Not All Instructions Are Forgotten Equal

Tomás P. Korenblit

Universidad Nacional de San Martín, Buenos Aires, Argentina
tomaskorenblit@gmail.com

Abstract. For an autonomous LLM agent, a safety guardrail is just another instruction given at the start of a run, and as context grows, the agent keeps some instructions while dropping others. Because a mean compliance rate aggregates across instructions, it cannot reveal which one was dropped. We measure retention one instruction at a time by conducting 25-turn coding sessions on five instruction-following models and three open-source Python codebases. In these sessions, we plant twelve ordinary coding preferences as casual inline requests, score the resulting code with deterministic checkers on a 0–3 ordinal scale, and fit a Bayesian ordered-logistic model with hierarchical effects per decision type to 244 observations. Treatment effects span -2.4 to $+5.4$ on the log-odds scale (group-level standard deviation 2.1), showing considerable variation across preferences. Most preferences are already followed by default, and only three of twelve benefit from being restated (two under a single-model robustness check). Since these preferences stand in for guardrails, this unevenness is a safety concern. Because a guardrail must hold on every turn, aggregate compliance can remain high even after the one rule that matters is skipped. As a result, a deployed agent needs per-instruction monitoring to catch it.

Keywords: instruction retention , context rot , Bayesian inference , AI safety , LLM evaluation

No Todas las Instrucciones se Olvidan Igual

Abstract. Para un agente autónomo basado en un LLM, una salvaguarda no es más que otra instrucción dada al inicio de una ejecución, y a medida que el contexto crece, el agente conserva algunas instrucciones mientras descarta otras. Como una tasa de cumplimiento promedio combina todas las instrucciones, no puede revelar cuál se descartó. Medimos la retención instrucción por instrucción mediante sesiones de programación de 25 turnos en cinco modelos que siguen instrucciones y tres repositorios de Python de código abierto. En esas sesiones, plantamos doce preferencias de programación comunes como pedidos informales intercalados, puntuamos el código resultante con verificadores deterministas en una escala ordinal de 0 a 3, y ajustamos un modelo bayesiano de regresión logística ordinal con efectos jerárquicos por tipo de decisión sobre 244

observaciones. Los efectos de tratamiento van de −2,4 a +5,4 en la escala log-odds (desviación estándar a nivel de grupo de 2,1), lo que muestra una variación considerable entre preferencias. La mayoría de las preferencias ya se cumplen por defecto, y solo tres de doce se benefician de reiterarse (dos bajo una verificación de robustez con un solo modelo). Como estas preferencias sirven de sustituto de salvaguardas, esta disparidad es una preocupación de seguridad. Como una salvaguarda debe sostenerse en cada turno, el cumplimiento agregado puede mantenerse alto incluso después de que se omita la única regla que importa. Por eso, un agente desplegado necesita monitoreo por instrucción para detectarlo.

Keywords: retención de instrucciones , degradación de contexto , inferencia bayesiana , seguridad de la IA , evaluación de LLMs

1 Introduction

Language models are increasingly deployed as autonomous agents that run for hundreds of turns in a loop, calling tools with no human reviewing each step. An agent is governed by its instructions, which are the operational limits and safety guardrails it is given, usually once and at the start, that must hold for the whole run. Yet adherence to instructions erodes as the context grows, a phenomenon we refer to as *context rot*. Laban et al. (2025) measured average compliance drops of 39% across 15 models in multi-turn settings. The cause lies in the architecture, where attention is not uniformly distributed across the context window, and information in middle positions receives less weight than at the boundaries (Chowdhury, 2026; Liu et al., 2024). An instruction stated once and then buried under accumulating tool output competes for attention against everything that follows it, and in a deployed loop there is no user to notice and restate it.

This is a safety concern before it is a cost concern. When the model drops a preference, a chat user notices and restates it, whereas a deployed agent has no such supervisor, so a guardrail that decays does so silently while the agent keeps acting through tools whose consequences are often irreversible (Mu et al., 2025). We plant ordinary coding preferences, and a preference and a guardrail are the same kind of object (an instruction stated once and then left to compete for attention), so how well a preference survives is a measurable stand-in for how a stated guardrail would hold. Repeating every instruction each turn, the standard mitigation, multiplies prompt cost and still cannot tell an operator which constraint remains in force. Because safety is conjunctive, an aggregate compliance rate says nothing about whether the one rule that matters is among those that were dropped.

