# TRANSFIVER: HUMAN–AI CO-INFERENCE THROUGH A SHARED EDITABLE STATE

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

For people and AI systems to reason together, people need to inspect and correct the information the system uses in subsequent computation. We introduce the TRANSparent Framework for Interactive, Verifiable, Editable Representation (Transfiver), an architecture for human–AI co-inference through a shared editable state. Its central idea is that interaction-specific information is maintained in a single persistent state that both the model and the human update.

Transfiver distinguishes two modes of state evolution. An implicit stream update requires the model to interpret ongoing interaction and determine whether new information revises an existing state item or creates a new one. In an explicit directed edit, a human inspects a record and changes it. Both act on the same underlying state, so a human correction changes the state that subsequent computation reads, rather than adding another instruction or separate record.

We evaluate two implementations that test complementary aspects of this design. Record-store experiments show two results. A retracted record stays out of the prompt when the system rebuilds its input for the next question. Its text stays readable for inspection and restoration. A learned model reads editable records while it generates each answer, so retracting the record that holds an answer removes that answer. These results support shared editable state as a basis for human–AI collaboration.

## 1 INTRODUCTION

Consider an assistant helping with a project. A person changes the recorded project lead, and the assistant acknowledges the correction. Yet on the next task it uses the previous lead's name. The person must repeat the correction without knowing whether it will hold this time. A correction must change the information the assistant uses on subsequent tasks.

We propose that a person and a model should work from one shared editable state. The information the person inspects and revises must be the information the model uses for subsequent computation. Recent work studies answering with the latest fact (Patel, 2026). This requirement extends beyond that goal. It concerns where the system retains information. It concerns how people can change it. It concerns whether those changes affect later behavior. Corrections provide a concrete test of this architecture because their intended effect can be compared with the model's subsequent answers.

When something goes wrong, a person usually cannot tell. Consider an assistant that works inside web applications on a person's behalf. A person asks it to order a loaner laptop from the company's service catalogue. The assistant clicks and types its way through the site and then reports: *Task completed successfully! I have ordered 5 loaner laptops.* No order was placed.

This case is one of 1,232 in a public dataset of such attempts (Lù et al., 2025). Annotators checked each site to see whether the requested action had happened. On the attempts that failed and ended with a message to the person, nine in ten of those messages reported no problem.

A second model can assess whether the task was completed by reading the execution record. The published judges detect most failures (Table 1), but detect fewer among attempts ending with an assistant message. Thus neither the assistant's report nor an external assessment reliably establishes whether the requested action occurred.

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

| What the person is given | Of 100 failed runs, how many it catches | Of 100 successful runs, how many it calls failures |
|---|---|---|
| The agent's closing message to the person | 11 | 2 |
| A second model that reads the whole record | 81–86 | 12–24 |
| on runs that end with no message | 87–91 | 13–27 |
| on runs that end with a message | 63–70 | 10–20 |

Table 1: Failure detection against expert judgements, in percent. The first row is scored only on runs that end with a message to the person. Ranges span six published judge evaluations, each covering a subset of the data.

A report of completion does not end when the person reads it. What the assistant recorded about the task is what it reads on the next one. An assistant that reports an order it never placed carries that claim forward as a stored fact. Even a correct failure verdict gives the person no operation for changing that record. A person can issue another instruction. Whether it holds depends on whether the stored record changed, and an instruction that only reaches one prompt leaves that record as it was. The architectural question is therefore how to give people access to the state that determines subsequent behavior.

Transfiver is an architecture organized around one persistent, editable state $S_t$. The model reads this state to answer, and updates it as interaction continues. A person can inspect and correct the same stored information. Section 3 defines the requirements this shared state must satisfy. Interaction-specific information lives in this state. Both parties update it. The model updates it implicitly, by folding an ongoing interaction into it. A person updates it explicitly. The person stops the use of a stored record, restores its use, or rewrites its content. When a person corrects a stored fact, subsequent answers must use the corrected content. When a person stops a record from being used, that decision must remain in the state until the record is restored or changed again. We call this operation *retraction*. The record is marked as not to be used for current answers. Its text is kept. A person can read it or put it back into use.

Figure 1 illustrates why the same state must serve computation and human editing.

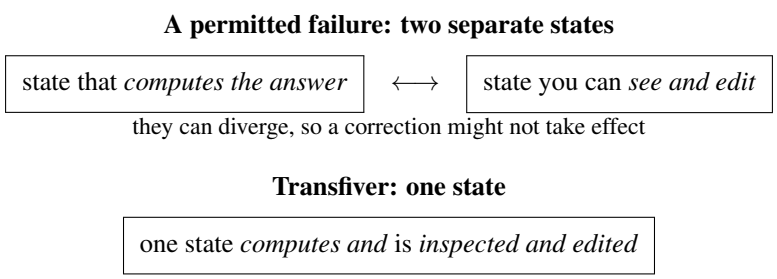


Figure 1: Transfiver uses one state for computation and human editing. A correction changes the state used for subsequent answers.

Transfiver expresses this requirement as follows:

> Memory, reasoning, and output are not divided. The state underlying what a person sees and corrects is the state from which the system computes, updates, and answers.

Transfiver may be embedded within a larger system, provided that it remains the sole persistent interaction-specific state through which prior interaction can affect future computation.

We evaluate two implementations of these requirements. A record store with a frozen language-model reader tests whether edits persist across questions while preserving historical access. A learned model tests whether editing a particular stored record changes the answer computed from it. Both implementations undergo save–reload checks.

**Contributions.**

1. We propose Transfiver, an architecture in which the model and the person update the same persistent state. The state a person inspects and edits is the state the model uses for subsequent computation.
2. We build a learned model that answers from the same state a person can inspect and edit.
3. We test the connection between state edits and subsequent behavior: persistence and historical access in the record store, and answer dependence and restoration in the learned model.

## 2 Related Work

Learned external memory, fast weights, test-time training and Titans provide self-updating computational state (Graves et al., 2014; Santoro et al., 2016; Ba et al., 2016; Sun et al., 2025; Behrouz et al., 2025). Transformers and state-space models retain sequence information (Vaswani et al., 2017; Gu & Dao, 2024). RAG, RETRO, MemGPT, Larimar and memory layers expose or address stored records (Lewis et al., 2020; Borgeaud et al., 2022; Packer et al., 2024; Das et al., 2024; Berges et al., 2025). Transfiver additionally requires the editable object to be the sole persistent interaction-specific computational state. MemGPT exposes editable memory blocks, but its manager acts on context buffers. Our LongMemEval experiments test what happens when a record is removed from one turn's context while remaining available for later retrieval. Larimar attaches an episodic module beside the model.

Supersede trains agents to answer from current rather than stale facts (Patel, 2026). Transfiver additionally makes a user's correction to a selected record an operation on the state subsequently read. Parameter editing (Meng et al., 2022; 2023) instead changes model weights. Our deployment distinction is between fixed shared parameters and changing interaction state.

Concept bottlenecks, persistent concept interventions and self-explaining models connect interpretation to intervention (Koh et al., 2020; Steinmann et al., 2024; Alvarez-Melis & Jaakkola, 2018). Transfiver additionally requires that the editable state retain all information from past interactions that can affect future answers, and that edits can be reversed. Causal abstraction provides a standard for testing the claimed interpreted variables (Geiger et al., 2025), while LongMemEval supplies long-term-memory tasks (Wu et al., 2025).

## 3 What Transfiver requires

The architecture places two requirements on the shared state $S_t$. It must contain all information retained from the interaction that can affect later behavior, and its editable records must participate in computation. Model parameters $\theta$ are learned before deployment and remain fixed during use. A directed edit changes a record at a supplied address. A stream update requires the model to determine where incoming information belongs.

Figure 2 shows how the person and model work with this shared state.

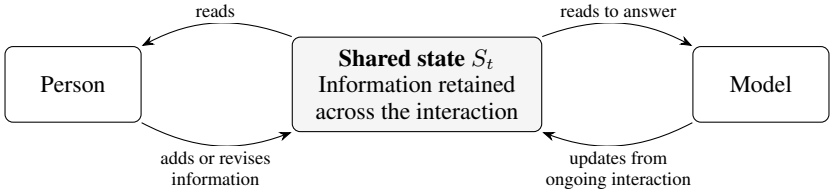


Figure 2: Working through the shared state. The person inspects and revises records. The model reads them to answer and updates them as interaction continues.

Table 2 summarizes the concepts used below.

| Concept | Meaning in this paper |
|---|---|
| Shared state $S$ | The persistent information a person inspects and edits is the same information the model uses in subsequent computation. |
| Parameters $\theta$ and state $S$ | Parameters are learned before deployment and remain fixed during use. Interaction updates the state. |
| Directed edit | A person selects a record and changes its content or whether it may be used. Experiments supply the record and replacement from benchmark annotations. |
| Stream update | Without a supplied storage address, the model must decide whether an incoming event updates an existing record or creates a new one. |
| History sufficiency | With the same parameters and saved state, the same future inputs induce the same distribution over behavior, regardless of earlier history. |
| State intervention | Change a supporting record and check its effect on the answer. Change another record as a control, then undo the edit to check restoration. |
| Retraction | Stop using a record for current answers while retaining its content for inspection and possible restoration. |

Table 2: Core concepts and operations.

### 3.1 History sufficiency

If two interaction histories leave exactly the same saved state, the same future input should induce the same distribution over future behavior. Otherwise, something outside that state still remembers a difference between the histories. We call this requirement *history sufficiency*.

Let $H_t = (x_1, \ldots, x_t)$ denote everything that has happened up to time $t$. We denote the saved state obtained from history $H_t$ by $S(H_t) = S_t$. The shared parameters $\theta$ may contain general abilities learned before use, but they may not quietly store facts that belong to one user's interaction history.

