TASK-ORIENTED MEMORY COMPILATION: EXECUTABLE STATE REPRESENTATIONS FOR LONG-CONTEXT LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Answering questions over long interaction histories often requires tracking updated facts, following relations, and aggregating scattered evidence. Retrieval and compression methods supply selected text, which can leave these operations to the answering model, and typically rely on auxiliary neural models to produce it. We propose Task-Oriented Memory Compilation (TOMC), which instead executes these operations while building memory. TOMC computes state, relation, or count records from the history and combines them with selected source text under a fixed token budget. It is training-free, runs on CPU, and supplies plain text to black-box LLM APIs. On the BEAM long-term memory benchmark, with 100K–1M-token histories and three strong LLM API readers (GPT-5.1, Rednote preview, and DeepSeek 4.1 Flash), **TOMC achieves higher overall mean scores than six retrieval, memory, and compression baselines**. It also uses 32.4–36.6% fewer input tokens than the strongest baseline, LIGHT, and **11–17× shorter record construction time than per-call neural compressors**. At 500K and 1M tokens, TOMC scores higher in all 36 reader–baseline–length comparisons; its largest gains over BEAM-RAG and LIGHT are in temporal reasoning, which ablation links to the records. These results suggest that executing task operations during memory construction complements retrieval and offers an alternative to token compression.

1 INTRODUCTION

Long conversational histories make it difficult for language models to locate and use the information needed to answer a question (Tavakoli et al., 2026). For example, a user may change a project deadline in a later conversation while leaving earlier requirements unchanged. Answering a question about the current plan then requires combining the revised deadline with the still-valid requirements. Constructing compact memory from these histories requires balancing answer quality, model input length, and the computation required to build the memory (Jiang et al., 2024; Gu et al., 2026).

Existing approaches retrieve passages relevant to the current question, compress text, or combine several forms of memory. LongLLMLingua (Jiang et al., 2024) uses question-aware scoring and token pruning to produce compact inputs, while BRIEF-Pro (Gu et al., 2026) summarizes retrieved evidence. For long conversations, LIGHT (Tavakoli et al., 2026) combines retrieved conversation segments, recent turns, and a scratchpad that stores accumulated facts and is filtered for the current question. They provide selected or condensed text through a model-agnostic reader interface (Figure 1, left). Yet compact inputs can still require the answering model to resolve state updates or compute counts.

At the same time, the neural scoring, summarization, and scratchpad construction described above require model inference in addition to answer generation (Jiang et al., 2024; Gu et al., 2026; Tavakoli et al., 2026). **We therefore ask whether executing supported task operations during memory construction can improve answer quality without auxiliary neural-model inference.** We study this question with CPU-based construction and a model-agnostic text interface to the reader.

We introduce Task-Oriented Memory Compilation (TOMC), a training-free method that executes task operations during memory construction. TOMC uses predefined rules to select supporting evidence and compute task-specific records. These operations track current values from supported state updates, record links between entities, or count term occurrences within a specified scope. The

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.

compiled records give the answering model the results of these operations, but can omit wording and context needed for semantic questions. TOMC therefore combines the records with selected source text under a fixed token budget (Figure 1, right). Memory construction runs entirely on CPU without auxiliary neural-model inference. The model-agnostic text interface supports black-box API readers without access to their weights or architecture.

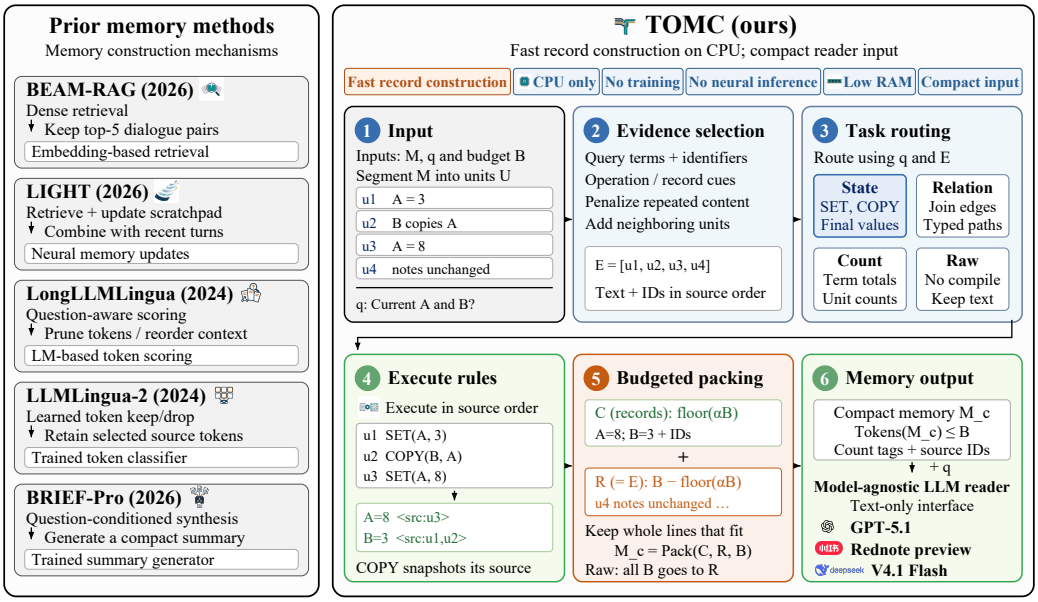

Figure 1: Memory construction mechanisms. TOMC executes supported task operations on CPU and packs records with source text under budget B. State is illustrated; other routes are alternatives. Fast record construction and CPU-only operation reduce construction overhead; measured time and memory are reported in Table 3. Readers are shared by TOMC and paired baselines.

Our contributions are threefold:

- **Executing task operations during memory construction.** Retrieval and compression methods pass selected or shortened text to the answering model, which may still need to resolve state updates, follow relations, or compute counts. TOMC instead executes these operations with predefined rules while building memory, and combines the resulting compiled records with selected source text under a fixed token budget.
- **Higher answer quality with fast, CPU-only record construction.** Prior compressors were mostly evaluated with earlier or smaller readers. With three strong LLM API readers (GPT-5.1, Rednote preview, and DeepSeek 4.1 Flash) on 600 BEAM questions, TOMC scores higher than six retrieval, memory, and compression baselines overall and in all 36 reader–baseline–length comparisons at 500K and 1M tokens. It needs no training, GPU, or auxiliary neural model: record construction takes 4.21 s per call on CPU, 11–17× shorter than for per-call neural compressors, and the memory uses 32.4–36.6% fewer input tokens than LIGHT.
- **Compiled records and selected source text both help.** Removing the compiled records lowers temporal-reasoning scores for all three readers, and removing the selected source text lowers F1 on key–value retrieval and multi-hop QA. On variable tracking, state records built from the full history reach 100% recall for every reader, versus 23.8–58.4% for query selection.

2 Related Work

Context compression aims to reduce model input length while retaining information needed for answering. LLMLingua-2 (Pan et al., 2024) learns which tokens to retain, while DAC (Zhao

et al., 2025) combines attention and entropy scores to select tokens. ICAE (Ge et al., 2024) and xRAG (Cheng et al., 2024) instead supply compressed context as learned latent representations rather than text. Reported reader evaluations include GPT-3.5-Turbo and Mistral-7B for LLMLingua-2 (Pan et al., 2024), Qwen2-7B and Llama-3.1-8B for DAC (Zhao et al., 2025), and Llama-3.1-8B/70B and GPT-4.1-nano for BRIEF-Pro (Gu et al., 2026). These results do not establish whether the same benefits hold for newer, high-capability readers.

Memory and context-management methods organize interaction histories for use in later queries and agent steps. LIGHT (Tavakoli et al., 2026) combines retrieved conversation segments, recent turns, and a query-filtered scratchpad. TiMem (Li et al., 2026) organizes conversations in a temporal memory tree and adapts memory recall to query complexity. ACE (Liao et al., 2026) retains original messages alongside their abstractions. Before each agent step, it selects which messages to include in full, include as abstractions, or omit from the model input.

Other approaches use explicit operations on structured data to support question answering. APEX-MEM (Banerjee et al., 2026) stores conversational facts in an append-only temporal graph. A language-model agent queries this graph to resolve conflicting information and perform temporal and relational computations. Chain-of-Table (Wang et al., 2024) uses a language model to select predefined table operations and their arguments at each reasoning step. TOMC instead uses predefined rules to parse evidence, select task operations, and construct textual memory on CPU without auxiliary neural-model inference.

