives a retention snapshot, not a decay curve. We cannot estimate per-instruction decay rates (γ in the BKT framework) or determine at which turn compliance begins to drop. Fitting actual forgetting curves would require compliance measurements at every turn, which in turn requires designing prompts that give the model an opportunity to demonstrate each decision at each turn without making the conversation artificial.

The deterministic checkers have construct validity limitations. When a checker cannot find a relevant code pattern, it defaults to a score of 2 ("no clear signal"). Four decision types have default-2 rates above 50%: `architecture_extend` (79%), `docs_regex_comments` (75%), `code_style_listcomp` (72%), and `testing_assert_almost` (50%). The three decisions with the strongest treatment effects (`parametrize`, `standalone`, `no_new`) have 0% default-2 rates, so the headline findings are not driven by this artifact. The `no_new_imports` checker penalizes any `import` statement, including standard library modules that may be required by the task.

This paper does not implement Bayesian Knowledge Tracing. It measures the per-instruction heterogeneity that would make a BKT-based reinforcement system useful. Building and evaluating such a system, including closed-loop token-budget allocation and real-time compliance tracking, is future work.

6.4 Future work

Three directions follow from these results. First, dense temporal measurement: scoring compliance at every turn would allow fitting per-instruction decay curves and estimating how quickly each instruction fades. Second, a closed-loop evaluation: implementing the reinforcement ranking as a runtime system and measuring whether selective reinforcement outperforms uniform reinforcement in practice. Third, negation analysis: systematically comparing negated instructions with positive equivalents to understand when and why telling a model what *not* to do backfires.

7 Conclusion

We measured per-instruction retention in AI coding assistants and found that it varies by over an order of magnitude. Some instructions are retained when stated, some are already followed by default, and one is actively harmed by reinforcement. This heterogeneity is consistent across five model architectures and three levels of context pressure, and it replicates when the analysis is restricted to a single model with matched baseline and treatment conditions ($r = 0.80$).

The practical implication is that uniform reinforcement strategies waste most of their token budget. A posterior-derived ranking can identify which instructions to reinforce and which to leave alone. The per-instruction retention probabilities estimated here are the input that a Bayesian Knowledge Tracing system would need to operate. Building that system is the next step.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

Abril-Pla, O., Andreani, V., Carroll, C., Dong, L., Fonnesbeck, C. J., Kochurov, M., Kumar, R., Lao, J., Luhmann, C. C., Martin, O. A., Osthege, M., Vieira, R., Wiecki, T., & Zinkov, R. (2023). PyMC: A modern, and comprehensive probabilistic programming framework in Python. *PeerJ Computer Science*, *9*, e1516.

Chowdhury, B. D. (2026). Lost in the middle at birth: An exact theory of transformer position bias. *arXiv preprint arXiv:2603.10123*.

Corbett, A. T., & Anderson, J. R. (1994). Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*, *4*(4), 253–278.

Dongre, V., Rossi, R. A., Lai, V. D., Yoon, D. S., Hakkani-Tur, D., & Bui, T. (2025). Drift no more? Context equilibria in multi-turn LLM interactions. *arXiv preprint arXiv:2510.07777*.

Du, Y., Tian, M., Ronanki, S., Rongali, S., Bodapati, S. B., Galstyan, A., Wells, A., Schwartz, R., Huerta, E. A., & Peng, H. (2025). Context length alone hurts LLM performance despite perfect retrieval. *Findings of the Association for Computational Linguistics: EMNLP 2025*, 23281–23298.

He, Y., Jin, D., Wang, C., Bi, C., Mandyam, K., Zhang, H., Zhu, C., Li, N., Xu, T., Lv, H., Bhosale, S., Zhu, C., Sankararaman, K. A., Helenowski, E., Kambadur, M., Tayade, A., Ma, H., Fang, H., & Wang, S. (2024). Multi-IF: Benchmarking LLMs on multi-turn and multilingual instructions following. *arXiv preprint arXiv:2410.15553*.

Hoffman, M. D., & Gelman, A. (2014). The No-U-Turn Sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, *15*, 1593–1623.

Laban, P., Hayashi, H., Zhou, Y., & Neville, J. (2025). LLMs get lost in multi-turn conversation. *arXiv preprint arXiv:2505.06120*.

Leviathan, Y., Kalman, M., & Matias, Y. (2025). Prompt repetition improves non-reasoning LLMs. *arXiv preprint arXiv:2512.14982*.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., & Liang, P. (2024). Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, *12*, 157–173.

Mu, N., Lu, J., Lavery, M., & Wagner, D. (2025). A closer look at system prompt robustness. *arXiv preprint arXiv:2502.12197*.

Seyboldt, A. (2024). *Nutpie: A fast sampler for Bayesian posteriors*. https://github.com/pymc-devs/nutpie

Zhang, J., Yang, J., & Wang, K. (2025). Large language models as discounted Bayesian filters. *arXiv preprint arXiv:2512.18489*.