Let $c$ be a future sequence of inputs. We write $F_\theta(H_t, c)$ for the deployed system's behavior after history $H_t$ under those inputs, and Law for its probability distribution. The requirement is

$$S(H_t) = S(H'_t) \Longrightarrow \mathrm{Law}(F_\theta(H_t, c)) = \mathrm{Law}(F_\theta(H'_t, c)) \quad \text{for every admissible continuation } c. \tag{1}$$

A Transfiver step reads the saved state and the new input, then produces an answer and the updated state:

$$(S_{t+1}, y_t) = \mathcal{M}_\theta(S_t, x_t). \tag{2}$$

The state update is denoted by $S_{t+1} = T_\theta(S_t, x_t)$. During use, this update changes $S$ while $\theta$ remains fixed. A directed edit changes a selected record in $S$ directly. Any conversion of input or output formats must retain no interaction history of its own.

To inspect and edit the state, a person is shown its records and which records contributed to an answer. The view may sort or filter them, but each displayed record must identify the stored location the model reads. Editing it changes that location. The view need not expose raw tensors.

### 3.2 State restoration

Save $(\theta, S_t)$, restore it in a fresh process, and give the same next input. Under matched randomness, the restored system should reproduce the same next state and pre-decode output, and induce the same distribution over future behavior.

The audit tests history sufficiency on the evaluated continuations. A behavioral difference under matched inputs and randomness reveals information missing from the saved state or a failure to restore it correctly.

Restoring a saved state differs from restoring a record's use. The former reloads the complete snapshot, including its log and clock. The latter only re-enables a retracted record. The history of the edit remains.

**Definition** (Nothing persistent outside $S$)**.** Every interaction-specific persistent variable that can affect the distribution of future outputs is a member of $S_t$ and is included in serialization. Step-local

queries, activations, and fresh random draws may be temporary, but interaction-specific information may not persist through an unlogged cache.

### 3.3 State interventions

Saving the state is not enough: the editable information must also participate in answering. We therefore change a record that supports an answer and ask the same question again. We repeat the operation on a different record to check whether the answer changes simply because something in the state was edited. Finally, we undo the edit and ask whether the original answer returns.

We measure retraction by counting how many previously correct answers become incorrect when the supporting record is retracted. Applying the same operation to other records tests whether correct answers are lost merely because some record was retracted. For rewriting, we count how often the answer matches the replacement value, separately from how often the original answer remains. Restoring the edited record tests whether the answer returns. Reloading the saved state tests whether the same answers are reproduced.

## 4 Experiments and Results

The experiments test two requirements of a shared editable state: edits must persist across questions, and the edited information must participate in answer computation. A record-store experiment tests persistence, historical access and restoration on LongMemEval. A learned-model experiment tests whether answers depend on the particular records that a person can edit. Save–reload checks in both implementations test whether the saved state preserves the evaluated behavior.

Evaluation code applies the edits to records identified by the benchmark annotations. LongMemEval questions form the development set. The held-out MQuAKE-CF edit pairs are absent from training.

### 4.1 Persistent edits and access to earlier information

This experiment holds the system and the questions fixed and varies only where the correction is written. The system stores conversation records and selects four of them into the language model's prompt for each question. A person corrects one outdated record. The correction goes to one of three places. It can exclude the record from a single prompt. It can delete the record from the store. It can retract the record in the store. We then rebuild the input, ask the same question again, and measure whether that record is selected into the prompt and whether the answer is correct. Retraction and deletion both change what is stored. Retraction additionally keeps the text. This lets us measure what retaining earlier information buys.

We use all 44 LongMemEval knowledge-update questions whose annotated records identify both the old and current values (Wu et al., 2025). The same trained ranker selects four records for each language model under every condition.

**What changes in each condition.** All conditions concern the same annotated record containing an outdated value. They differ in whether they change an instruction, a single prompt, or the stored record.

| Condition | What is done to the record | Why it is here |
|---|---|---|
| No action | Leave the record unchanged | Baseline answer accuracy |
| Instruction in the prompt | Keep the record and instruct the model to ignore it | Whether an instruction prevents use |
| Input changed for this question | Exclude the record from this prompt only | Whether a one-time change persists |
| Record deleted | Remove the record from storage | Persistence without retaining content |
| Record retracted in the state | Disable selection for current answers; retain the content | Persistence with historical access and restoration |

When a record is removed, the next record in the same ranking fills its place, keeping the input at four records. Answers count as correct when they contain the reference string or a listed variant.

After the correction we rebuild the input and ask the same question again, adding no new conversation event. We also test historical access by allowing retrieval of retracted records and asking for the evidence containing the earlier value. In a separate restoration test, we re-enable the record and repeat the current-value question. Table 3 reports answer accuracy and whether the earlier record is retrieved in each setting.

| Intervention | Correct answers | | Is the record retrieved again? | | |
|---|---|---|---|---|---|
| | at once | next question | current question | historical question | after restoration |
| No action | 19–23 | — | — | — | — |
| Instruction in the prompt | 18–22 | — | — | — | — |
| Input changed for this question | 27–31 | 19–23 | 31 | 0 | — |
| Record deleted | — | 29–31 | 0 | 0 | not possible |
| Record retracted in the state | 27–31 | 27–31 | 0 | 31 | 31 |

Table 3: Edit persistence and access to the outdated record on 44 questions. Correct-answer counts span four readers (three for deletion). Retrieval columns count whether the record was selected into the prompt, not whether the answer was correct. Dashes indicate no separate result reported.

Excluding the outdated record from one prompt improves the immediate answer. The record is selected again on the next question. This happens in 31 of the 44 questions. Correct-answer counts return to the no-action baseline for every reader. Retraction prevents this reselection in all 44 cases and preserves the immediate improvement. Deletion gives the same current-value accuracy as retraction for the three readers tested. These results show that persistence depends on changing the information retained between questions.

Retraction also preserves access to the earlier information. The ranker selects the outdated record whenever it is allowed to. This happens on the same 31 of the 44 questions in every such condition. A prompt-only exclusion leaves the store untouched. A historical question permits retrieval of retracted records. Restoring the record's availability returns it to use. Deletion cannot retrieve it at all. Retrieval does not guarantee a correct answer: three readers identify the earlier evidence in 14–17 of those 31 cases, and Yi-1.5-9B in 2.

The same persistence comparison applies to rewriting a value. A replacement shown for one question disappears from the next prompt. A replacement written into the store remains available. Repeating the replacement in every prompt produces the same correctness outcomes as the stored edit.

### 4.2 Answer computation depends on editable records

Storing a correction and using it are separate properties. We test the second with a learned model that reads editable records during answer generation. Suppose answers depend on those records. Disabling a supporting record should then affect its answer more than disabling another record. Rewriting its content should change the value generated. These interventions test the computational role of the shared state.

The model has 2,560 memory slots, each a storage location for a record and the learned vectors used to find and read it. The language-model backbone is a frozen Qwen2.5-7B-Instruct. A learned mechanism assigns read weights to the slots for the current question. These weights determine each slot's contribution to a distribution over tokens copied from the stored text. A learned gate combines this copy distribution with the backbone's output distribution. The records themselves are not inserted into the question prompt.

**What is learned and what a person edits.** The write and read/copy modules are trained on released MQuAKE rewrite pairs (Zhong et al., 2023), with subject–relation overlaps with the Fact-Consolidation evaluation context excluded. The backbone stays frozen and all parameters are fixed during evaluation. Incoming records are stored in separate slots.

Each slot holds the record's text, learned vectors for finding and reading it, and an availability flag controlling whether it may be used. The complete state is saved and reloaded. We apply two edits:

- *Retraction:* clear the flag so the record no longer contributes tokens to the answer, while keeping its content.
- *Rewriting:* replace the stored text at the same location and recompute the token-derived keys used for copying. The slot's learned key and value vectors remain unchanged.

Both operations change fields used by the next answer without retraining the model. We evaluate answers with the model selecting the record itself and with a supplied record address. In the latter condition, called *record named* in the tables, the read and copy mechanisms use the specified record. Benchmark annotations supply this address. This separates record selection from generation given the selected record.

**Dependence on the supporting record.** The hypothesis is that the model computes its answer from the record a person can edit, rather than from the other records or from the frozen backbone. If it holds, retracting the record that holds the answer should remove that answer, and retracting any other record should leave it in place. FactConsolidation from MemoryAgentBench (Hu et al., 2026) supplies 455 record lines and 66 questions that can be linked to stored old/current pairs. Six independently trained seeds retain every line without overwriting. Held-out MQuAKE-CF supplies 288 lines and 144 questions across 37 relations, evaluated with six seeds. Answers are generated greedily. An answer counts as correct when it contains the released answer or one of its aliases, after spacing and punctuation are removed. The model often answers with the surrounding phrase. It rarely answers with the value alone. One answer reads "of India." for a stored value of "India". Answer counts use this rule throughout. Counts of a written replacement use exact match, because the claim there is that the stored text comes out unchanged.

For each question, we retract the slot containing the current answer and ask the same question again. Of the answers that were correct before this operation, 203 of 209 on FactConsolidation and 489 of 494 on MQuAKE-CF are no longer correct (Table 4).

| What is counted | FactConsolidation | Held-out MQuAKE-CF |
|---|---|---|
| *Counts pooled over training seeds; the seed count is in parentheses* | | |
| Questions asked, and how many seeds | 396 (6) | 864 (6) |
| Answered correctly before any edit | 209 | 494 |
| of those, lost after retracting that record | 203 | 489 |
| A written replacement comes out exactly | 95 of 330 (5) | — |
| A written replacement is removed again by retraction | 54 of 54 (3) | 143 of 144 (3) |
| Answers unchanged after saving and reloading | 198 of 198 (3) | 432 of 432 (3) |
| *Share of answers that contain the value* | | |
| The stored value, with the record named | .917 | .884 |
| with no record named | .528 | .572 |
| The value it replaced, with the record named | .000 | .008 |
| with no record named | .071 | .093 |
| *Named record altered before the question* | | |
| Named record retracted | .000 | .000 |
| A different record named instead | .005 | .000 |

Table 4: Learned-state results over six training seeds. Counts in the upper block are answers, except the rewrite row, which counts replacements emitted exactly, and the reload row, which counts answers unchanged after saving and reloading. Retraction rows count only answers that were correct beforehand.