3 METHOD

3.1 PROBLEM FORMULATION AND OVERVIEW

Given a context M, a question q, and a token budget B, TOMC constructs a compressed context M_c for question answering:

$$M_c = \mathrm{Compress}(M, q, B), \qquad \mathrm{Tokens}(M_c) \leq B. \tag{1}$$

Tokens is the configured memory-length counter. The budget applies only to M_c, excluding the question and system prompt. The reader interface is model-agnostic: $a = \mathrm{LLM}(M_c, q)$. Reference answers are not used to construct memory or generate answers.

TOMC first segments M and selects evidence relevant to q. It then executes predefined operations to construct state maps, relation paths, or count tables and combines these records with selected source text under budget B. The raw-text route skips compilation and constructs memory from selected source text alone. Figure 1 (right) and Algorithm 1 summarize this workflow.

3.2 SOURCE SEGMENTATION AND EVIDENCE SELECTION

TOMC divides M into an ordered sequence U of text segments, each with a unique source identifier. The reference implementation treats each nonempty line as one segment. Selected state updates are executed in source order, and source identifiers link compiled state and relation records to their supporting text. TOMC selects evidence E from U for question q using fixed rules. The reference implementation scores units using query-term overlap, query-identifier matches, and predefined cues for operations and structured records. It selects units iteratively, penalizes lexical overlap with previously selected units, and adds immediate neighbors for context. Selected units retain their original text and identifiers and are returned in source order. Appendix B.2 gives the scoring rules and settings.

3.3 TASK-SPECIFIC COMPILATION

TOMC uses predefined rules to select a route z from STATE, RELATION, COUNT, and RAW. The reference implementation selects z using predefined keywords in q. The first three routes produce compiled records C for current variable values, relations, or counts, respectively. The RAW route produces no compiled records and uses selected source text alone.

The following operations describe the reference compiler. Appendices A and B provide detailed algorithms and document differences from the experimental implementations.

Algorithm 1 Task-Oriented Memory Compilation

Require: original context M, question q, memory budget B
Require: fixed configuration Θ, including memory counter Tokens and lexical tokenizer ν
Ensure: compressed context $M_c = \mathrm{Compress}(M, q, B)$ with $\mathrm{Tokens}(M_c) \leq B$

$U \leftarrow \textsc{Segment}(M; \Theta)$ ▷ source IDs, order, and text
$E \leftarrow \textsc{Select}(U, q; \Theta)$ ▷ selected units in source order
$z \leftarrow \textsc{Route}(q, E; \Theta)$ ▷ choose one of the four routes below
if $z = \textsc{State}$ **then**
  $F \leftarrow \textsc{ParseFacts}(E, z; \Theta)$
  $C \leftarrow \textsc{CompileState}(F)$ ▷ ordered SET / snapshot COPY
else if $z = \textsc{Relation}$ **then**
  $F \leftarrow \textsc{ParseFacts}(E, z; \Theta)$
  $(\ell_1, \ldots, \ell_k) \leftarrow \textsc{RelationLabels}(q; \Theta)$ ▷ k labels; $k = 0$ if absent
  $C \leftarrow \textsc{CompilePath}(F, (\ell_1, \ldots, \ell_k))$ ▷ direct edges if $k = 0$
else if $z = \textsc{Count}$ **then**
  $U_{\mathrm{scope}} \leftarrow \textsc{CountScope}(U, E, q; \Theta)$
  $C \leftarrow \textsc{CompileCounts}(U_{\mathrm{scope}}, \nu)$ ▷ occurrence / source-unit counts
else ▷ $z = \textsc{Raw}$
  $C \leftarrow \varnothing$
end if
$R \leftarrow \textsc{RawBackup}(E, q, z; \Theta)$ ▷ raw candidates $R \subseteq E$
$M_c \leftarrow \textsc{Pack}(C, R, B; \Theta, \mathrm{Tokens})$ ▷ budget includes tags and source fields
return M_c
Model-agnostic reader interface: $a = \mathrm{LLM}(M_c, q)$.

For the STATE and RELATION routes, fixed rules parse supported statements in E into operation records F in source order. Executing F produces compiled records C. Each record carries an ordered source-ID list J and source order o. Here, v, w are variables, val is a value, h, t are the source and target entities, and ℓ is a relation label.

$$\begin{aligned} \texttt{SET}(v, \mathrm{val}, J, o) &\quad \text{assign value val to } v, \\ \texttt{COPY}(v, w, J, o) &\quad \text{assign the current value of } w \text{ to } v, \\ \texttt{EDGE}(h, \ell, t, J, o) &\quad \text{store relation } h \xrightarrow{\ell} t. \end{aligned} \tag{2}$$

The reference parser handles one supported statement per line; unmatched lines remain candidates for source-text retention.

The STATE route executes `SET` and `COPY` records in source order, starting from an empty state. Each operation changes only its target variable; copying an undefined variable yields the unknown value $\perp$. For example, the sequence `A=3; B copies A; A=8` yields `A=8, B=3`: changing `A` after the copy leaves `B` unchanged. The compiler emits final variable values with their supporting source IDs; Algorithm 2 specifies the source-ID updates. For questions asking only for the final values of a fixed set of variables, executing the updates leaves one value per variable, regardless of the number of updates (Appendix H).

The RELATION route joins `EDGE` records in F along a requested sequence of relation labels, matching the target of each edge to the source of the next. For example, `Alice -[member_of]-> TeamA` and `TeamA -[located_in]-> London` form a path from `Alice` to `London` along the label sequence `member_of > located_in`. Each path record stores its endpoints, label sequence, and supporting source IDs. The reference implementation reads the label sequence from a `path:` or `relation:` field in q; if neither is provided, it returns direct edges. Algorithm 3 gives the detailed procedure.

The COUNT route uses a lexical tokenizer ν to extract terms from source units in a specified scope U_{scope}. For each term, it emits a record containing its total occurrences and the number of source units containing it. For example, in `red red blue` and `blue`, both terms occur twice, but `red` appears in one unit and `blue` in two. The scope can include all eligible source units or only selected evidence E; the reference implementation uses E. Algorithm 4 gives the computation.

Table 1: BEAM answer quality across history lengths. Cells pair baseline / full TOMC scores (0–100); dashed rows pair TOMC w/o compilation / full TOMC from separate matched batches. Bold marks the higher paired score.

