

Verified Prompt Memory in a Frozen Language Model: A Paired Arithmetic Case Study

Daniel Schmitter

Abstract

External verification can control which generated examples enter a language model’s prompt memory, but a gain from that memory is not necessarily progressive self-improvement. We analyze a retained 120-problem multiplication experiment with a frozen Qwen2.5-VL 7B model. Verified memory answers 59 problems correctly, compared with 37 without memory and 16 with unfiltered generated examples. Paired accounting shows 36 problems solved only by verified memory and 14 only by the no-memory control. However, verified accuracy changes from 30/60 in the first half to 29/60 in the second, providing no evidence of compounding improvement. Inspection of the implementation establishes that examples are selected by recency, not nearest-neighbor retrieval, and that the external check verifies the parsed final integer rather than every reasoning step. We provide an explicit memory-update model, complete outcome accounting, and deterministic parser tests. Missing prompts, response traces, model digests, execution-error logs, and randomized run order limit reproducibility and causal attribution. The result is an exploratory observation about externally filtered context, not a new general self-improvement algorithm or a spline-memory result.

1 Introduction

A frozen language model can change its behavior when its context changes. An external store of earlier successful solutions is one way to make that context depend on experience without updating weights. The resulting system contains at least three mechanisms: generation, a criterion for accepting experience, and selection of stored experience for the next prompt. Success of the complete system does not establish that its neural model has learned a new persistent capability.

We examine a small arithmetic experiment because it supplies an exact external final-answer check. The central empirical questions are whether filtered examples outperform absent or unfiltered examples on the retained stream, and whether performance rises as the store grows. The contribution is a source-reconciled paired case study and a precise boundary between answer-verified prompt memory and recursive self-improvement. No claim is made that example filtering or in-context learning is new.

2 Related work

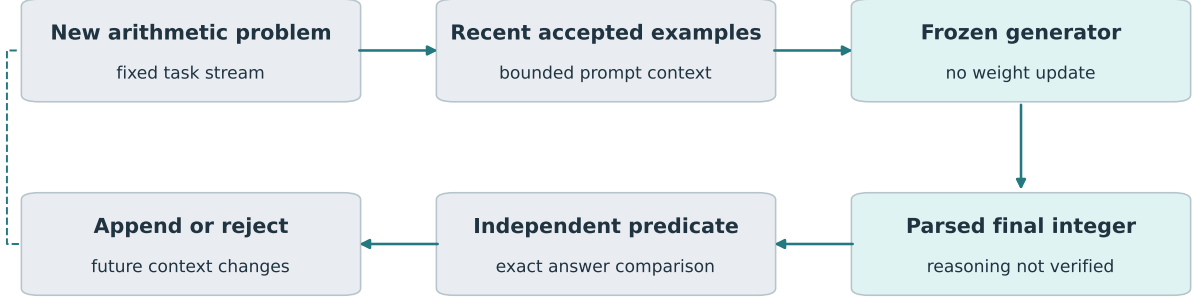
Reflexion uses feedback and episodic text memory to improve subsequent language-agent behavior without weight updates [1]. Self-Refine uses a model’s own feedback to revise its outputs [2]. The present implementation instead applies an exact arithmetic predicate to the final parsed answer and stores successful generated text. It neither constructs verbal reflections nor iteratively repairs a failed answer. These distinctions identify a simple ablation, not a new replacement for those established methods.

The broader operator-spline project motivates concern with what a memory preserves. Here the stored object is text, not spline coefficients, a Gram matrix, or a protected continuous function. An append-only store preserves accepted records while changing which records are visible in a finite prompt. Therefore retention of stored bytes, preservation of an earlier answer, and improved future accuracy remain different properties.

3 Architecture and information flow

The generator is frozen, but the system is stateful because accepted examples alter later prompts. A recency selector exposes a bounded part of the accumulated store. The verifier checks a parsed final integer and controls admission to that store. It does not check the intermediate explanation, update the neural weights, or guarantee that an accepted example helps another problem.

Verification controls the memory boundary



Read left to right above, then right to left below. Arrows show data or computation flow. A later iteration uses the retained state.

Figure 1: Verification controls the memory boundary. Boxes distinguish supplied information, fitted components, and the quantity evaluated. Arrows show computation or data dependence, not a newly trained deep network.

The paired results and chronology answer different questions. The aggregate comparison supports more useful context in the verified arm on this stream. The first/last-half comparison does not support an improving capability curve. A larger store can coexist with a constant-size prompt and unchanged or worsening accuracy. A clean fixed few-shot control is missing, so online accumulation cannot be isolated from simply having correct demonstrations.