To distinguish dependence on the supporting record from general sensitivity to edits, we repeat retraction with four control records. They exclude both the current and superseded records. One control is an unrelated record. One is another highly weighted record. Two more are the records whose read weights are closest to those of the current and superseded records. Each control loses at most one correct answer per seed, supporting dependence on the selected facts. The record holding the answer carries several times more read weight on average than any control record. A read that finds the right record should behave this way.

**Separating record selection from answer generation.** We apply the supplied-address test to HaluMem (Anonymous, 2026), which provides 82 questions over 1,594–1,914 memory lines per batch. We compare the answer record with the model's own selection, a different record, the retracted answer record, and no state. Generated answers are scored by reference-value containment. Some replacement records state both values, for example "changed from A to B". We therefore also measure whether answers contain the prior value, on the 29 questions with an identifiable short prior value.

| | Share of answers that hold | | | |
|---|---|---|---|---|
| Which record the read is given | the current value only | the prior value only | both | neither |
| *Records in the state* | | | | |
| The record holding the answer | .149 | .126 | .609 | .115 |
| That same record, retracted | .000 | .000 | .000 | 1.000 |
| A different occupied record | .000 | .006 | .000 | .994 |
| The model's own choice | .011 | .017 | .023 | .948 |
| No state | .000 | .000 | .000 | 1.000 |
| *Same records as prompt text* | | | | |
| Records in the prompt, last 30k tokens | .207 | .138 | .000 | .655 |
| with the answer line identified | .172 | .241 | .241 | .345 |

Table 5: HaluMem, six seeds, on the 29 of 82 questions whose replacement line names a short prior value. HaluMem writes a replacement as "changed from A to B", so an answer that copies the line holds both values. Prompt rows are one greedy pass over the frozen backbone. Each batch exceeds the model's 30k-token window, so the prompt rows see the last 30k tokens of the records.

Table 5 reports the outcome. When the model selects the record itself, its answer contains neither the current nor the prior value in .948 of the 29 questions. When a person names the record holding the answer, that falls to .115. The current value is then in the answer in .758 of them. A record can hold both values, though, and the decoder copies the line that holds them. Of those answers, .609 carry the prior value alongside the current one. Another .126 carry the prior value alone. Naming the right record does not by itself keep outdated information out of the answer.

**Comparison with records supplied as prompt text.** A person who retracts a record has said not to use it. Does reading the editable state honour that, and does putting the same records in the prompt? We compare answers from the state with five prompt conditions (Table 6). The first gives all records. The second gives all records with the answer line identified. The third gives only the current and superseded lines. The fourth gives only the current line. The fifth gives the four records a frozen BGE retriever ranks highest for the question. The retriever condition puts four records in the prompt, the same number the record-store experiments of Section 4.1 select. These conditions distinguish difficulty locating the relevant information from difficulty choosing between current and outdated values. Every prompt tells the model that later lines replace earlier ones. The state instead stores each record's position, so the order needs no instruction.

No prompt condition keeps the superseded value out of the answer. Retrieval leaves it there most often, and it survives even when the prompt holds the answer line alone. Reading from the state leaves it at .000 on FactConsolidation and .008 on held-out MQuAKE-CF. Retrieval is not the reason: it puts the record holding the answer in the prompt on every question of both corpora, 66 of 66 and 144 of 144. A prompt can carry the right record and still not say which record a person has withdrawn.

Identifying the answer line improves the prompt model's current-value-only score on both corpora. Providing that line alone improves it further. Yet even with only that line, some answers contain the outdated value. The comparison separates finding the current record from excluding outdated content in the answer.

**Generating a newly written value.** Retraction tests whether a record is needed for an answer. Rewriting tests whether changing its content changes the answer accordingly. On FactConsolidation, we replace only the value in the stored line, preserving its subject and relation. Ten fixed replacement words are used, none of which appears in the old or the current value. Across five seeds the old

| Where the records are | Share of answers that hold | | | |
|---|---|---|---|---|
| | the current value only | the old one only | both | neither |
| *FactConsolidation, 66 questions* | | | | |
| All records in the prompt | .076 | .879 | .030 | .015 |
| answer line named | .561 | .288 | .121 | .030 |
| answer and superseded line only | .424 | .470 | .091 | .015 |
| answer line only | .818 | .015 | .076 | .091 |
| Retrieved records in the prompt, top four | .667 | .242 | .015 | .076 |
| Records in the state, record named | .917 | .000 | .000 | .083 |
| *Held-out MQuAKE-CF, 144 questions* | | | | |
| All records in the prompt | .076 | .875 | .035 | .014 |
| answer line named | .229 | .479 | .271 | .021 |
| answer and superseded line only | .556 | .389 | .049 | .007 |
| answer line only | .840 | .021 | .056 | .083 |
| Retrieved records in the prompt, top four | .604 | .326 | .035 | .035 |
| Records in the state, record named | .876 | .000 | .008 | .116 |

Table 6: Which values the generated answer holds. Prompt rows are one greedy pass over the frozen backbone. State rows average six training seeds. The retrieval row puts the four records a frozen BGE encoder ranks highest in the prompt.

answer remains in $0/330$ cases. The replacement is emitted exactly in $95/330$. Rewriting suppresses the original answer more reliably than it produces the exact replacement.

**Restoration and persistence across reload.** We test whether the operations remain consistent when applied sequentially to the same state. We write a replacement, retract the edited fact, restore its use, and save and reload the state, asking the same question after each step. A write counts as successful when the answer after the write is the written value. Across three seeds per corpus, retraction removes the written value in $54/54$ successful writes on FactConsolidation and $143/144$ on MQuAKE-CF. Restoration recovers every written answer removed by retraction. Reloading leaves all $198$ and $432$ answers unchanged, respectively.

## 5 Conclusion

Transfiver proposes one shared editable state for human–AI collaboration. The person and the model revise the same information, and the model uses that information in subsequent computation. All persistent information from their interaction that can affect future behavior must be included in this state.

The learned model makes stored facts and their availability directly editable within its answer computation. Preventing a fact's use changes answers that depend on it. Writing a replacement is less reliable. The conversation experiments show why it matters where an edit is made. Omitting a record from one input leaves it available for later retrieval. Changing its stored status persists. Deletion also prevents reuse, while retraction retains the record for inspection and restoration.

The next step is to learn when incoming information should revise an existing record. The evaluated learned model primarily stores incoming records separately. Recognizing and updating the appropriate existing record would extend the same shared-state mechanism to ongoing interaction.

## 6 Limitations

The experiments establish selected properties of the architecture. Benchmark annotations stand in for the person, supplying the record address and the correction. No experiment follows one person and one model over time. The learned model makes errors in selecting records and in generating replacements. Retraction acts on a whole record, so an outdated value inside a record that stays in use remains available to the answer. Save–reload agreement covers the tested continuations. Deployment further requires an interface, access control, and a retention policy.

## ETHICS STATEMENT

We use public datasets and collected no new private conversation data. Incorrect or malicious edits to the shared state can affect subsequent answers. The record store logs editing operations. Deployment also requires authentication and access controls for state changes.

## REPRODUCIBILITY STATEMENT

The supplementary archive contains the two implementations, per-question results for every experiment, the scoring scripts, a trained checkpoint, and instructions for retraining and for reproducing each reported count. Scoring rules are given with each experiment.

## AI USE STATEMENT

AI tools helped write and edit the manuscript, find related work, refine experimental designs, implement methods, check experiments, and interpret results. Language models also served as experimental readers and answer judges, as described in the experimental methods.

## REFERENCES

David Alvarez-Melis and Tommi S. Jaakkola. Towards robust interpretability with self-explaining neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 7786–7795, Red Hook, NY, USA, 2018. Curran Associates Inc.

John R. Anderson, Daniel Bothell, Michael D. Byrne, Scott Douglass, Christian Lebiere, and Yulin Qin. An integrated theory of the mind. *Psychological Review*, 111(4):1036–1060, October 2004. doi: 10.1037/0033-295X.111.4.1036.

Anonymous. Halumem: Evaluating hallucinations in memory systems of agents. In *Submitted to ACL Rolling Review - January 2026*, 2026. URL https://openreview.net/forum?id=PDzE0elDvE. under review.

Jimmy Ba, Geoffrey Hinton, Volodymyr Mnih, Joel Z. Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pp. 4338–4346, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. $\tau^2$-bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=8GjSf9Rh7Z.

Vincent-Pierre Berges, Barlas Oguz, Daniel HAZIZA, Wen tau Yih, Luke Zettlemoyer, and Gargi Ghosh. Memory layers at scale. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=ATqGm1WyDj.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego De Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 2206–2240. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/borgeaud22a.html.

Payel Das, Subhajit Chaudhury, Elliot Nelson, Igor Melnyk, Sarathkrishna Swaminathan, Sihui Dai, Aurélie Lozano, Georgios Kollias, Vijil Chenthamarakshan, Jiří Navrátil, Soham Dan, and Pin-Yu Chen. Larimar: large language models with episodic memory control. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Atticus Geiger, Duligur Ibeling, Amir Zur, Maheep Chaudhary, Sonakshi Chauhan, Jing Huang, Aryaman Arora, Zhengxuan Wu, Noah Goodman, Christopher Potts, and Thomas Icard. Causal abstraction: A theoretical foundation for mechanistic interpretability, 2025. URL https://arxiv.org/abs/2301.04709.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines, 2014. URL https://arxiv.org/abs/1410.5401.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=tEYskw1VY2.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. Incorporating copying mechanism in sequence-to-sequence learning. In Katrin Erk and Noah A. Smith (eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1631–1640, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1154. URL https://aclanthology.org/P16-1154/.

Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in LLM agents via incremental multi-turn interactions. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=DT7JyQC3MR.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HklBjCEKvH.

Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5338–5348. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/koh20a.html.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stanczak, Peter Shaw, Christopher Pal, and Siva Reddy. Agentrewardbench: Evaluating automatic evaluations of web agent trajectories. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=fQcUZMPIvu.

Kevin Meng, David Bau, Alex J Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=-h6WAS6eE4.

Kevin Meng, Arnab Sen Sharma, Alex J Andonian, Yonatan Belinkov, and David Bau. Mass-editing memory in a transformer. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=MkbcAHIYgyS.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards llms as operating systems, 2024. URL https://arxiv.org/abs/2310.08560.

Vedant Patel. Supersede: Diagnosing and training the memory-update gap in LLM agents, 2026. URL https://arxiv.org/abs/2606.27472.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pp. 1842–1850. JMLR.org, 2016.

Matthias Scheutz, Scott A. DeLoach, and Julie A. Adams. A framework for developing and using shared mental models in human-agent teams. *Journal of Cognitive Engineering and Decision Making*, 11(3):203–224, 2017.

Abigail See, Peter Liu, and Christopher Manning. Get to the point: Summarization with pointer-generator networks. In *Association for Computational Linguistics*, 2017. URL https://arxiv.org/abs/1704.04368.

David Steinmann, Wolfgang Stammer, Felix Friedrich, and Kristian Kersting. Learning to intervene on concept bottlenecks, 2024. URL https://openreview.net/forum?id=oNkYPgnfHt.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (learn at test time): RNNs with expressive hidden states, 2025. URL https://openreview.net/forum?id=eifW0W0xgt.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory, 2025. URL https://arxiv.org/abs/2410.10813.

Zexuan Zhong, Zhengxuan Wu, Christopher D Manning, Christopher Potts, and Danqi Chen. MQuAKE: Assessing knowledge editing in language models via multi-hop questions. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=0hTPJBnncc.

# A LongMemEval experiments

## A.1 Per-reader counts for the 44-item lifecycle experiment

Table 7 gives each reader's counts for the conditions of Table 3, and Table 8 adds deletion from the store.

| Condition | Mistral-7B | Qwen2.5-7B | Phi-3-mini | Yi-1.5-9B |
|---|---|---|---|---|
| *Correct current-value answers, immediately after intervention (out of 44)* | | | | |
| No action | 19 | 23 | 20 | 22 |
| Prompt instruction to ignore record | 20 | 22 | 18 | 22 |
| Input changed for this question | 29 | 31 | 29 | 27 |
| State retraction | 29 | 31 | 29 | 27 |
| *Correct current-value answers, next question (out of 44)* | | | | |
| Input changed for this question | 19 | 23 | 20 | 22 |
| State retraction | 29 | 31 | 29 | 27 |
| *Model correctly identifies the earlier evidence record (out of 44)* | | | | |
| Input changed for this question | 0 | 0 | 0 | 0 |
| State retraction | 17 | 14 | 17 | 2 |
| *Store checks: cases out of 44, identical for all four readers* | | | | |
| Old record selected again: input-side correction | | 31 | | |
| Old record selected again: state retraction | | 0 | | |
| Old record selected after restoring its use | | 31 | | |
| All saved fields match after reload | | 44 | | |

Table 7: Input changed for this question and state retraction on 44 questions. The first three blocks count successful answers. The last block checks record selection and saved fields rather than generated answers. Historical questions ask the reader to identify the record containing the earlier value.

| Condition | Mistral-7B | Qwen2.5-7B | Phi-3-mini |
|---|---|---|---|
| *Old record selected again on the next question* | | | |
| Input changed for this question | 31 | 31 | 31 |
| Store deletion | 0 | 0 | 0 |
| State retraction | 0 | 0 | 0 |
| *Correct current-value answers on the next question* | | | |
| Input changed for this question | 19 | 23 | 20 |
| Store deletion | 29 | 31 | 29 |
| State retraction | 29 | 31 | 29 |
| *Earlier record retrieved for historical questions* | | | |
| Input changed for this question | 0 | 0 | 0 |
| Store deletion | 0 | 0 | 0 |
| State retraction | 31 | 31 | 31 |

Table 8: Input changed for this question, store deletion and state retraction on 44 questions, evaluated with three readers. Historical retrieval counts whether the earlier record is selected, not whether the model answers correctly from it.

### A.2 Controls and paired results for the 44-item lifecycle experiment

These are details of Section 4.1. The 44 eligible items are a subset of the 78 questions in the comparison panel below.

**Question selection and ranker training.** Of the 78 knowledge-update questions, eight lack an annotated earlier turn and 26 lack the current answer text in the annotated turn. The remaining 44 questions contain both values in a form the operation can identify. A ranker trained on frozen BGE embeddings selects four records. It is fitted in four folds, so no question is evaluated by a ranker trained on that question. The same ranker serves every intervention condition.

**Controls and scoring.** Withholding both evidence turns leaves at most 1 of 44 answers correct. Selecting the four lines most similar to the question yields 14–24 correct answers out of 44; selecting the four most recent yields 10–11. Neither selection rule uses learned parameters. An answer counts correct when it contains the reference string or a variant the benchmark lists. Under exact equality the same generations score 7, 8 and 12 of 44 with no action, against 8, 11 and 16 after retraction.

The direction holds in each of the three readers checked and the size does not, because these readers wrap the value in a sentence.

**Retraction versus instruction.** Table 9 compares retraction and a prompt instruction with no action on the same questions.

| Reader | Retraction wins / losses | Retraction $p$ | Instruction $p$ |
|---|---|---|---|
| Mistral | 10 / 0 | .002 | 1 |
| Qwen | 8 / 0 | .008 | 1 |
| Phi | 9 / 0 | .004 | .5 |
| Yi | 9 / 4 | .27 | 1 |

Table 9: Retraction and prompt instructions compared with no action on 44 questions. Wins count answers made correct by retraction. Losses count answers made incorrect. Both $p$ columns use exact McNemar tests.

Retraction gains correct answers without losses for three readers; Yi has both gains and losses.

**Writing and repeated delivery.** We deliver the same supplied correction in three ways: write it into the record, show it only for the current question, or show it again on every question. All three explicitly include the corrected record and render identical input on the intervention turn in $44/44$ cases. Table 10 compares their next-question outcomes.

| Next-question outcome | Mistral | Qwen | Phi |
|---|---|---|---|
| Correct: one-time correction | 19/44 | 23/44 | 20/44 |
| Correct: written correction | 41/44 | 39/44 | 35/44 |
| Correct: repeated correction | 41/44 | 39/44 | 35/44 |
| Written vs. one-time, paired $p$ | $4.8 \times 10^{-7}$ | $3.1 \times 10^{-5}$ | $2.8 \times 10^{-4}$ |

Table 10: Answers to the next question after a one-time, stored or repeatedly supplied correction. Counts are reference-string matches out of 44 questions. The $p$ values compare stored and one-time corrections using exact McNemar tests.

On the next question, the one-time correction disappears in every case; the other two remain (Table 10). Repeated delivery and a stored edit produce identical correctness outcomes on this test.

### A.3 Baseline comparisons on the 78-item panel

Table 11 compares conditions on the 78 LongMemEval knowledge-update questions used by Supersede (Patel, 2026). The upper block receives no supplied correction. The lower block receives the annotated record address and replacement value.

All conditions use Qwen2.5-3B-Instruct (revision `aa8e725`), greedy decoding and a 64-token answer limit. The released Supersede matcher (commit `677993d`) accepts a gold variant or token overlap of at least .6.

| Condition | Correct out of 78 | Accuracy |
|---|---|---|
| *Autonomous: no supplied correction* | | |
| Bounded self-managed memory (Supersede, reported) | 7 | .090 |
| Bounded self-managed memory (our rerun) | 8 | .103 |
| Trained bounded memory (Supersede, reported) | 13 | .167 |
| Unedited state, same four-record read | 36 | .462 |
| Transfiver, stream update alone | 39 | .500 |
| Full context | 50 | .641 |
| *Annotated: the correction is supplied* | | |
| Transfiver, supplied edit alone | 67 | .859 |
| Transfiver, both routes | 68 | .872 |

Table 11: LongMemEval knowledge-update results on 78 questions with Qwen2.5-3B and the released Supersede matcher. Only the lower block receives an annotated correction.

**Supplied corrections and input size.** Corrections are supplied on 76 questions. The other two have only one annotated evidence turn. The edit uses the benchmark's first annotated turn as its address and the benchmark's labelled current value as its content. That labelled value is the answer string in 51 of the 76, and an equivalent short form such as "25:50" in the rest. Five questions require both current and earlier information, so historical queries also receive the edit history.

The read always includes the edited record. Inputs contain four records in 64 cases and five or six in the remaining 14, where the correction is added to a full read. The edit condition answers $56/64$ correctly on the four-record subset. Removing the inclusion rule omits the edited record in $25/78$ cases and reduces accuracy to $57/78$. The state view contains a median of 821 characters, compared with the baseline note's 300-character limit. Table 12 compares conditions with matched input budgets and supplied corrections.

**Effect of editing.** Compared with the unedited state, supplied edits gain 32 correct answers and lose one (exact McNemar $p = 7.9 \times 10^{-9}$). Adding stream updates to the edits gains two and loses one. Both conditions answer $66/76$ corrected questions correctly, with the net gain occurring on the two questions without supplied corrections.

Snapshot undo restores every edited state's fingerprint. Loading saved states in a fresh process reproduces selection, view and answer in $78/78$ cases.

**Frontier reader and scoring.** We also evaluate `gpt-5.5` on the same prompts (execution settings in Appendix A.10). The LLM judge credits $74/78$ answers for full context, $57/78$ for bounded memory, $47/78$ for stream updates, $71/78$ for supplied edits and $75/78$ for both routes. Both routes gain 19 answers and lose one against bounded memory (exact McNemar $p = 4.0 \times 10^{-5}$), and gain three and lose two against full context.