	GPT-5.1		Rednote preview		DeepSeek 4.1 Flash		Mean
Baseline	Base. / TOMC	Δ (%)	Base. / TOMC	Δ (%)	Base. / TOMC	Δ (%)	Δ (%)
Overall (all lengths)							
LLMLingua-2-D (2024)	37.69 / **60.16**	↑ 59.6	34.30 / **54.49**	↑ 58.9	33.13 / **57.29**	↑ 72.9	↑ 63.8
LongLLMLingua (2024)	35.43 / **55.03**	↑ 55.3	34.34 / **50.28**	↑ 46.4	39.01 / **57.18**	↑ 46.6	↑ 49.4
BRIEF-Pro (2026)	36.40 / **54.71**	↑ 50.3	35.30 / **51.03**	↑ 44.6	40.17 / **57.18**	↑ 42.4	↑ 45.7
LLMLingua-2-H (2024)	44.67 / **60.45**	↑ 35.3	39.47 / **56.16**	↑ 42.3	37.13 / **57.09**	↑ 53.8	↑ 43.8
BEAM-RAG (2026)	52.47 / **59.25**	↑ 12.9	51.63 / **55.90**	↑ 8.3	51.57 / **56.91**	↑ 10.4	↑ 10.5
LIGHT (2026)	53.75 / **59.48**	↑ 10.7	51.75 / **55.90**	↑ 8.0	55.65 / **56.91**	↑ 2.3	↑ 7.0
TOMC w/o compilation	55.00 / **56.20**	↑ 2.2	**51.25** / 50.34	↓ 1.8	49.74 / **52.74**	↑ 6.0	↑ 2.1
100K tokens							
LLMLingua-2-D (2024)	51.70 / **62.78**	↑ 21.4	45.03 / **56.24**	↑ 24.9	44.73 / **58.22**	↑ 30.1	↑ 25.5
LongLLMLingua (2024)	39.23 / **57.29**	↑ 46.0	35.36 / **50.53**	↑ 42.9	43.55 / **57.63**	↑ 32.3	↑ 40.4
BRIEF-Pro (2026)	36.67 / **55.90**	↑ 52.5	34.54 / **52.18**	↑ 51.1	44.43 / **57.63**	↑ 29.7	↑ 44.4
LLMLingua-2-H (2024)	53.37 / **63.58**	↑ 19.1	45.19 / **58.53**	↑ 29.5	45.05 / **56.65**	↑ 25.7	↑ 24.8
BEAM-RAG (2026)	53.58 / **62.26**	↑ 16.2	53.48 / **58.25**	↑ 8.9	51.57 / **58.21**	↑ 12.9	↑ 12.7
LIGHT (2026)	60.50 / **62.26**	↑ 2.9	57.05 / **58.25**	↑ 2.1	**61.01** / 58.21	↓ 4.6	↑ 0.1
TOMC w/o compilation	57.89 / **58.45**	↑ 1.0	**55.78** / 52.78	↓ 5.4	53.05 / **53.19**	↑ 0.3	↓ 1.4
500K tokens							
LLMLingua-2-D (2024)	36.68 / **58.99**	↑ 60.8	33.96 / **56.71**	↑ 67.0	30.49 / **59.22**	↑ 94.3	↑ 74.0
LongLLMLingua (2024)	34.36 / **56.82**	↑ 65.4	34.86 / **54.54**	↑ 56.5	36.54 / **59.43**	↑ 62.6	↑ 61.5
BRIEF-Pro (2026)	37.42 / **55.18**	↑ 47.5	35.71 / **54.00**	↑ 51.2	39.49 / **59.43**	↑ 50.5	↑ 49.7
LLMLingua-2-H (2024)	37.31 / **59.00**	↑ 58.1	37.13 / **57.94**	↑ 56.0	33.53 / **59.58**	↑ 77.7	↑ 63.9
BEAM-RAG (2026)	53.09 / **59.10**	↑ 11.3	52.88 / **56.04**	↑ 6.0	53.08 / **59.46**	↑ 12.0	↑ 9.8
LIGHT (2026)	49.48 / **59.60**	↑ 20.4	52.38 / **56.04**	↑ 7.0	54.78 / **59.46**	↑ 8.5	↑ 12.0
TOMC w/o compilation	56.13 / **57.47**	↑ 2.4	**50.99** / 50.85	↓ 0.3	50.25 / **56.02**	↑ 11.5	↑ 4.5
1M tokens							
LLMLingua-2-D (2024)	24.67 / **58.69**	↑ 137.9	23.90 / **50.54**	↑ 111.5	24.56 / **54.47**	↑ 121.7	↑ 123.7
LongLLMLingua (2024)	32.66 / **50.91**	↑ 55.9	32.81 / **45.75**	↑ 39.4	36.94 / **54.47**	↑ 47.5	↑ 47.6
BRIEF-Pro (2026)	35.10 / **53.03**	↑ 51.1	35.66 / **46.88**	↑ 31.5	36.58 / **54.47**	↑ 48.9	↑ 43.8
LLMLingua-2-H (2024)	43.34 / **58.78**	↑ 35.6	36.01 / **51.97**	↑ 44.3	32.74 / **55.10**	↑ 68.3	↑ 49.4
BEAM-RAG (2026)	50.75 / **56.38**	↑ 11.1	48.51 / **53.40**	↑ 10.1	50.05 / **53.06**	↑ 6.0	↑ 9.1
LIGHT (2026)	51.25 / **56.59**	↑ 10.4	45.83 / **53.40**	↑ 16.5	51.15 / **53.06**	↑ 3.7	↑ 10.2
TOMC w/o compilation	50.96 / **52.68**	↑ 3.4	46.92 / **47.34**	↑ 0.9	45.90 / **49.02**	↑ 6.8	↑ 3.7

Reading. $\Delta = 100(T/B - 1)$; Mean averages reader-wise changes before rounding. Green ↑/red ↓: improvement/decline. D/H: direct/hierarchical LLMLingua-2. **Comparisons.** Baselines follow overall baseline mean; paired sets and scoring protocols vary. w/o compilation removes compiled records/instructions, retaining source text and routing. Ablations use matched scoring across dated batches; cross-reader means are descriptive. Counts and protocols: Table 7, Appendices C, F.

3.4 Memory Construction under a Token Budget

The final step combines compiled records C with source-text candidates R to construct M_c. Here R denotes the source units considered for inclusion; the reference implementation offers all selected evidence, so $R = E$. Compiled records present computed task information, while source text preserves wording and context that these records may omit. Let Pack denote the procedure that serializes and selects these candidates under budget B:

$$M_c = \text{Pack}(C, R, B), \qquad \text{Tokens}(M_c) \leq B. \tag{3}$$

Compiled state and relation records include their supporting source IDs in the serialized text, for example `[STATE] target = staging <src:u19>`. Source-text candidates retain their identifiers, for example `[RAW u19] target = staging`. Reference count records contain the computed statistics with an empty source-ID field. The budget accounts for all serialized content, including record labels, source IDs, and separators.

When C is nonempty, the reference implementation allocates $\lfloor \alpha B \rfloor$ tokens to compiled records and the remaining budget to source text, where $\alpha \in [0, 1]$. When C is empty, source text receives the entire budget. Compiled records are considered in their configured order, while source units follow their original order. Within each allocation, the procedure includes complete lines that fit and skips those that do not. Unused budget is not transferred between the two blocks. After joining the blocks, the procedure checks that $\mathrm{Tokens}(M_c) \leq B$.

4 Experiments and Results Discussion

(1) Does TOMC improve answer quality over long conversational histories? Long histories are where a fixed-budget memory must decide what to keep. We use BEAM (Tavakoli et al., 2026), which tests ten long-term memory abilities, on 600 questions from 60 conversations: 20 conversations at each of 100K, 500K, and 1M tokens, with one question per conversation for each ability. We compare TOMC with retrieval and memory systems (BEAM-RAG and LIGHT (Tavakoli et al., 2026)) and with text compressors (LongLLMLingua (Jiang et al., 2024), BRIEF-Pro (Gu et al., 2026), and direct and hierarchical LLMLingua-2 (Pan et al., 2024)) on the same histories. To isolate the compiled records, TOMC w/o compilation removes them and their instructions while keeping source text and routing fixed.

(2) Which retained information supports each task? BEAM scores do not show which part of the memory is responsible. We therefore run three targeted diagnostics from the official RULER generators (Hsieh et al., 2024), 100 examples per task: key–value retrieval (NIAH) tests whether retrievable facts survive construction, variable tracking (VT) whether state records can replace explicit answer candidates, and multi-hop QA (QA2) whether source text remains necessary. Each task compares TOMC (the state route for VT) with head–tail truncation and query selection, and then independently removes the extra source block (NIAH, QA2) or the candidate rows (VT).

(3) What does memory construction cost? Because TOMC constructs memory without auxiliary neural models, we measure construction time, neural-model use, peak GPU allocation, and peak host RAM, excluding reader inference and judging.

All experiments use GPT-5.1 (OpenAI, 2025), Rednote preview (dots studio, 2026), and DeepSeek 4.1 Flash (DeepSeek-AI, 2026) as API readers, with memory budgets of 8,192 tokens for BEAM and 4,096 estimated tokens for RULER; reader input includes prompt overhead. We report BEAM scores (0–100) on paired valid questions, scored under matched protocols within each comparison, token F1 for NIAH/QA2, and answer-item recall for VT. Baselines and readers are detailed in Appendices B.3 and B.4; subset selection, scoring, statistics, and our RULER memory conditions and metrics in Appendix D; the BEAM ablation in Appendix F; and resource measurement in Appendix G.

4.1 Does TOMC Improve Answer Quality over Long Histories?

At 500K and 1M, TOMC achieves higher mean scores in all 36 reader–baseline–length comparisons (Table 1). Its mean relative gain over LIGHT is 0.1% at 100K versus 12.0% and 10.2% at 500K and 1M. This indicates that task-oriented memory construction remains useful for the three strong API readers when answering questions over long histories.

The long-history comparisons (500K and 1M; Figure 2a) test whether executing supported task operations and retaining selected source text can improve on retrieval, memory fusion, and text compression. Against BEAM-RAG's retrieved passages and LIGHT's episodic, working, and scratchpad memories, TOMC achieves relative score gains of 9.4% and 11.0% while reducing reader input by 2.6% and 43.1%, respectively, consistent with task-operation results adding to locating and maintaining relevant information. Against LongLLMLingua's question-aware compression and hierarchical LLMLingua-2's learned token retention, TOMC improves scores by 54.7% and 56.4% with 20.8% and 42.4% less input, supporting task-specific record construction as an alternative to

selecting and compressing important tokens. Direct LLMLingua-2 shows a larger gap (94.6%) but includes 200 empty memories at 1M. Against BRIEF-Pro's question-conditioned evidence summaries, TOMC scores 46.8% higher with 238.5% more input: records with more retained source text favor answer quality, whereas generated summaries favor compact input. These system-level comparisons also differ in selection and input length; the ablation below isolates the compiled records.