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 instructions are retained unequally, the mean hides the per-instruction differences a reinforcement policy would act on. An instruction that is already retained does not need reinforcement, and one that repetition harms should not receive it. A policy cannot tell these two cases apart without per-instruction estimates.

Bayesian Knowledge Tracing (Corbett & Anderson, 1994) is a framework for per-instruction estimation. Originally developed in education, BKT estimates the probability that a student has mastered a concept based on a sequence of observed responses. We adapt this framing, where the LLM is the student and its instructions are 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 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`) shows a marginal negative effect (its 94% HDI barely excludes zero), which most likely reflects a measurement artifact. Third, within the resolution of this design, model and codebase size explain little variance in retention. Because baseline data comes from a single model, we read this cautiously, because the design shows model-invariance but cannot establish that model effects are absent. A Bayesian model can then identify which instructions to reinforce and which to leave alone.

2 Related work

Instruction compliance degradation. As conversations grow, models follow fewer of their instructions. 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 are retained to different degrees.

Positional attention and prompt saturation. Compliance degradation arises from the attention architecture. Liu et al. (2024) identified a U-shaped attention curve, where 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, whereby compliance with any individual rule drops sharply as the number of guardrails in a system prompt increases. A model

therefore follows a rule less reliably the deeper it sits and the more instructions crowd around it.

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 how prompt repetition affects 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 Knowledge Tracing. BKT has tracked per-concept mastery probabilities in education for over 30 years (Corbett & Anderson, 1994). The per-instruction heterogeneity that makes BKT useful there also appears in LLM instruction compliance, where each instruction is a concept.

3 Method

3.1 Decision planting protocol

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

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, spanning 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 user phrases each instruction as a natural preference (e.g., "Important: when writing array operations for this project, always use NumPy broadcasting instead of for loops").

In the **baseline** condition, we inject the same files and issue the same task, but plant no preferences. The baseline lets us 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. Parsing

the same patterns each run keeps results reproducible. 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.

σ_β, the group-level standard deviation of treatment effects, measures how much instructions vary in their response to being stated. A large value means instructions benefit differently from reinforcement, the condition selective reinforcement requires.

We initially fit a full model including per-model and per-codebase intercepts. Both group-level standard deviations were small, with posteriors concentrated near zero ($\sigma_{\text{model}}, \sigma_{\text{codebase}} \approx 0.3$, 94% HDI upper bound below 0.7), so these

terms add no explanatory power. We therefore report the simpler per-decision model, whose diagnostics are clean (zero divergences, all $\hat{R} \leq 1.01$, minimum ESS = 1311). With baseline data from a single model (Section 6.3), per-model effects are in any case weakly identified.

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.

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, namely 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 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 was 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.

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		✓

4.3 Dataset

We ran 28 conversations in total, comprising 9 baseline and 19 treatment. Models were accessed through OpenRouter (Qwen, Gemini, Gemma, Nemotron) and the OpenAI API (GPT-5.4-nano), queried at each provider's default sampling settings (temperature not overridden), one generation per turn, capped at 2048 (Qwen) or 4096 tokens. Each score comes from that single generation. Not every conversation produced a scorable response at every test turn. The final dataset holds 244 compliance observations, each 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 at `https://github.com/korentomas/not-all-instructions`.

5 Results

The model converged cleanly, with zero divergences, $\hat{R} \leq 1.01$ for all parameters, and a minimum effective sample size of 1311. 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 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: $[2.94, 8.10]$), where 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 of near-zero baseline and retention when stated.

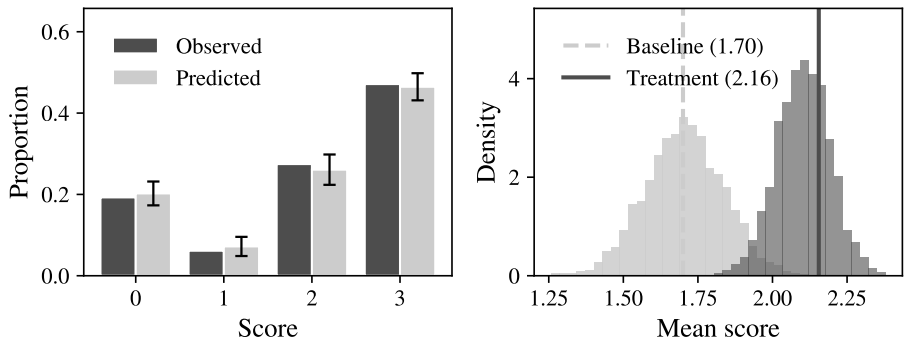

Fig. 1. Posterior predictive check. Left: the model reproduces the observed score distribution across all four ordinal categories. Right: the model captures the difference in mean compliance between baseline (1.70) and treatment (2.16) conditions.