4 Generator, verifier, and memory dynamics

Let F be a frozen generator, x_t the next integer pair, M_t the accepted example list, and R_k the function selecting its most recent k entries. The generator receives an instruction to give brief steps and a final integer marker. The memory policy applies the following update, where V compares the parsed final answer with exact integer multiplication.

$$o_t = F(x_t, R_k(M_t)), \quad V(x_t, o_t) = \mathbf{1}\{\text{parse}(o_t) = a_t b_t\}, \quad (1)$$

$$M_{t+1} = \begin{cases} M_t \parallel (x_t, o_t), & V(x_t, o_t) = 1, \\ M_t, & \text{otherwise.} \end{cases} \quad (2)$$

The no-memory arm sets the contextual example list to empty. The unfiltered arm appends every response with a parseable integer, whether correct or not. Thus the unfiltered arm is not literally every attempted response: unparseable outputs do not enter its list. None of the arms changes F 's parameters. The verified store begins empty, so the first example must be generated without stored experience.

Proposition 1 (accepted-record invariant). If the verifier computes the target integer correctly and parsing is deterministic, every appended verified record has a correct parsed final integer for its stored question. *Proof.* The empty list satisfies the property. An update either leaves the list unchanged or appends an item satisfying the predicate. Induction proves the invariant. This says nothing about intermediate statements in that item, the generator's next answer, or performance on a different task.

A recency window is bounded even when the underlying list grows. Older accepted items remain stored but may no longer appear in the prompt. The code's introductory comment describes nearest retrieval and compounding accuracy; the implementation uses a final list slice and the retained outcomes do not exhibit that trend. We use the implementation and results, not those aspirational comments, to define the method.

5 Experimental methods

The saved artifact names qwen2.5vl:7b, three-digit multiplication, and three aligned Boolean arrays of length 120. It records 59 accepted examples. The source uses a local Ollama generation endpoint, temperature zero, and a maximum of 220 generated tokens. Its defaults are seed zero and four recent examples. Because the saved artifact omits the command line and model digest, those defaults describe the available runner rather than a cryptographically established configuration of the historical execution.

The generator samples positive integer operands from a half-open interval. Under the three-digit default, the upper bound excludes 999. The same generated problem list is passed to all three arms in a fixed order: verified, no memory, then unfiltered. Exceptions are caught and converted to empty responses, which count as incorrect. Their number and type are not retained separately. Full prompts, raw model responses, operand pairs, timing, and request failures are absent from the compact result file.

Our analysis does not call the model. We count all outcomes, split the existing chronological arrays into their first and last sixty attempts, and tabulate paired agreement. The half split is a descriptive diagnostic of the claimed progressive trend, not a newly frozen confirmatory endpoint. We do not select the best rolling window. The supplementary file also includes all six consecutive twenty-attempt blocks.

The stream is adaptive: accepted earlier outputs affect later prompts. Consequently, the 120 outcomes are not independent replications of a fixed treatment. Fixed arm order also leaves possible service or model-state effects unrandomized. We report exact descriptive counts rather than attaching an iid significance test or binomial confidence interval that would conceal these design limitations. Deterministic parser fixtures isolate what the acceptance predicate actually checks.

6 Results

Table 1: Complete saved outcomes. Each arm attempts the same 120 problem positions.

Arm	Correct / 120	Accuracy	First 60	Last 60
Verified memory	59	49.17%	30	29
No memory	37	30.83%	18	19
Unfiltered memory	16	13.33%	7	9

Verified context exceeds the no-memory arm by 22 correct answers, or 18.33 percentage points, and the unfiltered arm by 43 answers, or 35.83 percentage points. These are system-level differences on the saved stream. They are not estimates of a universal verification effect: the prompts differ in content, length, and the histories selected by earlier outcomes.

Table 2: Paired counts relative to verified memory. The four cells sum to 120 in each row.

Other arm	Both wrong	Only other correct	Only verified correct	Both correct
No memory	47	14	36	23
Unfiltered	51	10	53	6

The paired table makes two facts visible. Verified memory corrects many positions missed by a control, but it also loses positions that the control solves. It is therefore not a monotone per-problem improvement. The first/last-half result likewise does not support increasing competence as more examples accumulate: verified accuracy changes from 50.00% to 48.33%. The twenty-attempt counts are 11, 8, 11, 12, 9, and 8.

Parser fixtures expose the verification boundary. A comma-formatted final integer is accepted as an integer; a response without any number is unparseable. If text after the final marker contains another number, the last number is selected. Thus the string "#### 123 then check 9" parses as nine. A correct final integer following an invalid derivation still passes. The audit cannot retrospectively classify such reasoning defects because the original response texts were not saved.