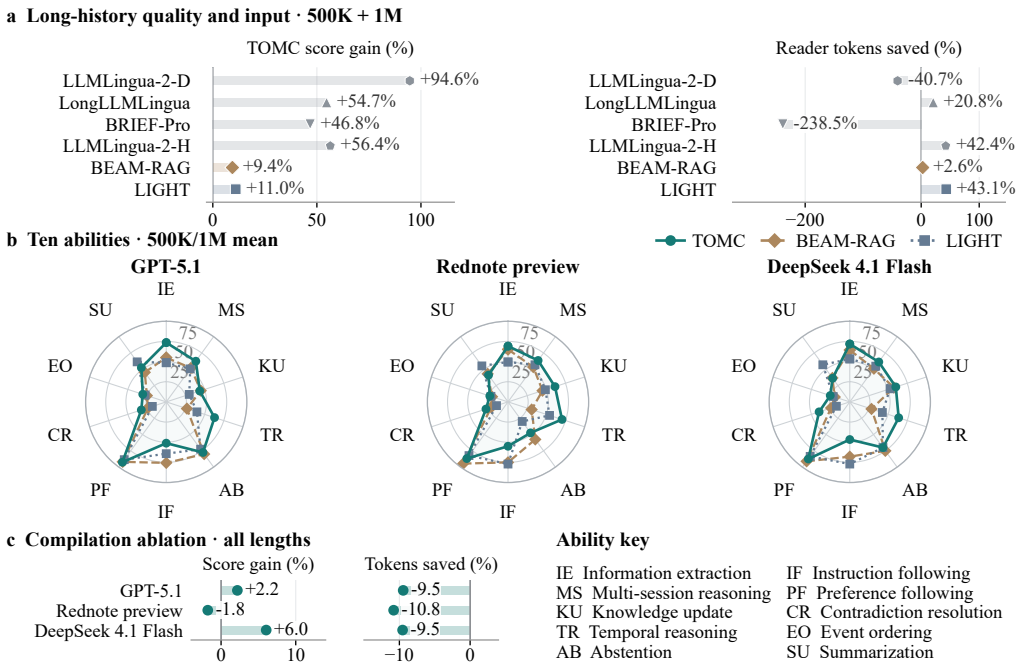

Figure 2: BEAM results. (a) TOMC score gain $100(T/B - 1)$ and input savings $100(1 - T/B)$: pool valid 500K/1M pairs within readers, then average percentages. Negative savings mean more input; gray denotes neural compression baselines. (b) Abilities average 500K/1M means equally using common valid questions and the same scoring protocol (GPT-5.1: 198/199; others: 200/200); axes span 0–100. (c) Full TOMC relative to w/o compilation across all lengths; zero is the reference.

The ability profiles are mixed (Figure 2b; Appendix C). At 500K and 1M, TOMC outperforms both BEAM-RAG and LIGHT for all three readers on temporal reasoning (by 16.3–39.6 points), information extraction (4.1–24.9), contradiction resolution (6.9–22.5), multi-session reasoning, and event ordering, and also outperforms LIGHT on knowledge update and preference following. It trails both baselines on instruction following (13.0–30.4 points), BEAM-RAG on abstention (2.5–10.0), and LIGHT on summarization (8.9–20.7). The largest gains, on temporal reasoning, information extraction, and contradiction resolution, suggest that task records can complement retrieval- and scratchpad-based memories on these question types.

Restoring compiled records and instructions changes overall scores by +1.20, -0.91, and +3.01 points for GPT-5.1, Rednote preview, and Flash, while increasing input by 9.5–10.8% (Figure 2c). Temporal reasoning (+8.33, +7.50, and +4.86 points) and instruction following improve for all three readers, including Rednote despite its overall decrease, whereas event ordering is the only ability that declines for all three (-0.79 to -1.73 points; Appendix F). Records that resolve time-dependent facts thus help even strong readers, whereas ordering questions depend on event sequences that resolved records may not preserve.

4.2 Mechanism Analysis and Ablation Studies

Table 2 and Figure 3 report task comparisons (panel A; Figure 3a,b) and independent component removals (panel B; Figure 3c,d). Unlike the BEAM ablation, these removals probe which retained information supports each task.

Table 2: RULER quality and reader-input tokens. Reader cells report score (0–100) / input tokens. Bold marks highest scores, lowest input, and largest savings within each task/pair, including ties; it does not imply significance or joint superiority. Mean score and savings average three readers.

Condition	GPT-5.1 Score / Tokens	Rednote preview Score / Tokens	DeepSeek 4.1 Flash Score / Tokens	Mean score	Tokens saved (%)
A. Task-level comparisons					
NIAH — Key–value retrieval (token F1)					
Query selection	91.75 / 3,380	50.98 / 3,611	**100.00** / 3,448	80.91	*Ref.*
Head–tail truncation	5.00 / 3,209	2.55 / 3,309	5.00 / 3,304	4.18	5.9
TOMC (ours, full)	**92.50** / **1,631**	**72.41** / **1,746**	**100.00** / **1,655**	**88.30**	**51.8**
VT — Variable-state tracking (answer-item recall)					
Query selection	58.40 / 4,370	58.20 / 4,599	23.80 / 4,375	46.80	*Ref.*
Head–tail truncation	90.80 / 4,065	87.20 / 4,082	96.00 / 4,072	91.33	8.4
State route (ours, diagnostic)	**100.00** / **449**	**100.00** / **509**	**100.00** / **478**	**100.00**	**89.2**
QA2 — Multi-hop question answering (token F1)					
Query selection	**46.55** / 3,527	**35.01** / 3,789	**60.90** / 3,570	**47.49**	*Ref.*
Head–tail truncation	24.66 / 3,551	31.77 / 3,807	7.86 / 3,593	21.43	-0.6
TOMC (ours, full)	38.70 / **2,570**	28.22 / **2,851**	55.62 / **2,687**	40.85	**25.5**
B. Component ablations: independent removal tests					
NIAH (repaired pair) — Key–value retrieval (token F1)					
TOMC (ours, full)	**85.00** / 3,399	**76.84** / 3,650	**100.00** / 3,471	**87.28**	*Ref.*
TOMC w/o extra source	79.03 / **1,911**	68.76 / **2,046**	99.00 / **1,967**	82.26	**43.7**
VT — Variable-state tracking (answer-item recall)					
State route (ours, diagnostic)	**100.00** / 449	**100.00** / 509	**100.00** / 478	**100.00**	*Ref.*
State route w/o candidates	**100.00** / **338**	**100.00** / **382**	**100.00** / **366**	**100.00**	**24.3**
QA2 — Multi-hop question answering (token F1)					
TOMC (ours, full)	**38.70** / 2,570	**28.22** / 2,851	**55.62** / 2,687	**40.85**	*Ref.*
TOMC w/o extra source	10.40 / **1,076**	9.53 / **1,236**	5.40 / **1,177**	8.44	**57.0**

Tokens include prompt overhead; means are rounded to integers. Each task starts with its token reference (Ref.), above a dashed rule: Query selection in A, full/state in B. Savings average $100(1 - T/T_{\text{ref}})$ across readers using unrounded means. NIAH in B is a repaired paired rerun with fixed records; A retains the historical comparison, whose full condition also lost compiled blocks. Query/head–tail were not rerun. VT uses the state route; routed fields may retain source text. Valid sets can differ. Counts, controls, and the packing audit are in Appendix E.

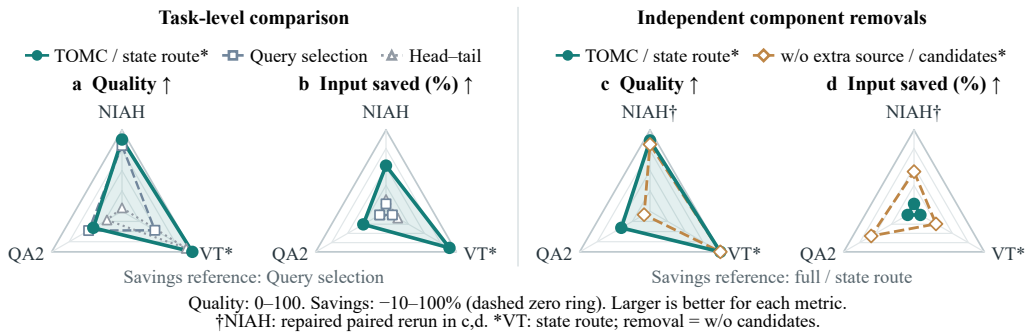

Figure 3: RULER task comparisons (a,b) and component removals (c,d). Quality: F1 (NIAH/QA2), recall (VT). Scores and within-reader savings average three readers. †NIAH uses the repaired paired rerun in c,d; a,b retain the historical comparison. Quality and savings are separate objectives. Values, batches, and valid counts: Table 2, Appendix E.