The programmatic matcher scores these conditions $69/78$, $44/78$, $49/78$, $67/78$ and $69/78$, respectively. It rejects some number spellings and changes in grammatical person that the judge accepts. The ordering of stream updates and bounded memory therefore reverses between graders. On 56 questions with two stream-update runs, 33 generated answers differ. These repeats measure answer variability rather than establishing a stable ranking between conditions.

### A.4 Input-budget and correction-matched controls

The edit condition of Table 11 shows a larger view than the note it is compared against, so its advantage could be budget rather than structure. Table 12 separates the two by capping the state view at the note's own budget, and adds a retriever and a full-context condition on three further readers. Every condition uses the same 78-question panel and the released Supersede matcher. The three readers are Qwen2.5-7B-Instruct, Mistral-7B-Instruct and Llama-3.1-8B-Instruct. The retriever uses a fixed BGE encoder to rank raw conversation turns by cosine similarity to the question. It presents the top four turns in chronological order.

| Condition | Qwen2.5-7B | Mistral-7B | Llama-3.1-8B |
|---|---|---|---|
| Full context | .808 | .679 | .718 |
| Self-managed notes, 300 chars | .359 | .256 | .205 |
| Embedding retrieval, top 4 | .397 | .538 | .474 |
| Supplied edit, top 4 | .756 | .769 | .744 |
| Supplied edit, view capped to 300 chars | .590 | .577 | .590 |
| Retriever with the same supplied correction, top 4 | .756 | .859 | — |

Table 12: Accuracy on 78 questions under three readers. The capped state view and self-managed notes have a 300-character limit. Supplied edits and correction-matched retrieval receive the same annotated replacement values. Other conditions do not.

Capping the edited view reduces accuracy for every reader, but it remains above self-managed notes at the same character limit. The edited view also receives a supplied correction. With that correction given to both systems, the retriever matches the edited store on Qwen and exceeds it on Mistral.

A retriever can also be told what a person retracted. We add a condition that deletes the annotated superseded turns from the store before retrieval, so it receives the same addresses the directed edit acts on. It answers 33, 40 and 35 of 78 on Qwen2.5-7B, Mistral-7B and Phi-3-mini, against 31, 42 and 34 for the same retriever with nothing deleted. Paired, the deletion wins four and loses two, wins three and loses five, and wins five and loses four. Deleting the stale turn moves the retriever by at most two items on the turn itself.

The related 44-item lifecycle experiment (Table 8) tests a different outcome: store deletion persists across turns just as retraction does, but does not retain the earlier record for historical access.

**Matching the correction.** For the final row of Table 12, the retriever receives the same corrected line at the same annotated address as the directed-edit condition. That line occupies one of four records; frozen BGE retrieval supplies the other three. The two shared-reader columns are the comparison in Table 13.

The capped edited view uses 300 characters on average. Notes use 259 characters on Qwen and fill the budget on the other two readers.

**Numerical precision.** The reported results use float16. A bfloat16 check reproduces the tested Mistral scores and changes the tested Llama scores by up to three questions.

### A.5 A RETRIEVER GIVEN THE SAME CORRECTION

We give an ordinary retriever the same correction as the edited store on the broader 78-question LongMemEval panel (Table 13). Both systems place the corrected line at the annotated record's address and show four records to the same reader. Scores use the released matcher, which accepts a gold variant or token overlap of at least .6.

| Same supplied correction, four-record view | Qwen2.5-7B | Mistral-7B |
|---|---|---|
| Edited record store | 59/78 | 60/78 |
| Retriever with the same correction | 59/78 | 67/78 |

Table 13: Correct answers on 78 questions when both systems receive the same corrected record. Each column uses the same reader and the released Supersede matcher.

With the same correction supplied, the retriever matches the edited store on Qwen and answers more questions correctly on Mistral. Each question is asked once, on the turn where the correction is supplied.

### A.6 EFFECT OF INCLUDING EDITED RECORDS

The 78-item edit condition and the 44-item write condition explicitly reserve space for the edited record in the model's input. We test how much answer accuracy depends on this inclusion rule by

varying the number of records provided, with and without the rule. The experiment uses the 44-item store and Mistral-7B, holding all other settings fixed. Table 14 reports immediate answer accuracy after the written correction of Section 4.1.

| Records the read may show | 1 | 2 | 4 | 8 |
|---|---|---|---|---|
| Edited record in view without the rule | $14/44$ | $25/44$ | $34/44$ | $41/44$ |
| Correct with the rule | 39 | 40 | 41 | 40 |
| Correct without it | 27 | 33 | 37 | 39 |
| Items the rule wins, loses | $13, 1$ | $7, 0$ | $4, 0$ | $2, 1$ |
| Exact McNemar $p$ | .002 | .016 | .125 | 1 |

Table 14: Effect of always including the edited record, using Mistral-7B on 44 questions. Columns give the number of records shown. Correct answers contain the reference string. Wins and losses compare correctness with and without the rule on the same questions.

Including the edited record matters most when the reader sees few records. With a larger input, retrieval more often includes it without the rule.

### A.7 Conversation summarization

We test whether a repeatedly updated summary preserves the information needed to answer knowledge-update questions. On the same 44 items, the model reads one conversation session at a time and rewrites a note of at most 300 characters. Subsequent sessions use this note without re-reading earlier sessions. We compare answers from the final note with answers from the full conversation. Final notes average 300 and 268 characters for the two readers.

| Measure | Mistral | Qwen |
|---|---|---|
| Correct from full history | 27/44 | 22/44 |
| Correct from rewritten note | 4/44 | 4/44 |
| Note wins / losses vs. full history | 0 / 23 | 1 / 19 |
| Paired $p$ | $< 10^{-6}$ | $4 \times 10^{-5}$ |
| Correct after retraction under note | 3/44 | 7/44 |
| Paired $p$ vs. note alone | 1 | .25 |

Table 15: Full-history and rewritten-note answers on 44 questions, scored by reference-string containment. Columns use Mistral-7B and Qwen2.5-7B. Wins and losses compare the note with full history. The $p$ values use exact McNemar tests.

The note loses most correct answers, and retracting the superseded record does not reliably recover them.

### A.8 Errors with full conversation context

The full-context condition answers 50 of 78 under the same reader, answer limit and matcher. Of the 28 it misses, 5 carry the reference value in the text and are refused on surface form, a digit against a spelled number. The other 23 assert a different value, and several of those are the value the conversation later replaced: “three bikes” against a reference of 4, “three tops” against five, “every two weeks” against every week. Seeing every turn does not settle which turn is current.

### A.9 Reader-repeat stability diagnostic

We repeated the no-action and state-retraction conditions in three fresh processes for each of the four readers, using the same 44 questions and decoding settings. Every reader produced identical normalized answers across the three repeats in both conditions. Only one repeat’s per-question outputs are released alongside the aggregate.

### A.10 FRONTIER READER EXECUTION SETTINGS

The frontier reader uses a 2048-token completion budget, following the released evaluation script for reasoning models. This budget includes reasoning and answer tokens.

A hosted model has no revision to pin, so we record the run instead. Every call in the frontier evaluation of Appendix A.3 went to `gpt-5.5` as served by the gateway on 2026-09-08 UTC, at temperature 0. This covers all five conditions and the judge. The judge prompt is taken from the released script. The nominal four-token judge setting is not an effective cap for a reasoning model. The supplied evaluation wrapper raises reasoning-model budgets to at least 2048 tokens, and the recorded judge usage includes reasoning tokens.

## B LEARNED SHARED-STATE MODEL

### B.1 LEARNED-STATE IMPLEMENTATION

Two implementations appear in this paper under the shared-state requirements. The record store of Section 4.1 provides the operation log used to distinguish restoring records from restoring the full earlier state. The learned model is the one used in Section 4.2. Its visible state and its computed-from state are the same tensors, with no prompt between them. The evaluated model generates answers with the copy decoder described below. An availability flag controls whether each record may be used, allowing retraction without deleting its content.

A learned router allocates writes between existing and empty slots. The evaluated token decoder combines copying with the frozen backbone. Addressing and readout equations appear in Appendix B.2.

Table 16 defines the implementation terms. The record store delivers selected records as prompt text. The learned model instead reads stored tokens within its answer computation.

| Term | Meaning |
|---|---|
| Record / content | A stored unit of information / the information it contains. |
| Slot | A storage location holding a record and the vectors used to find and read it. A record may contain a factual line or a conversational turn. |
| Key $K$ / value $V$ | Learned vectors used to match a query and represent stored information. The value vector is distinct from the answer text. |
| Availability flag $\Sigma$ | Controls whether the record may be used for current answers. |
| Token buffer $T$ | The stored text from which answer tokens can be copied. |
| Read weight | A slot's weight in the mixture used for copying. |
| Copy distribution | Token probabilities obtained by combining contributions from stored text. |
| Copy gate | The learned proportion mixing the copy distribution with the backbone's token distribution. |
| Placement gate | Allocates write mass between existing and empty slots. |
| Anchor $A$ | A learned vector associated with an empty slot that helps determine where to write an incoming record. |

Table 16: Terms used to describe the learned implementation.

**Training and editable fields.** The learned state serializes fifteen fields: $K, V, O, \Sigma, T, M, A$, a clock, per-slot and per-edge timestamps, a relation matrix and its event counts, and provenance, lifecycle and signature vectors. Table 16 defines the record and readout fields. Reload restores all fifteen fields, including those not directly edited in the intervention tests. There is no persistent recurrent hidden state. The current event is encoded into a step-local vector $h_t$, and a shared key encoder gives query $q_t$.

Training optimizes the state and read/copy modules with answer loss, a write-preservation auxiliary, target-slot selection supervision and a contrastive key loss. The curriculum increases distractors from 8 to 256. Reported checkpoints use $1{,}024$ training steps and a bias toward empty slots.