7 Discussion

The simplest interpretation is that externally filtered examples supplied more useful context than the two tested alternatives. A clean fixed few-shot set is an important missing control: the saved study does not

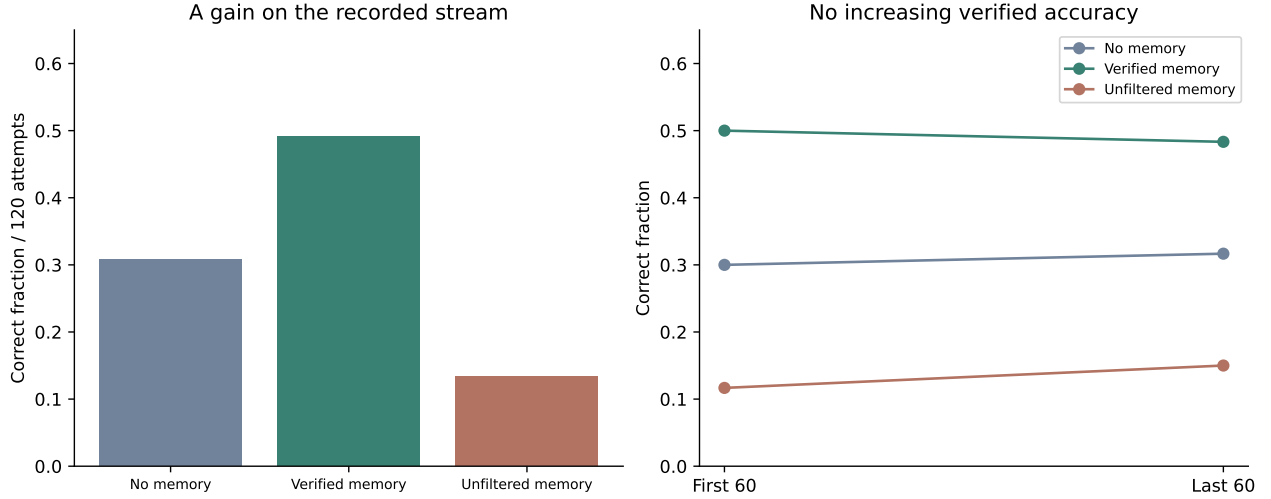


Figure 2: All saved attempts, with no new model calls. Aggregate verified accuracy is higher, while the chronological half comparison does not increase. Connecting two descriptive halves is not a fitted learning curve or a confidence interval.

distinguish continual accumulation from access to a few correct demonstrations. Nor does it isolate answer correctness from rationale quality, prompt length, example order, or recency. An exact multiplication routine could also answer this task directly, so the experiment is not an efficient arithmetic application.

The memory invariant is nonetheless a useful design concept when a task has an inexpensive independent check. It constrains what enters a store. It does not imply no forgetting by the neural model, reliable retrieval, autonomous discovery of new tasks, or recursively increasing problem-solving power. Those mechanisms are absent here. Calling the store "never forget" would describe retained text only, not preserved performance.

The supplement preserves the complete original arrays, paired and chronological accounting, parser tests, and source hashes. No prompts are reconstructed as though they had been retained, and no new model execution is substituted for the historical run. The limited provenance means this manuscript is a transparent exploratory case study, not a fully reproducible confirmatory benchmark. Further publication claims would require a independently logged evaluation; they are not manufactured by editorial revision.

8 Application boundary and research implication

This is an architecture for filtered experience, not a demonstrated recursive improvement loop. Its strongest invariant is about stored records. A self-improving system would additionally need persistent gains on new tasks and a controlled explanation of what mechanism improved. The arithmetic case study makes that distinction testable without claiming a spline mechanism where none is present.

9 Conclusion

Verified prompt memory improves aggregate accuracy on one saved arithmetic stream, while unfiltered self-generated context performs worse than no memory. The same record contains no progressive accuracy gain and no spline mechanism. An external acceptance rule can preserve answer-verified examples, but that invariant is weaker than correct reasoning, stable task performance, or recursive self-improvement. Keeping those claims separate makes the observed result both useful and scientifically interpretable.

References

- [1] N. Shinn et al. Reflexion: Language Agents with Verbal Reinforcement Learning. NeurIPS, 2023.
<https://arxiv.org/abs/2303.11366>
- [2] A. Madaan et al. Self-Refine: Iterative Refinement with Self-Feedback. NeurIPS, 2023.
<https://arxiv.org/abs/2303.17651>