A marginal negative effect. The only decision type with a negative estimate is `dependencies_no_new` ($\beta = -2.36$, 94% HDI $[-4.46, -0.03]$). Its magnitude is uncertain, since the 94% HDI only just excludes zero, though the negative direction itself is credible ($P(\beta < 0) = 0.98$). The baseline mean score is 2.25 (models naturally avoid adding imports), while stating "do not add new imports" lowers the treatment mean to 0.77. One hypothesis is that the explicit negation confuses the model's instruction following. But the checker for this type (`no_new_imports`) counts *any* import statement as new, including re-shown imports already present in the file (Section 6.3), so we suspect a measurement artifact here and do not read it as evidence of harm. Isolating the negation from the content (comparing "avoid new imports" with "use only existing imports") would settle it.

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). The models already follow these conventions, so restating them does not change the score.

5.2 Model and codebase effects

As described in Section 3.3, a model with per-model and per-codebase intercepts placed both group-level standard deviations near zero, so we report the per-decision model. Treatment-condition compliance is consistent across the five models (26B to 120B for the three with disclosed sizes) and three levels of context pressure (98K to 203K tokens), and no per-model or per-codebase effect is detectable at this resolution. With a single-model baseline and shrinkage priors, though, this supports model-invariance only weakly, and we cannot fully separate per-model retention (Section 6.3). One reading we can set aside, though, is that

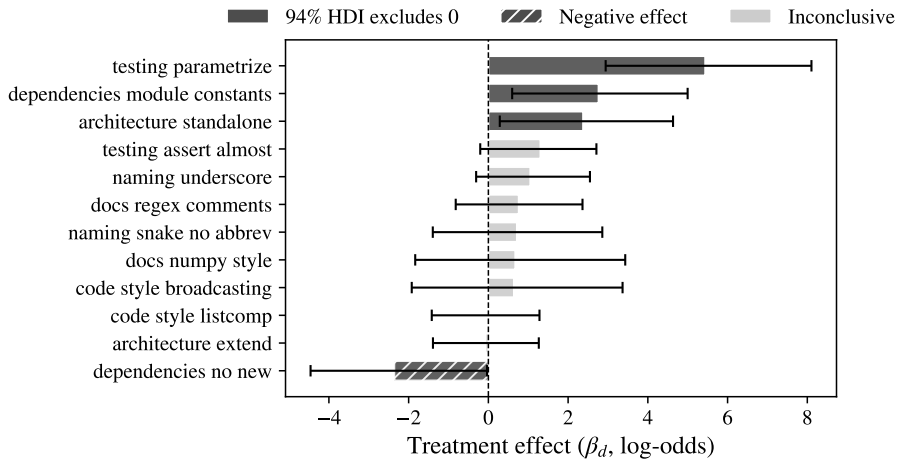

Fig. 2. Per-decision treatment effects (β_d) with 94% HDI. Dark bars: 94% HDI excludes zero. Hatched: negative effect. Light bars: inconclusive.

retention does not climb from the 26B to the 120B model, so the heterogeneity does not appear to be model capability under another name. If these differences were a capability effect, the largest model would comply most.

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).

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 with a marginal negative effect. A low gain may mean the model already complies, or that the checker saw nothing (a default-2 ceiling), neither of which licenses dropping a safety-relevant rule. It saves tokens on ordinary preferences but says nothing about which guardrails to drop.