The occupancy indicator $O$ marks occupied slots, and $M$ marks valid token positions. Retraction sets $\Sigma_i = 0$. Rewriting replaces $T_i, M_i$ and recomputes token-derived copy keys while leaving $K, V$ unchanged.

### B.2 Slot addressing and readout

The learned core assigns a read weight to each slot from the current query and state. Let $\alpha_{t,i}$ denote this weight before applying the availability flag. Answer generation combines tokens copied from the buffer $T$ with the frozen backbone's token distribution. Write $M$ for the buffer mask and $\Sigma$ for the availability flag a person sets. A slot is readable only if it is occupied, available and non-empty, and the read weight is renormalised over exactly those slots,

$$\tilde{\alpha}_{t,i} \;\propto\; \alpha_{t,i}\, O_i \mathbb{1}[\Sigma_i > .5]\, \mathbb{1}[\exists j : M_{ij}], \qquad \sum_i \tilde{\alpha}_{t,i} = 1. \tag{3}$$

The read then scales these weights by $\Sigma$. Retracting a slot sets $\Sigma_i = 0$, which drops it from both steps and sends its read weight to exactly zero. This excludes the retracted slot from the copy distribution. Within a readable slot the decoder attends over buffer positions the walk has not already used, under a position bias $b_{t,ij}$,

$$\beta_{t,ij} = \mathrm{softmax}_j\left(\frac{\langle W_q h_t,\; W_k e(T_{ij})\rangle}{\sqrt{d}} + b_{t,ij}\right) \text{ on } M_{ij} \wedge \neg C_{t,ij}, \tag{4}$$

where $C_{t,ij}$ marks a position that already supplied an emitted token. The bias has two cases. Before slot $i$ has copied anything, the bias is $b_{t,ij} = -\lambda\,(1 - \epsilon_{ij})$, where $\epsilon_{ij} = 1$ when position $j$ holds a token the question $Q$ does not contain and position $j{-}1$ does not. The start is therefore pushed onto the first token of a stretch the question did not already say. Once slot $i$ has copied position $j_i$, $b_{t,ij} = -\mu\,|j - (j_i{+}1)|$ walks the stored line left to right. We use $\lambda = 8$ and $\mu = 2$ in every run. Neither rule adds a parameter, and training applies the same bias, so $\beta$ is trained under the rule it is decoded under. The copy distribution places the weight of every buffer position on the token stored there,

$$p_t^{\mathrm{copy}}(v) = \sum_{i,j} \tilde{\alpha}_{t,i}\, \beta_{t,ij}\, \mathbb{1}[T_{ij} = v]. \tag{5}$$

The copy branch follows pointer-generator and copying-mechanism decoders (Gu et al., 2016; See et al., 2017). A learned copy gate $g_t = \sigma(w^\top h_t)$ mixes it with the frozen backbone, $p_t = g_t\, p_t^{\mathrm{LM}} + (1 - g_t)\, p_t^{\mathrm{copy}}$, and the next token is $\arg\max_v p_t(v)$. Two rules let the answer end. Stop tokens are scored from the backbone at full strength, $p_t(s) \leftarrow \max\big(p_t(s), p_t^{\mathrm{LM}}(s)\big)$, with $g_t$ floored at .1. This prevents the copy gate from down-weighting the backbone's stop-token scores. Decoding also halts when the walk reaches the end of the line it is reading, that is when position $j_i{+}1$ is masked or already used. Across the six seeds the copy branch supplies the larger contribution for .987, 1.00, 1.00, .996, .652 and .996 of the emitted tokens. The generated-token distribution depends on the current question, fixed parameters and the readable state, including its token buffer. The intervention and reload tests check the reported behavior of this combined system.

**State updates in the evaluated implementation.** The evaluated write mechanism computes an address among occupied slots and another among empty slots, each with a hard forward choice and a straight-through gradient. A learned placement gate $p$ allocates write mass between them. Let $w$ be the binary forward write decision, $a_i^{\mathrm{occ}}$ and $a_i^{\mathrm{empty}}$ the two addresses, $k, v$ the incoming event's encoded key and value, and $\hat{v}$ the learned update candidate computed from the event and the addressed prior values. Then

$$u_i = w(1-p)a_i^{\mathrm{occ}}, \qquad n_i = wpa_i^{\mathrm{empty}}, \qquad d_i = u_i + n_i, \tag{6}$$

$$K_i' = (1 - d_i)K_i + d_i k, \tag{7}$$

$$V_i' = (1 - d_i)V_i + u_i\hat{v} + n_i v, \tag{8}$$

$$O_i' = O_i + (1 - O_i)n_i. \tag{9}$$

When the write fires, the token buffer and mask at $\arg\max_i d_i$ are replaced with the incoming record's tokens and valid positions. Directed text edits instead replace that buffer at the supplied address and recompute the copy key from the edited tokens, so the read is addressed by the new text;

the slot's learned $K, V$ are left as written. Section 4.2 measures what reaches the answer through this path.

The initial anchors are learned parameters. Their values are included in the saved state as $A$ and remain unchanged during deployment transitions. Slot relabeling must permute the anchors along with all other slot-indexed fields, and both axes of relation matrices. This is the symmetry required of slot relabeling.

### B.3 Learned-model results on FactConsolidation

We report six training seeds for the learned model of Section 4.2. The model combines a learned slot-selection and copy mechanism with a frozen Qwen2.5-7B-Instruct backbone. FactConsolidation provides $455$ records and $100$ questions; $66$ questions can be linked to stored old/current pairs and are evaluated here. Answers are generated greedily and scored by the containment rule of Section 4.2. Every seed stores all $455$ records without overwriting, with a write probability of .9997–.9999.

Even when the correct record is supplied, the model sometimes copies extra words or omits part of the answer. Errors therefore arise both in selecting a record and in copying its answer. Emptying the state leaves no correct answers in any seed. Per-seed diagnostic outputs are included in the supplementary files.

**Retracting the supporting record removes the answer.** We retract the record containing the current answer, then ask the same question again. Table 17 compares this operation with retracting other records. Across seeds, target retraction removes $203/209$ previously correct answers and gains none (exact McNemar $p = 1.5 \times 10^{-11}$–$3.7 \times 10^{-9}$ per seed). The four controls lose four previously correct answers in total; each control retracts one record on every question. No control has $p < .25$ in any seed.

| | Training seed | | | | | |
|---|---|---|---|---|---|---|
| Record retracted | 71 | 72 | 73 | 74 | 75 | 76 |
| None | 38 | 39 | 34 | 30 | 31 | 37 |
| Current-answer record | 1 | 2 | 1 | 1 | 0 | 1 |
| Superseded record | 38 | 40 | 35 | 30 | 32 | 39 |
| *Controls: neither current nor superseded record is retracted* | | | | | | |
| Unrelated record | 38 | 39 | 34 | 31 | 31 | 37 |
| Another highly weighted record | 37 | 41 | 34 | 31 | 32 | 37 |
| Closest weight to superseded record | 38 | 39 | 34 | 31 | 31 | 37 |
| Closest weight to current record | 37 | 41 | 35 | 32 | 31 | 39 |
| *Use of the superseded value* | | | | | | |
| Old-value answers before retraction | 0 | 5 | 6 | 1 | 8 | 3 |
| Old-value answers after target retraction | 11 | 12 | 11 | 8 | 20 | 12 |
| Superseded slot ranked first before edit | 0 | 2 | 4 | 1 | 7 | 3 |

Table 17: Retraction results on 66 questions per seed. The upper blocks count correct current-value answers. The final block reports use of the superseded value. Controls retract a different record and retain both the current and superseded records.

| | Training seed | | | | | |
|---|---|---|---|---|---|---|
| Record retracted | 71 | 72 | 73 | 74 | 75 | 76 |
| None | 77 | 83 | 90 | 87 | 71 | 86 |
| Current-answer record | 1 | 3 | 0 | 0 | 0 | 1 |
| Superseded record | 78 | 84 | 96 | 91 | 80 | 95 |
| *Controls: neither current nor superseded record is retracted* | | | | | | |
| Unrelated record | 76 | 83 | 90 | 88 | 70 | 86 |
| Another highly weighted record | 77 | 82 | 90 | 89 | 70 | 85 |
| Closest weight to superseded record | 77 | 82 | 90 | 87 | 72 | 86 |
| Closest weight to current record | 78 | 82 | 91 | 89 | 73 | 86 |

Table 18: The same retraction and controls on the 144 held-out MQuAKE-CF questions, counting answers that contain the current value. Retracting the record holding the answer removes 489 of the 494 answers that were correct beforehand. No control removes more than five.

A closest-weight control selects a different slot whose read weight is nearest to the specified record's weight. The closest-weight control for the current record loses one previously correct answer across the six seeds. The control carries less weight than the target, .055–.069 against .28–.46 on average across seeds, and carries more than the target in about a fifth of questions. On average the read concentrates weight on the record that holds the answer. The median effective number of slots receiving read weight is 2.2–5.4 across seeds.

Retracting the superseded record never lowers accuracy by more than one question per seed. On held-out MQuAKE-CF it raises accuracy on every seed, by up to nine questions. Removing the outdated record does not remove the current answer. Restoring the changed field to its saved value returns the state to its previous contents; the sequential restoration test is described next.

**Writing and sequential intervention details.** We replace only the value in the supporting record, keeping its subject and relation. Replacements come from ten fixed words, none of which appears in the old or the current value. Table 19 separates suppression of the original answer from successful generation of its replacement.

| Measure | 71 | 72 | 73 | 74 | 75 |
|---|---|---|---|---|---|
| Correct original answer before edit | 22 | 23 | 15 | 16 | 14 |
| Original answer after edit | 0 | 0 | 0 | 0 | 0 |
| Exact replacement after edit | 18 | 25 | 17 | 18 | 17 |
| Copy-branch share | 1.00 | 1.00 | 1.00 | 1.00 | .997 |

Table 19: Writing results on 66 questions per training seed. The first three rows count exact answers. Copy-branch share is the fraction of generation steps where copying contributes more than the backbone to the selected token's score.