Key–value retrieval with retained source text. The archived NIAH comparison gives mean F1 88.30 versus 80.91 for query selection, with 51.8% input savings. Because these archived full memories lost their compiled blocks, the comparison reflects TOMC's selected source text: the gain comes from Rednote preview (72.41 versus 50.98), whereas GPT-5.1 and Flash match query selection with about half the input. In a separate paired rerun with repaired packing, removing extra

source lowers mean F1 from 87.28 to 82.26 while saving 43.7% input; compiled records remain fixed. Records alone thus retain most retrieval quality, but retained source still adds 6.0 and 8.1 F1 for GPT-5.1 and Rednote preview. Query/head–tail were not rerun; Appendix E reports the historical packing failure and repaired protocol.

Variable tracking succeeds without candidate rows. The state route reaches 100% recall for every reader, versus 23.8–58.4% for query selection, with 89.2% less input; removing candidates keeps 100% (a saturated tie) and saves a further 24.3%. Compiling only selected evidence E yields 20.4–20.6% recall, suggesting that query-based selection omits intermediate assignments, so dependency chains should be executed over the full history.

Semantic multi-hop QA exposes a quality–input trade-off. On QA2, full TOMC uses 25.5% less input than query selection but scores lower (40.85 versus 47.49 F1); without the extra source block, F1 drops to 8.44 (Table 2B). With E fixed, raw source text scores highest for every reader, 4.96–18.27 points above compiled records plus source, which use 22.2–22.8% less input. Structured records should therefore supplement, not replace, source passages for semantic multi-hop questions.

4.3 Computational Efficiency and Resource Requirements

Table 3: Recorded construction time, auxiliary neural stages, and peak memory use. Times keep each method's recorded unit and are compared only within the same unit. *LIGHT time covers the full 600-memory batch; its memory is measured on one writer/filter worker.

Method	Neural stage	Recorded construction time	Peak GPU allocation (GiB)	Peak host RAM (RSS, GiB)
TOMC	**None**	**4.21 s/call**	**0 (CPU)**	3.75
BEAM-RAG	BGE-small	23.94 s/conv.	0 (CPU)	1.45
LongLLMLingua	7B compressor	72.07 s/call	18.50	10.24
BRIEF-Pro	3B compressor	47.94 s/call	11.81	5.45
LIGHT*	32B AWQ writer/filter	41.53 h/batch	80.82	6.04
LLMLingua-2-D	BERT compressor	9.16 s/conv.	1.55	1.85
LLMLingua-2-H	BERT compressor	10.19 s/conv.	1.55	1.62

Table 3 reports memory construction cost, excluding reader inference and judging. **TOMC constructs memory in 4.21 s per call on CPU, without a GPU or an auxiliary neural model**. Among per-call methods, LongLLMLingua and BRIEF-Pro take 72.07 and 47.94 s (17.1× and 11.4×) and reach 18.50 and 11.81 GiB peak GPU allocation. LLMLingua-2 and BEAM-RAG build memory per conversation and LIGHT per batch, so their times are not compared per call; they rely on BERT, BGE-small, and 32B writer/filter models, and LLMLingua-2 and LIGHT reach 1.55 and 80.82 GiB peak GPU allocation, the latter for one writer/filter worker, including its KV pool. TOMC's 3.75 GiB peak host RSS exceeds that of CPU BEAM-RAG (1.45 GiB) and LLMLingua-2 (1.62–1.85 GiB). Task-oriented memory for API readers thus adds no local GPU inference; Appendix G details timing units, stage coverage, and replay protocols.

5 Conclusion

We presented TOMC, a training-free method that executes task operations while building memory and combines the resulting compiled records with selected source text under a fixed token budget. Our results answer the question posed in the introduction: executing these operations during memory construction improves answer quality without auxiliary neural-model inference. With three strong LLM API readers on BEAM histories of 100K–1M tokens, TOMC scores higher than six retrieval, memory, and compression baselines overall and in all 36 reader–baseline–length comparisons at 500K and 1M tokens, while using 32.4–36.6% fewer input tokens than LIGHT. Record construction takes 4.21 s per call on CPU, 11–17× shorter than for per-call neural compressors. Compiled records and selected source text both help: compiled records improve temporal reasoning, selected source text improves key–value retrieval and multi-hop QA, and state records built from the full history reach 100% recall on variable tracking. These results suggest that executing task operations during memory construction complements retrieval and offers an alternative to token compression when strong LLMs answer questions over long interaction histories.

AI USE STATEMENT

Large language models assisted with drafting and editing text, literature comparison, and auxiliary code checks. Benchmark scores were retained from the archived evaluation outputs. The authors are responsible for verifying the final analyses, references, and claims.

REPRODUCIBILITY STATEMENT

Appendix B gives the TOMC and baseline configurations, Appendix D the evaluation protocols and statistics, and Appendix G the resource-measurement protocol. The anonymous supplementary material contains the TOMC implementations, frozen per-question scores and API usage, table and figure scripts, and offline verification scripts for all reported results.

REFERENCES

Pratyay Banerjee, Masud Moshtaghi, Shivashankar Subramanian, Amita Misra, and Ankit Chadha. APEX-MEM: Agentic semi-structured memory with temporal reasoning for long-term conversational AI. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16470–16489, 2026. doi: 10.18653/v1/2026.acl-long.749. URL https://aclanthology.org/2026.acl-long.749/.

Xin Cheng, Xun Wang, Xingxing Zhang, Tao Ge, Si-Qing Chen, Furu Wei, Huishuai Zhang, and Dongyan Zhao. xRAG: Extreme context compression for retrieval-augmented generation with one token. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/c5cf13bfd3762821ef7607e63ee90075-Abstract-Conference.html.

DeepSeek-AI. DeepSeek-V4.1-Flash: Smarter, faster, more efficient. Official release announcement, DeepSeek API Docs, 2026. URL https://api-docs.deepseek.com/news/news260910/. September 10, 2026.

dots studio. dots3-note Preview. Hugging Face model card, 2026. URL https://huggingface.co/dots-studio/dots3-note-prev. Released August 2026 by dots studio (RedNote). Accessed 2026-09-05.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/0b276510ec2d3f6613a8b60c41ff0438-Abstract-Conference.html.

Jia-Chen Gu, Junyi Zhang, Di Wu, Yuankai Li, Kai-Wei Chang, and Nanyun Peng. BRIEF-Pro: Universal context compression with short-to-long synthesis for fast and accurate multi-hop reasoning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 14221–14241, 2026. doi: 10.18653/v1/2026.findings-acl.696. URL https://aclanthology.org/2026.findings-acl.696/.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2404.06654.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1658–1677, 2024. doi: 10.18653/v1/2024.acl-long.91. URL https://aclanthology.org/2024.acl-long.91/.

Kai Li, Xuanqing Yu, Ziyi Ni, Yi Zeng, Yao Xu, Zheqing Zhang, Xin Li, Jitao Sang, Xiaogang Duan, Xuelei Wang, Chengbao Liu, and Jie Tan. TiMem: Temporal-hierarchical memory consolidation for

long-horizon conversational agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 21700–21720, 2026. doi: 10.18653/v1/2026.findings-acl.1091. URL https://aclanthology.org/2026.findings-acl.1091/.

Ning Liao, Zihao Long, Xiaoxing Wang, Xue Yang, Yaoming Wang, Ziyuan Zhuang, Xunliang Cai, Rongxiang Weng, and Junchi Yan. ACE: Pluggable adaptive context elasticizer across agents. arXiv preprint, June 2026. URL https://arxiv.org/abs/2606.31564.

OpenAI. Introducing GPT-5.1 for developers. Official developer release announcement, 2025. URL https://openai.com/index/gpt-5-1-for-developers/. November 13, 2025.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 963–981, 2024. doi: 10.18653/v1/2024.findings-acl.57. URL https://aclanthology.org/2024.findings-acl.57/.

Mohammad Tavakoli, Alireza Salemi, Carrie Ye, Mohamed Abdalla, Hamed Zamani, and J. Ross Mitchell. Beyond a million tokens: Benchmarking and enhancing long-term memory in LLMs. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/d7f0cfa0fe759b033d5262e1bb7d4065-Abstract-Conference.html.

Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, and Tomas Pfister. Chain-of-Table: Evolving tables in the reasoning chain for table understanding. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/f53fd88a4340063ecd258c0ae9948b40-Abstract-Conference.html.

Yi Zhao, Zuchao Li, Hai Zhao, Baoyuan Qi, and Guoming Liu. DAC: A dynamic attention-aware approach for task-agnostic prompt compression. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 19395–19407, 2025. doi: 10.18653/v1/2025.acl-long.952. URL https://aclanthology.org/2025.acl-long.952/.

A Detailed Algorithms

Algorithms 2–4 detail the reference compiler's state, relation, and count computations, using the notation in Table 4. Their outputs are converted to compiled records C before packing. Reference code, settings, and differences from the experimental implementations are documented in Appendix B.