6 Discussion

6.1 What determines retention

Retention tracks default behavior. Instructions that ask for something the model would not do by default are retained when stated, whereas those that match existing behavior show no treatment effect. The three decisions with clear positive effects (`parametrize`, `module_constants`, `standalone`) all have baseline

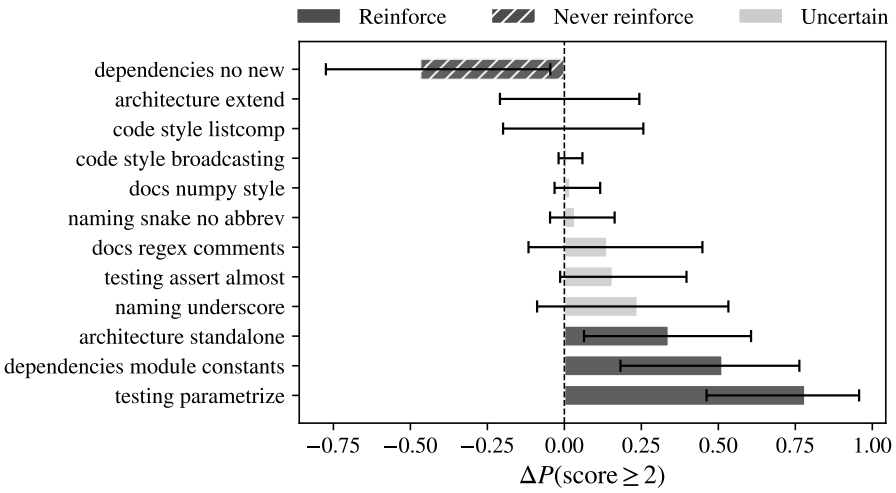

Fig. 3. Expected change in $P(\text{score} \geq 2)$ from reinforcing each decision, with 94% HDI. Dark bars: HDI excludes zero. Hatched: reinforcement is harmful.

compliance near zero. The decisions with no treatment effect (`broadcasting`, `numpy_style`, `snake_no_abbrev`) have baseline compliance near 3.0. The models' defaults already match these conventions, so restating them changes nothing.

The negative effect on `dependencies_no_new` is the only one in the study. We call it marginal, since its 94% HDI barely excludes zero, though its direction is credible ($P(\beta < 0) = 0.98$). What stays open is the magnitude, and whether it reflects negation framing or the checker artifact (Section 6.3). 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. It is unclear whether the negative treatment effect reflects something about the content of the instruction (import management) or its framing (negation). Which case holds determines the fix. If negation is the problem, reformulating as "use only existing imports" might produce a positive effect. If import management is inherently hard to retain, no framing will help. Distinguishing these hypotheses requires a controlled comparison that rephrases the same instruction with and without negation. We adopt this comparison in the future work below.

6.2 Robustness checks

Baseline data comes from Qwen only, so we refit the model on Qwen observations alone (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). Three of the strongest effects replicate, with `parametrize` ($\beta = 6.13$), `module_constants`

Table 2. Reinforcement priorities derived from posterior estimates. Gain is $\Delta P(\text{score} \geq 2)$ from reinforcement. The five skipped decisions are `broadcasting`, `listcomp`, `architecture_extend`, `snake_no_abbrev`, and `numpy_style`.

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
`5 other types`	$< +0.04$	crosses zero	Skip
`deps_no_new`	-0.47	$[-0.77, -0.05]$	Likely artifact

($\beta = 2.25$), 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. Two decision types retain 94% HDI entirely above zero in the Qwen-only data, down from three in the full model.

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).

6.3 Limitations

Baseline data comes from one model only (Qwen). The sensitivity analysis ($r = 0.80$) leaves this concern partly open. Future replications should include baseline conditions for every model tested.

The instructions we plant are ordinary coding preferences, not safety guardrails, and the test turns measure code style. None involves a harmful action taken or avoided. We read preferences as a proxy for guardrails insofar as both are user instructions competing for the same attention, but that link remains a hypothesis grounded in the shared mechanism. If anything, the proxy may overstate guardrail fragility, because actual guardrails often sit at position zero in the system prompt, the most stable position (Liu et al., 2024), and are reinforced in training, so our finding applies most directly to newly stated, in-context constraints. One direct test remains open, namely whether a stated safety constraint decays the same way, and whether the decay then lets a harmful action through. All of our evidence also comes from Python coding sessions. Whether the same per-instruction structure holds for tool-use agents under safety constraints in other domains is untested.