Suppressing the original answer is more reliable than generating the exact replacement across all five seeds.

**What the rewrite logs show.** We re-examined the existing question-level outputs and token-source logs from these five seeds (Table 20). Most cases that fail after rewriting were already answered incorrectly before the edit. For each generated token, the log identifies the slot contributing the most copy probability to that token. The logs record the largest contributor; other slots may also contribute to each token. The counts and examples are computed from the released logs.

| Before rewriting | Cases | Exact replacement afterwards |
|---|---|---|
| Correct answer | 90 | 76 |
| Incorrect answer | 240 | 19 |
| Total | 330 | 95 |
| **Among the rewrites that did not come out exactly** | | **Cases** |
| Every copied token came from one slot | | 235 |
| The answer still contains the replacement | | 79 |

Table 20: Rewrite outputs across five seeds. The lower block covers the 235 rewrites that did not produce the replacement exactly, and counts the slots logged as the largest copy contributor at each generation step.

The outputs make the latter failures concrete. In seed 71, changing Charles Messier's stored nationality to "quicksand" produces "Charles Messier is a citizen of quicksand". The replacement is there, inside a longer phrase, so exact matching counts it as a failure. This happens in 79 of the 235 failures. Thus some failures arise even when the edited record supplies the output, without requiring a complex contextual interpretation.

The sequential write–retract–restore–reload results are reported in Table 4. Restoration recovers every written answer removed by retraction: $54/54$ on FactConsolidation and $143/143$ on MQuAKE-CF.

The selection and copying diagnostics are in Appendix B.5; the fresh-process reload audit is in Appendix C.1.

### B.4 Retraction during an incoming stream

We test whether a retraction remains effective when the model subsequently stores a new record. Unlike the tests after the full corpus has been stored, this experiment applies the edit before the current-value record arrives. We split the corpus in two. The first pass streams every released line except the 66 that carry a current value, so 389 lines occupy 389 slots. The line carrying the current value is then streamed, in three conditions on the same event: nothing touched, the slot holding the superseded line retracted first, and an unrelated slot retracted first.

Storing the incoming record neither re-enables nor overwrites the retracted slot. The superseded value appears in one answer in both the no-edit and unrelated-retraction conditions, and in no answer after retracting its own record. Across three seeds the incoming record is written to a different slot in every one of the 198 cases, with the mean probability assigned to a new slot ranging from .917 to .996. Thus retraction persists through the tested updates, but the model primarily stores the new information separately and selects between records at read time.

### B.5 Selection and copying diagnostics

**Locating the answer boundary.** Copying can continue beyond the answer when the value occurs inside a conversational turn. One configuration appends the benchmark answer as a delimited value field and starts the copy at the first token of a stretch absent from the question, with the model trained on that format. With the correct record supplied it answers $21/44$ exactly, against $2/44$ for the configuration used elsewhere. The value field is the benchmark's own labelled value, and the two configurations differ in both training and decoding. The same start rule lowers exact-match accuracy on FactConsolidation in every tested seed. With no record supplied this configuration answers $1/44$, so record selection is a separate source of error.

**Selecting conversational records.** A frozen BGE baseline ranks records by similarity to the question. Table 21 compares its first-ranked record with learned selection.

| Corpus | Records | BGE top-1 | Learned top-1 |
|---|---|---|---|
| FactConsolidation | 455 | 9/66 | (32–40)/66 |
| MQuAKE-CF | 288 | 23/144 | (80–91)/144 |
| Conversational turns | 1,032 | 9/44 | 7/44 |

Table 21: Questions whose correct record is ranked first. Ranges cover six FactConsolidation seeds and three MQuAKE-CF seeds. Conversational results use one evaluated configuration.

Learned selection performs better on short factual records but worse on conversational turns. On the conversational set, BGE places the correct record in its top four for $30/44$ questions; the learned read puts it first in $7/44$. On both factual sets, BGE's top four covers every evaluated question.

# C State restoration and formal properties

## C.1 Saving and reloading the learned state

To test whether the serialized learned state retains the information needed for subsequent answers, we compare evaluation immediately after streaming the records with evaluation after reloading the saved state in a fresh process. One process streams the record lines, writes the whole state to disk, and runs every condition. A second process starts from the file instead of the stream and runs the same conditions on the same questions. The two runs agree on every field of every record. The reload row of Table 4 scores the answers themselves: $66$ questions on FactConsolidation and $144$ on held-out MQuAKE-CF, each over three training seeds, and every answer is unchanged. The remaining $34$ FactConsolidation questions resolve to no stored pair and are scored in no condition. Logged slot selection, the generated string and the scored verdict all match. On these conditions the reloaded run is indistinguishable from the streamed one.

## C.2 Symmetry and rollback

Let $g_\theta(S, x)$ denote the answer under state $S$ and input $x$, and let $\mathrm{Edit}(S, i, q)$ replace record $i$ with content $q$. If slot numbers have no meaning, simply reordering the slots should not change the model's behavior. For a permutation $P$, the next-state distribution should permute in the same way while the output distribution remains unchanged:

$$T_\theta(PS, x) \overset{d}{=} PT_\theta(S, x), \qquad g_\theta(PS, x) \overset{d}{=} g_\theta(S, x), \tag{10}$$

where $\overset{d}{=}$ denotes equality in distribution. This matters for lifecycle operations as well: retracting or restoring a record should depend on its state and history. An arbitrary storage index must not decide it.

**Proposition 1** (Rollback restoration)**.** Assume fixed $\theta$ and history sufficiency. If an edit and its undo restore the serialized state exactly,

$$\mathrm{Undo}(\mathrm{Edit}(S, i, q)) = S,$$

then an identical future continuation induces the same distribution over future behavior as it did from $S$. Under matched randomness, a deterministic transition conditioned on that randomness reproduces the same trajectory.

Table 22 distinguishes the restoration operations.

| Operation | What is restored |
|---|---|
| Restore a record's use | Make a retracted record available for current answers again. The log retains the operations and the clock remains advanced. |
| Restore a snapshot | Reload the entire saved state, including its log and clock, to reproduce the state at the time of saving. |

Table 22: Restoring a record's use and restoring the full saved state.

For the event-logging store, the rollback proposition applies to loading an exact saved snapshot. Restoring a record's availability while retaining the operation log does not satisfy it.

# D Information model and state evolution

This section describes the broader information model and state-update principles behind Transfiver. It extends beyond the slot implementation evaluated here, including relationships between records and rules for interpreting incoming events.

## D.1 The information model and how the state changes

### D.1.1 One folder is not enough

In Transfiver, an item is not limited to a single folder or context. The same item can belong to several projects without being copied. Items that look similar are not necessarily related, while very different items may be strongly connected by a real event or relationship. For example, a bank password and a server password are lexically similar, but should not be merged, whereas a particular server and its random password may be lexically distant but joined by a specific event. Transfiver therefore represents both what an item means and how it is connected to other items.

### D.1.2 The canonical interaction boundary

The shared-state requirements apply at the boundary where interaction enters Transfiver. A turn may contain user-authored text, a model statement, a tool or world result, or text quoted or carried into a later user turn. The raw turn and its local context are retained in the event history. Transport is evidence about how text arrived. A user may quote an assistant, approve it, reject it, or issue a new instruction next to it. Interpretation and uncertainty therefore belong to the learned transition and its readable provenance, rather than to a hard-coded "user text is true" flag.

The serialization and intervention requirements are formalized in Section 3.

## D.2 How Transfiver learns and changes

### D.2.1 Two ways a state changes: directed edit and stream update

A directed edit supplies the record address. A stream update must infer whether an incoming event concerns an existing record or requires a new one. Successful directed edits therefore do not establish successful interpretation of the stream. When that identity is uncertain, the proposed transition should retain the event so later evidence can resolve the connection.

### D.2.2 When both routes reach the same item

In the proposed interaction protocol, a directed edit is an event with an author and time. If an edit and a stream update arrive together, the edit is applied last to the named field. This orders the two operations without treating all user-authored text as true. A later event may revise the field again. The reported experiments apply operations sequentially.

#### D.2.3 OFFLINE LEARNING AND ONLINE STATE EVOLUTION

Transfiver separates learning the dynamics from using the learned dynamics. During offline training, future task loss must reach the transition:

$$S_{t+1} = T_\theta(S_t, z_t), \qquad \theta \leftarrow \theta - \eta \nabla_\theta \sum_{t=1}^{H} \mathcal{L}(y_t, y_t^\star). \tag{11}$$

where $H$ is the training horizon and $z_t$ encodes the event arriving at step $t$; the complete step that produces $y_t$ appears in Equation 2. During deployment, $\theta$ is fixed and stream updates evolve $S$ through the forward transition:

$$\theta \text{ fixed}, \qquad S_{t+1} = T_\theta(S_t, z_t). \tag{12}$$

Directed edits are the other route by which $S$ changes: they modify $S$ directly at an address the user supplies, without changing $\theta$. Neither route touches the parameters. Thus "learning from one event" means persistent state change without gradient retraining.

#### D.2.4 WHAT MAY BE FIXED AND WHAT MAY NOT

Fixed dimensions and optimization settings specify computation; fixed source trust, importance, or semantic priority would instead substitute a designer's judgment for the user's. Delivery metadata may be routed deterministically, but interpreting its meaning belongs to the learned transition. User-dependent judgments require a stated signal, such as explicit marking or subsequent use; absence of use alone is not negative evidence.

New failures should first be tested using expressive learned combinations of content, context, source, time, use, importance and relation. A new semantic primitive is justified only when those distinctions fail suitable counterexamples.

## E SUPPORTING EXPERIMENTS

### E.1 WHERE RECORD SELECTION FAILS

This experiment asks what makes the choice of record hard.

We take released web-agent attempts and ask the model which step first produced an environment error. The answer is a step number, so answering at all requires picking the right record out of the stored steps. The environment's own error marker gives the reference. We registered this test before running it, with the stopping rule that a result indistinguishable from chance would end the attempt.