In Algorithm 2, S stores current variable values, and $P_S(v)$ lists the source IDs supporting $S(v)$. In both the state and relation kernels, MergeSources concatenates two source-ID lists and keeps each ID once, in order of first occurrence. Path records use the field order (source entity, label sequence, target entity, source IDs); $\|$ appends a label to the sequence.

Table 4: Notation for the TOMC compiler.

Symbol	Meaning	Symbol	Meaning
M	original context	q	current question
M_c	compressed context	B	memory-token budget
a	generated answer	a_{ref}	reference answer
Compress	compression procedure	LLM	fixed answering model
U	ordered source units	E	selected evidence
z	representation route	F	parsed operation records
C	compiled records	$P_S(v)$	source IDs supporting the current value of v
R	source-text candidates	Θ	fixed compiler configuration
ν	lexical tokenizer	Tokens	memory-length counter
α	compiled-memory fraction	U_{scope}	declared count scope

Algorithm 2 State compiler for ordered `SET`/`COPY` operations

Require: source-ordered `SET`/`COPY` records F
Ensure: final state S and source-ID map P_S

$S \leftarrow \varnothing$; $P_S \leftarrow \varnothing$
for $f \in F$ in source order **do**
  if $f = \texttt{SET}(v, \mathrm{val}, J, o)$ **then**
    $S(v) \leftarrow \mathrm{val}$; $P_S(v) \leftarrow J$
  else if $f = \texttt{COPY}(v, w, J, o)$ **then**
    $\mathrm{val} \leftarrow S(w)$ if defined, else $\perp$
    $J_w \leftarrow P_S(w)$ if defined, else the empty list
    $S(v) \leftarrow \mathrm{val}$; $P_S(v) \leftarrow \textsc{MergeSources}(J_w, J)$
  end if
end for
return (S, P_S)

Algorithm 3 Typed relation-path compiler

Require: `EDGE` records F, labels $(\ell_1, \ldots, \ell_k)$ with $k \geq 1$, configured $\mathrm{path_cap}$

order the `EDGE` records of each label by source entity, target entity, and source order
$\mathrm{paths} \leftarrow \{(h, (\ell_1), t, J) : \texttt{EDGE}(h, \ell_1, t, J, o) \in F\}$
for $j = 2, \ldots, k$ **do**
  $\mathrm{next_paths} \leftarrow \varnothing$
  for $(h, \mathrm{labels}, t, J) \in \mathrm{paths}$ **do**
    for $\texttt{EDGE}(t, \ell_j, t', J', o) \in F$ **do**
      add $(h, \mathrm{labels} \parallel (\ell_j), t', \textsc{MergeSources}(J, J'))$ to $\mathrm{next_paths}$
      if $|\mathrm{next_paths}| = \mathrm{path_cap}$ **then** stop expanding label ℓ_j
    end for
  end for
  $\mathrm{paths} \leftarrow \mathrm{next_paths}$
end for
sort paths by source entity, label sequence, target entity, and source IDs
return the first $\mathrm{path_cap}$ records

Algorithm 4 Occurrence and source-support count compiler

Require: declared scope U_{scope} and lexical tokenizer ν
Ensure: term-occurrence counts F_{occ} and source-unit counts F_{sup}
initialize $F_{\text{occ}}, F_{\text{sup}}$ to zero
for $u \in U_{\text{scope}}$ **do**
    $\text{terms} \leftarrow \nu(u)$
    for each occurrence of term in terms **do**
        $F_{\text{occ}}(\text{term}) \leftarrow F_{\text{occ}}(\text{term}) + 1$
    end for
    for each distinct term in terms **do**
        $F_{\text{sup}}(\text{term}) \leftarrow F_{\text{sup}}(\text{term}) + 1$
    end for
end for
return $(F_{\text{occ}}, F_{\text{sup}})$

B Implementation Details

The reference compiler (`code/tomc_reference.py`) implements the operations in Section 3 and Appendix A. BEAM and RULER use separate task-specific implementations, described below and in Appendix D.5. All settings are defined in the released code.

B.1 BEAM constructor

BEAM uses `beam_tomc_stabilized_v2`, which constructs one memory per question on CPU, with its optional language-model compiler disabled, under an 8,192-token `cl100k_base` cap. Histories above two million characters are first reduced by a deterministic head, tail, and query-signal prefilter (380 of 600 memories). The state operator extracts the last assignment by regular expressions, without snapshot `COPY`; graph records are within-unit entity co-occurrences, without typed path joins; counts cover the working context. The router selects the compiled branch on variable, frequency, common-word, needle, aggregation, numeric, or extreme-length cues and otherwise takes the raw-text route: 432 memories are compiled and 168 take the raw-text route. Compiled memories reserve 3,440 tokens for records and fill the remainder with lexically selected source text.

B.2 Reference compiler defaults

Whitespace-stripped nonempty lines receive source-order IDs. The selection score is $s(u, q) = L + 0.8I_{\text{id}} + 0.6O + 0.25A$, where L is the fraction of distinct query terms in u, I_{id} the fraction of distinct query identifiers appearing as case-sensitive substrings, and O, A indicate operation cues and structured-record prefixes. Greedy selection subtracts 0.30 times the maximum lexical-set Jaccard similarity to a selected unit, breaks ties by source order, and selects at most 128 units; it stops when the best penalized unit has a nonpositive base score, then adds immediate neighbors and returns units in source order. Routing tests State, Count, then Relation, with Raw as default. State, relation, and count records are capped at 96, 120, and 80, and packing uses $\alpha = 0.42$ and $R = E$.

B.3 Baseline configurations

All BEAM methods receive the same histories and questions under the same 8,192-token memory cap. **BEAM-RAG** uses BEAM's dense retriever: `BAAI/bge-small-en-v1.5` and FAISS retrieve five dialogue pairs on CPU, concatenated in rank order. **LIGHT** uses the same BEAM release with `Qwen/Qwen2.5-32B-Instruct-AWQ` for memory writing and query filtering, and GPT-5.1 in place of GPT-4.1-mini for scratchpad summarization; episodic, working, and scratchpad content share the budget 2:1:1. **LongLLMLingua** uses the official code with `NousResearch/Llama-2-7b-chat-hf`, 2,048-token history blocks, question-aware ranking, and `target_token=8192`; as in the official pipeline, its response recovery restores compressed spans in the reader answer. **BRIEF-Pro** passes up to 32,768 tokens of BM25+BGE-ranked candidates and the question to the released 3B Auto summarizer (`uclanlp/brief-pro`). **LLMLingua-2** uses `microsoft/llmlingua-2-bert-base-multilingual-cased-meetingbank` with `target_token=8192`; the direct configuration (LLMLingua-2-D) compresses the history in one pass, and the hierarchical one

(LLMLingua-2-H) compresses 65,536-token blocks and then their concatenation. Each conversation's memory is reused across its questions.

B.4 Reader configurations

Responses identify GPT-5.1 as `gpt-5.1-2025-11-13`, Rednote preview as `dots3-note-prev`, and DeepSeek 4.1 Flash as `deepseek-flash`; these identifiers record served endpoints, not immutable checkpoints. BEAM output is capped at 2,048 tokens and RULER output at 128. Flash uses temperature zero with thinking disabled, Rednote preview disables thinking, and GPT-5.1 sets neither temperature nor reasoning effort. The compression comparisons use earlier reader batches (Appendix D.4).

Code and data. The anonymous supplementary material contains the reference implementation and tests, the experimental BEAM and RULER code, frozen configurations, per-question scores and API usage, statistical analyses, table and figure scripts, and offline verification scripts for all reported results. Model weights, provider receipts, and full conversations are excluded; benchmark data come from their original releases.

C Detailed BEAM Results

Table 5 reports paired TOMC / baseline means by memory ability and history length for BEAM-RAG and LIGHT. Bold uses unrounded means; superscripts give valid counts below the planned 20 per ability and length (200 for Overall).

Table 5: BEAM ability results by history length for BEAM-RAG and LIGHT. Each cell reports TOMC / Baseline (0–100).