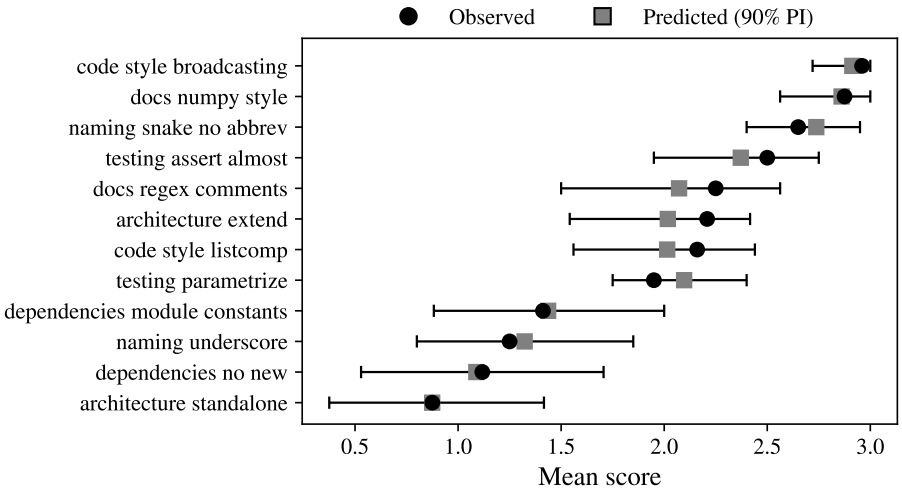

Fig. 4. Per-decision posterior predictive check. Observed means (dots) fall within the 90% posterior predictive interval for all 12 decision types.

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

Compliance was measured at turns 20–25 only. That window gives a retention snapshot, not a decay curve. We cannot estimate per-instruction decay rates or say at which turn compliance starts to drop. Fitting forgetting curves would mean scoring every turn, which in turn means designing prompts that let the model 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%, specifically `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.

This paper does not implement Bayesian Knowledge Tracing. It measures the per-instruction heterogeneity that would make a BKT-based reinforcement system useful. Such a system would reallocate token budgets through a feedback loop and track compliance as a run unfolds. We leave it to future work to build and evaluate.

6.4 Future work

Several directions follow. The most direct is matched baseline and treatment conditions for every model, which would estimate per-model effects directly, whereas the single-model check we ran can only infer model-invariance. The checkers, though reproducible, have construct-validity limits, so we would pair them with a blinded panel of cross-family LLM judges and report checker–judge and inter-judge agreement. To separate phrasing from content, we would re-run the instructions whose effect excludes zero (those with 94% HDI above zero) under negated paraphrases, and the negated one under positive paraphrases. Scoring every turn would fit per-instruction forgetting curves and decay rates. Finally, running the posterior-derived ranking as a runtime policy would test whether selective reinforcement beats uniform under a fixed token budget, and traces from deployed coding sessions would test whether the structure holds outside the harness.

7 Conclusion

We measured per-instruction retention in AI coding assistants and found treatment effects spanning nearly eight log-odds units, from instructions the model adopts only when told, to ones it already follows by default. The pattern appears across the five models and three levels of context pressure tested, and the heterogeneity replicates on a single model with matched baseline and treatment conditions ($r = 0.80$). With baseline data from one model, though, we cannot fully separate per-model from per-instruction effects. We read the model-level result as no detectable effect at this resolution and make no claim of established invariance.

Of 12 instruction types, only 3 show a reinforcement gain whose interval excludes zero in the full model (two when restricted to the single model with matched baseline). A uniform policy therefore wastes tokens on instructions that already comply. The retention probabilities estimated here are what a Bayesian Knowledge Tracing system would act on. We leave that system to future work.

Acknowledgments. I thank my partner, Lara Rodrigues La Moglie, for her boundless love and unwavering support; my caring family, for the opportunities that guided me all the way here; my alma mater, the Universidad Nacional de San Martín, within whose halls I still live; and all my teachers, and the teachers of Argentina, who live each day to inspire others. I also thank the anonymous reviewers, whose comments sharpened this work, and acknowledge the support of a BlueDot Impact Rapid Grant.

Use of large language models During the preparation of this work the author used Claude in order to improve the language and readability of the manuscript. After using this tool, the author reviewed and edited the content as needed and takes full responsibility for the content of the publication.

Disclosure of Interests. The author has 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