The read selects the marked step in $0$ of $66$ and $0$ of $72$ cases across two batches, and the marked step is not in its top five either. The same code selects the reference record in $80$ of $144$ MQuAKE-CF questions, so the selector is not broken; the difference is in the records. Web-agent steps read *In run X step N the agent did click(...)*, and each one is nearly a copy of its neighbours: the median similarity between a record's key and its nearest neighbour is .9996. There is almost nothing to address by. We report the result and make no claim about locating errors.

### E.2 STATUTE RECORD CONTROLS: CONSTRUCTION AND ADDITIONAL CHECKS

One requirement is that the answer is controlled by a particular stored record. Published statute amendments let us derive questions from replacement pairs without inventing the underlying facts or supporting lines. The record sits in the reader's prompt here, so this section measures control by a record on its own. Korean statutes publish their amendments as sentences of the form *in Article $n$, "A" is replaced by "B"*, so the replacement values and supporting line come from legislative text; questions are derived from those pairs. For one amendment the record is the pre-amendment article text followed by up to $49$ further amendment sentences of the same statute in force-date order, one of which licenses the answer. The reader is a frozen Llama-3.1-8B-Instruct, is told to answer only from the record, may answer "unknown", and must name the line it used.

The 1,101 questions contain 825 distinct answers; the most frequent accounts for .025 of them. To reduce surface-form cues, distractors are real amendment sentences of the same form, with preference for sentences quoting the same superseded expression. All figures use exact match after whitespace normalisation. A string-search baseline and record-removal controls below measure how much can be recovered from surface matching or prior knowledge.

Table 23 reports answer accuracy and record interventions. The first reader uses 1,101 questions from 144 statutes. The second uses a different draw of 1,101 questions, with 753 in common.

| Measure | Llama-3.1-8B | Qwen2.5-7B |
|---|---|---|
| Answers with amended value | .935 | .877 |
| Answers with superseded value | .006 | .007 |
| Names supporting line | .784 | .847 |
| Random line-selection accuracy | .021 | .020 |
| Answer changes: supporting line replaced | .712 | .761 |
| Answer unchanged: unrelated line replaced | .976 | .968 |

Table 23: Statute controls on 1,101 questions per reader. Values are proportions. The question sets overlap on 753 questions, so the reader comparison is not paired.

Replacing the supporting line changes answers much more often than replacing an unrelated line. A string-search rule finds the superseded expression and returns the replacement written beside it. On the first reader's questions, it answers correctly in .860 and identifies the supporting line in .649, below the reader on both measures. Its abstention rate is .116, versus .003 for the reader. The reader copies the entire supporting line verbatim in .001 of answers.

Korean statute text is published openly, so two further controls separate reading from recall. They form one separate run over 1,101 questions from the same corpus. It is the same draw the second reader uses, and 753 of its questions are shared with the run above. Each record is rebuilt from the published amendment history, and the run answers .929 with the record present. Removing the record and asking the same question answers 15, or .014: the reader does not know these amendments without being shown them. Replacing the line the reader names still gives the amended value in 229 of 1,099; the two items left out name no line. In 228 of those the value also appears on another line of the same record. Replacing every such line leaves 3 standing. What survives a single replacement is a second copy inside the record rather than recall.

### E.2.1 The record against what the reader knows

Two runs put the record in conflict with the reader's own knowledge. In both, a real amendment is written backwards: a published sentence of the same form replaces the current name with the superseded one, so following the record cannot be explained by the record agreeing with the world.

The first takes 53 ministry renamings. With no record the reader gives the new name in .396 and the old one in .189. With the real amendment it gives the new name in .943. With the amendment reversed it follows the record in .849 and returns to the name it knows in .019. Restricted to the 21 items it answers correctly with no record, it follows the record in .810 and returns to what it knows in 0 of 21.

The second repeats the construction on 200 items. With no record the reader knows the new value in .045 and gives the old one in .325; with the record it gives the new value in .885. With the amendment reversed it follows the record in .785 and its own knowledge returns in .015. The same run asks a question that presupposes the superseded value: the reader answers "no" and supplies the current value in .910, answers "no" without the current value in .055, and accepts the presupposition in 0 of 200.

## E.3 Naming the record to read

In deployment a person who corrects a record can also say which record the answer should come from. The read and the copy then use that record, and we ask the same held-out MQuAKE-CF questions. Table 24 reports six seeds.

| Share of answers containing | Record named | Not named | Difference |
|---|---|---|---|
| the stored value | .884 (.75–.99) | .572 (.49–.62) | $+.312$ |
| the superseded value | .008 (.007–.014) | .093 (.04–.13) | $-.085$ |

Table 24: Reference-value containment on 144 held-out MQuAKE-CF questions, averaged over six seeds. Parentheses give the minimum and maximum across seeds. Difference is the named-record score minus the unnamed-record score.

### E.4 Saying it was done, and doing it

We compare the assistant's completion reports with database outcomes in released $\tau^2$-bench transcripts (Barres et al., 2025). Restricting to the 2,160 episodes whose reference trajectory requires a database write, 684 end with a database that does not match the reference. Table 25 compares detecting failure from the closing message with detecting it from executed write calls. Of the 684 failed episodes, 352 end with a claim that the action was completed. Write-call comparisons detect more failures but require the reference trajectory, which is unavailable in deployment. This analysis tests what can be checked from the recorded actions.

| Failure detector | Failures detected | False-positive rate |
|---|---|---|
| Problem reported in closing message | 124/684 (.181) | .035 |
| Required write missing | 601/684 (.879) | .066 |
| Required write missing or extra write | 684/684 (1.000) | .147 |

Table 25: Detectors applied to the same released episodes. Each row reports how many of the 684 failed episodes that detector flags. False-positive rates are over 1,476 successful episodes. Write-call detectors compare executed calls with the reference trajectory.

## F Related work

The main text summarizes the closest comparisons. Here we discuss learned memory, editable representations and human–agent shared models in more detail. Transfiver combines human editing with the requirement that all persistent interaction-specific information used by the model reside in the edited state.

**Self-updating computational memory.** Neural Turing Machines and memory-augmented networks learn to address external memory (Graves et al., 2014; Santoro et al., 2016). Fast weights, test-time training, and Titans add state that changes on a faster time scale than ordinary parameters (Ba et al., 2016; Sun et al., 2025; Behrouz et al., 2025).

Transformers retain token-level context through attention (Vaswani et al., 2017); selective state-space models learn compact recurrent dynamics with strong efficiency and sequence performance (Gu & Dao, 2024).

**State-first agent memory.** Recent work measures the same failure we target from the model's side. Supersede shows that when an agent maintains a bounded self-managed memory instead of full context, accuracy on questions about changed facts drops from .92 to .77, and that the gap widens with conversation length rather than with compression ratio; it releases a reinforcement-learning environment that rewards answering from the current value and penalizes stale ones, and reports that fine-tuning nearly doubles held-out accuracy (Patel, 2026). That line of work makes the model a better custodian of its own memory. Transfiver takes the complementary position: the model may judge which stored item an incoming event concerns, but the authority to retract, restore and correct that item belongs to the user, and the item the user acts on is the item subsequent computation reads. Supersede does not expose the memory for human inspection or editing, and does not test intervention, rollback, or save–restore.

$k$NN language models, RAG, and RETRO combine a generator with explicit external records (Khandelwal et al., 2020; Lewis et al., 2020; Borgeaud et al., 2022); MemGPT manages memory tiers

around an LLM (Packer et al., 2024); and Larimar adds an editable episodic memory that supports one-shot knowledge updates without retraining the language model for each edit (Das et al., 2024). Memory Layers at Scale is closer in spirit, making a large addressable key–value store part of the forward computation rather than a retrieval sidecar (Berges et al., 2025).

ROME and MEMIT intervene on factual associations in Transformer parameters (Meng et al., 2022; 2023). Transfiver instead separates slowly learned shared dynamics $\theta$ from an online, user- or task-specific $S$. This avoids gradient editing during ordinary use, while imposing the stronger burden that state edits be causally localized and serializable.

**Shared mental models in human–agent teams.** Teaming research has long argued that effective human–agent collaboration depends on sufficiently aligned internal representations of the task, the equipment, the team members and their roles, and that these representations are what let one party anticipate the other. Scheutz, DeLoach and Adams (Scheutz et al., 2017) observe that most of this literature quantifies the benefit of such alignment while leaving open the data structures and processes that would operationalize it, and propose a formal framework to fill that gap. Transfiver can be read as one such operationalization at the level of a single persistent computational state. The distinction is between aligning separate representations and maintaining one shared editable object that the model uses for computation.

**States a person can see, and states a person can change.** Concept bottleneck models let people edit predicted concepts and propagate the change to outputs (Koh et al., 2020). Concept Bottleneck Memory Models retain past interventions and learn when to reapply them (Steinmann et al., 2024), making them a particularly close comparison for persistent human correction. Self-explaining networks build faithfulness into model design (Alvarez-Melis & Jaakkola, 2018). Transfiver shares the commitment to intervention and to faithfulness-by-design over post-hoc visualization. It differs by making the faithful, intervenable object *persistent and dynamically evolving* across interactions, and by requiring all persistent interaction-specific information that can affect future output to remain in that state. Causal-abstraction work supplies a stricter language for testing whether an interpreted high-level variable has the claimed causal role under interchange interventions (Geiger et al., 2025). That standard applies to Transfiver's requirements for viewing and editing the state.

ACT-R's declarative memory combines an item's own history with activation from currently active neighbors (Anderson et al., 2004).

**Evaluation.** LongMemEval identifies extraction, multi-session reasoning, temporal reasoning, knowledge update, and abstention as central long-term-memory abilities (Wu et al., 2025). Transfiver adds architectural audits: whether the state that appears correct is actually the state the system uses, and whether correction takes effect without parameter changes.