	GPT-5.1		Rednote preview		DeepSeek 4.1 Flash	
Memory ability	BEAM-RAG	LIGHT	BEAM-RAG	LIGHT	BEAM-RAG	LIGHT
100K						
Information extraction	**78.3**/65.0	**78.3**/76.7	**87.1**/63.8	**87.1**/78.3	**81.7**/58.3	81.7/**85.0**
Multi-session reasoning	**75.0**/54.1	**75.0**/69.6	**68.0**/61.4	**68.0**/67.1	**62.4**/52.4	62.4/**68.4**
Knowledge update	**60.0**/22.5	**60.0**/52.5	**62.5**/42.5	62.5/**65.0**	**65.0**/45.0	**65.0**/60.0
Temporal reasoning	**70.0**/55.0	**70.0**/50.0	**80.0**/67.5	**80.0**/65.0	**57.5**/**57.5**	57.5/**67.5**
Abstention	70.0/**80.0**	70.0/**80.0**	35.0/**60.0**	**35.0**/30.0	65.0/**80.0**	65.0/**70.0**
Instruction following	57.5/**72.5**	57.5/**80.0**	67.5/**75.0**	67.5/**75.0**	60.0/**62.5**	60.0/**70.0**
Preference following	95.0/**100.0**	95.0/**97.5**	**90.0**/**90.0**	90.0/**92.5**	**97.5**/90.0	**97.5**/92.5
Contradiction resolution	**32.5**/31.2	**32.5**/18.8	**23.8**/**23.8**	**23.8**/**23.8**	23.8/**25.0**	**23.8**/**23.8**
Event ordering	**23.6**/19.2	**23.6**/21.6	**20.2**/18.8	**20.2**/20.1	**22.3**/18.4	**22.3**/19.2
Summarization	**60.7**/36.2	**60.7**/58.4	**48.5**/32.1	48.5/**53.7**	**47.0**/26.5	47.0/**53.8**
Overall	**62.3**/53.6	**62.3**/60.5	**58.3**/53.5	**58.3**/57.1	**58.2**/51.6	58.2/**61.0**
500K						
Information extraction	**81.6**/59.2[19]	**82.5**/50.0	**82.5**/71.2	**82.5**/58.3	**85.0**/71.2	**85.0**/62.5
Multi-session reasoning	**70.0**/58.8	**70.0**/56.2	**65.0**/54.2	**65.0**/60.4	**60.8**/60.4	60.8/**61.2**
Knowledge update	**47.5**/40.0	**47.5**/25.0	**55.0**/40.0	**55.0**/50.0	**72.5**/55.0	**72.5**/50.0
Temporal reasoning	**57.5**/25.0	**57.5**/30.0	**63.3**/31.7	**63.3**/55.8	**55.0**/31.7	**55.0**/47.5
Abstention	75.0/**80.0**	**75.0**/65.0	50.0/**70.0**	**50.0**/40.0	65.0/**70.0**	65.0/**70.0**
Instruction following	45.0/**71.7**	47.4/**68.4**[19]	55.0/**75.8**	55.0/**76.7**	45.0/**63.3**	45.0/**75.0**
Preference following	**100.0**/92.5	**100.0**/92.5	92.5/**97.5**	**92.5**/85.0	**97.5**/92.5	**97.5**/85.0
Contradiction resolution	**28.8**/25.0	**28.8**/18.8	**28.8**/20.0	**28.8**/16.2	**46.2**/25.0	**46.2**/17.5
Event ordering	**32.1**/26.6	**32.1**/27.3	**23.5**/22.0	23.5/**24.4**	**28.7**/20.3	**28.7**/20.9
Summarization	**54.7**/52.5	54.7/**62.5**	44.9/**46.4**	44.9/**56.9**	38.9/**41.4**	38.9/**58.2**
Overall	**59.1**/53.1[199]	**59.6**/49.5[199]	**56.0**/52.9	**56.0**/52.4	**59.5**/53.1	**59.5**/54.8
1M						
Information extraction	**65.5**/50.6	**65.5**/49.8	55.8/**59.0**	**55.8**/40.6	**58.7**/57.4	**58.7**/44.1
Multi-session reasoning	**54.7**/45.2	**54.7**/45.0	**62.0**/52.5	**62.0**/53.3	**61.6**/41.4	**61.6**/48.3
Knowledge update	40.0/**50.0**	**40.0**/35.0	**67.5**/50.0	**67.5**/47.5	47.5/**60.0**	47.5/**55.0**
Temporal reasoning	**68.4**/28.9[19]	**70.0**/52.5	**77.5**/30.0	**77.5**/52.5	**72.5**/25.0	**72.5**/37.5

Continued on the next page

Table 5 — continued

	GPT-5.1		Rednote preview		DeepSeek 4.1 Flash	
Memory ability	BEAM-RAG	LIGHT	BEAM-RAG	LIGHT	BEAM-RAG	LIGHT
Abstention	**80.0**/**80.0**	**80.0**/**80.0**	**45.0**/**45.0**	**45.0**/20.0	75.0/**80.0**	**75.0**/70.0
Instruction following	55.0/**75.8**	55.0/**60.0**	55.0/**74.2**	55.0/**77.5**	48.3/**72.5**	48.3/**79.2**
Preference following	85.0/**91.2**	**85.0**/84.6	81.2/**91.7**	**81.2**/79.6	77.5/**89.6**	77.5/**82.1**
Contradiction resolution	**36.2**/22.5	**36.2**/17.5	**28.8**/23.8	**28.8**/13.8	**33.8**/23.8	**33.8**/17.5
Event ordering	**29.1**/23.9	**29.1**/27.9	**23.9**/19.6	**23.9**/20.1	**22.3**/18.9	**22.3**/22.2
Summarization	**50.4**/38.1	50.4/**60.2**	37.2/**39.5**	37.2/**53.4**	**33.5**/32.0	33.5/**55.6**
Overall	**56.4**/50.7[199]	**56.6**/51.2	**53.4**/48.5	**53.4**/45.8	**53.1**/50.1	**53.1**/51.1

C.1 Reader input

Table 6 reports API-reported reader input, including prompt overhead, on the 10,731 valid pairs of Table 1. History tiers contain different conversations.

Table 6: BEAM reader-input change of TOMC relative to each baseline (%), on the valid pairs of Table 1. Negative values mean TOMC uses fewer tokens. Each reader's change is $100(\bar{T}_{\mathrm{TOMC}}/\bar{T}_{\mathrm{baseline}} - 1)$ from unrounded means; equal-reader columns and the score gain average the three readers.

	Score gain	Overall input, by reader			Equal-reader input			
Baseline	(points)	GPT-5.1	Rednote	Flash	Overall	100K	500K	1M
LLMLingua-2-D	+22.28	+8.4	+6.6	+8.6	+7.9	-18.5	-27.1	+3475.9*
LLMLingua-2-H	+17.48	-33.8	-35.7	-33.6	-34.3	-18.8	-41.7	-43.0
LongLLMLingua	+17.90	-11.8	-6.0	-10.1	-9.3	+13.2	-19.8	-21.8
BRIEF-Pro	+17.02	+265.6	+287.7	+273.1	+275.5	+342.1	+277.6	+205.4
BEAM-RAG	+5.46	+9.7	+17.2	+12.0	+13.0	+44.9	-0.2	-4.9
LIGHT	+3.71	-36.6	-32.4	-35.3	-34.8	-18.1	-42.0	-44.3

*All 200 direct LLMLingua-2 memories at 1M are empty; their frozen scores are retained. Input is API-reported and includes prompt overhead; it excludes construction and judging. Scoring protocols differ across baselines (Table 7), so score gains are compared within rows.

D Evaluation Protocols and Statistics

D.1 BEAM evaluation set

The BEAM release provides 100K, 500K, 1M, and 10M tiers. All ten 10M conversations appeared in an earlier evaluation, so we exclude that tier and draw questions from the 90 conversations not used there: all 20 at 100K and 20 of 35 at each of 500K and 1M, in a seeded SHA-256 order. For each conversation and ability, we take the first of its two questions in exact-text SHA-256 order. This yields 600 questions balanced across three tiers and ten abilities, selected without answers, rubrics, difficulty labels, or model outputs. The BEAM memory cap of 8,192 `cl100k_base` tokens includes memory headers and separators but excludes the question and system prompt.

D.2 BEAM scoring

Each entry of Table 1 averages scores over common valid questions for one TOMC–baseline pair and reader, multiplied by 100; missing answers and invalid scores are excluded. Protocols and valid sets differ across entries (Table 7), so absolute scores are compared within pairs. BEAM-RAG and LIGHT use BEAM's released single-answer scoring procedure (revision `3e12035`) with a DeepSeek 4.1 Flash judge, without method identity or a competing answer; event ordering uses normalized Kendall τ_b, $(\tau_b + 1)/2$. GPT-5.1 and Rednote preview comparisons with LongLLMLingua and BRIEF-Pro use paired DeepSeek V4 Pro judgments (Appendix D.4); GPT-5.1's two LLMLingua-2 configurations and Rednote preview's hierarchical configuration use paired Flash judgments; the remaining compression comparisons use the native single-answer procedure. Each comparison uses TOMC scores from its own protocol.

Table 7: Overall valid question pairs and scoring protocols for the main BEAM table. Each comparison planned 600 questions in total, with 200 per history-length tier.

Baseline	GPT-5.1	Rednote preview	DeepSeek 4.1 Flash
LLMLingua-2-D (2024)	599 (C)	599 (E)	592 (D)
LongLLMLingua (2024)	593 (B)	595 (B)	600 (D)
BRIEF-Pro (2026)	595 (B)	596 (B)	600 (D)
LLMLingua-2-H (2024)	600 (C)	592 (C)	573 (D)
BEAM-RAG (2026)	598 (A)	600 (A)	600 (A)
LIGHT (2026)	599 (A)	600 (A)	600 (A)

A: current BEAM-RAG/LIGHT, native single-answer Flash scoring. B: archived paired Pro scoring. C: archived paired Flash scoring. D: archived Flash-reader answers, native single-answer Flash scoring. E: Rednote direct LLMLingua-2 follow-up, native single-answer Flash scoring.

D.3 Paired inference

For BEAM-RAG/LIGHT, confidence intervals use 10,000 conversation-cluster bootstrap draws stratified by history tier, and two-sided p-values use 100,000 paired sign flips over 60 conversations with Holm correction over the three readers of each comparator. Missingness-sensitive intervals allow each missing score difference to range over $[-1, 1]$. Reader-input savings are $1 - \sum \text{input}_{\text{TOMC}} / \sum \text{input}_{\text{baseline}}$ on the same valid pairs.

Table 8: Paired TOMC-minus-baseline differences for BEAM-RAG/LIGHT. Δ is the mean on valid pairs in percentage points. The 95% interval is the conversation-cluster bootstrap interval of the full-grid mean with missing differences set to zero; the missingness-sensitive interval lets each missing difference range over $[-1, 1]$. Holm correction covers the three readers of each baseline (Appendix D.3).

Baseline	Reader	Valid pairs	Δ	95% interval	Missingness-sensitive	p_{Holm}
BEAM-RAG	GPT-5.1	598	+6.78	[+3.90, +9.61]	[+3.39, +9.94]	<0.001
BEAM-RAG	Rednote preview	600	+4.27	[+1.36, +7.23]	[+1.36, +7.23]	0.005
BEAM-RAG	DeepSeek 4.1 Flash	600	+5.34	[+2.58, +8.20]	[+2.58, +8.20]	0.001
LIGHT	GPT-5.1	599	+5.73	[+2.81, +8.71]	[+2.57, +8.86]	0.002
LIGHT	Rednote preview	600	+4.14	[+1.15, +7.00]	[+1.15, +7.00]	0.017
LIGHT	DeepSeek 4.1 Flash	600	+1.26	[-1.65, +4.20]	[-1.65, +4.20]	0.414

D.4 Compression comparison batch

The GPT-5.1 and Rednote preview comparisons with LongLLMLingua and BRIEF-Pro use the same 600 questions; all reader answers, including TOMC's, were generated in this batch with a 2,048-token output cap and a shared answer prompt. DeepSeek V4 Pro (temperature 0, thinking disabled) receives the question, official rubric, and two anonymous answers in balanced positions, scores each criterion in $\{0, 0.5, 1\}$, and averages them. Each comparison passed its prespecified agreement checks on a 60-question calibration panel. Holm adjustment uses the batch's prespecified family of 16 reader–comparator comparisons, with comparisons not run in this batch entered as $p = 1$.

D.5 RULER conditions and metrics

We use RULER's `niah_multikey_1`, `vt`, and `qa_2` generators (Hsieh et al., 2024) (upstream commit `38da79d`, seed 42, maximum sequence length 65,536) and evaluate the first 100 examples per task. NIAH places four key–value pairs in an essay haystack and queries one key; VT uses one four-hop assignment chain in noise; QA2 uses HotpotQA questions with their context and distractor documents. Fixed-evidence conditions share selected evidence E; the others use the full history M (Table 10). The RULER implementation resolves assignment chains through a final alias map and extracts regular-expression relation edges. Its budget uses the estimate $\max(1, \lfloor |\text{text}|/3.7 \rfloor + 1)$, so 4,096 estimated tokens is not an exact API-token cap; routed fields may contain source text even without an added source block. VT reports answer-item recall (extra items are not penalized); NIAH

Table 9: Compression comparisons on matched valid pairs (0–100). The 95% interval is the conversation-cluster bootstrap interval widened by the range missing pairs could contribute; bold differences have intervals excluding zero. p_{Holm} uses the family in Appendix D.4. Reader input uses the same valid pairs; memory length averages all 600 constructions.

Method	Mem. tokens	TOMC	Comp.	Δ	95% interval	p_{Holm}	Reader input TOMC / comp.
GPT-5.1							
LongLLMLingua	5,838	55.03	35.43	**+19.60**	$[+14.81, +23.95]$	0.00016	5,275 / 5,979
BRIEF-Pro	1,312	54.71	36.40	**+18.31**	$[+13.71, +22.59]$	0.00016	5,269 / 1,441
Rednote preview							
LongLLMLingua	5,838	50.28	34.34	**+15.93**	$[+11.77, +19.76]$	0.00016	5,731 / 6,099
BRIEF-Pro	1,312	51.03	35.30	**+15.73**	$[+11.58, +19.58]$	0.00016	5,725 / 1,477

and QA2 report normalized token F1. These metrics are specific to our diagnostic evaluation rather than the official RULER evaluator.

Table 10: RULER conditions. M is the full supplied history and E the selected evidence, including source tags. Listed fields are offered to final packing, which can omit fields that do not fit. Results appear in Table 2 and Appendix E.

Condition	Input	Fields offered to final packing	Extra raw block
Raw text	E	Selected source text	None
Compiled	E	Routed state, source-text, edge, or count fields	None
Compiled + source	E	Same routed construction, plus query-selected text from E	From E
Head–tail	M	Head and reverse-ordered tail, each selected under half the budget	None
Query selection	M	Query-ranked source lines with IDs	None
TOMC (full)	M	Routed fields plus query-selected source text; evaluated on NIAH/QA2	From M
TOMC w/o extra source	M	Same routed construction, without the separate source block; called State route on VT	None
State route w/o candidates	M	VT only: value-to-variable rows and recent assignments with source IDs, without explicit answer-candidate rows	None

E Complete RULER Results

Table 11 reports the 63 task–reader–condition cells used in the paper (6,294 of 6,300 planned answers valid; six QA2 cells have 99). Conditions are defined in Table 10. For NIAH/QA2, TOMC (full) adds a separate source block and TOMC w/o extra source omits it; for VT, the state route has no separate source block, and State route w/o candidates also removes answer-candidate rows.

Historical NIAH packing failure. Reconstructing the archived NIAH memories with the original implementation shows that the key–value block exceeds the 4,096-token estimate and is discarded as a whole on 99 of 100 examples, so the historical TOMC (full) memories contain only selected source text and the TOMC w/o extra source memories only headers (rows marked †). Their frozen scores are kept; Table 2A uses the historical full row, and the component ablation uses the repaired rerun below.

Table 11: RULER quality and reader-input tokens for the conditions used in the paper. Scores are 0–100 (VT answer-item recall; NIAH/QA2 token F1); tokens are mean API-reported reader input, including prompt overhead. †Historical NIAH conditions affected by the packing failure; Table 2B uses the repaired pair in Table 12.

	GPT-5.1		Rednote preview		DeepSeek 4.1 Flash	
Condition	Score	Tokens	Score	Tokens	Score	Tokens
VT — Answer-item recall						
Fixed evidence E						
Raw text	9.80	4,356	20.40	4,582	2.60	4,362
Compiled	20.60	430	20.40	488	20.60	458
Compiled + source	20.60	2,485	20.60	2,678	20.60	2,515
Full history M						
Head–tail	90.80	4,065	87.20	4,082	96.00	4,072
Query selection	58.40	4,370	58.20	4,599	23.80	4,375
State route	100.00	449	100.00	509	100.00	478
State route w/o candidates	100.00	338	100.00	382	100.00	366
NIAH — Token F1						
Fixed evidence E						
Raw text	94.75	3,352	52.62	3,585	100.00	3,424
Compiled	69.00	1,901	76.05	2,056	99.00	1,925
Compiled + source	96.25	3,470	66.05	3,773	100.00	3,509
Full history M						
Head–tail	5.00	3,209	2.55	3,309	5.00	3,304
Query selection	91.75	3,380	50.98	3,611	100.00	3,448
TOMC w/o extra source†	0.00	143	0.00	142	0.00	151
TOMC (full)†	92.50	1,631	72.41	1,746	100.00	1,655
QA2 — Token F1						
Fixed evidence E						
Raw text	70.74	3,514	70.57	3,769	67.18	3,552
Compiled	47.69	2,089	40.20	2,235	43.28	2,133
Compiled + source	59.85	2,714	52.29	2,919	62.22	2,765
Full history M						
Head–tail	24.66	3,551	31.77	3,807	7.86	3,593
Query selection	46.55	3,527	35.01	3,789	60.90	3,570
TOMC w/o extra source	10.40	1,076	9.53	1,236	5.40	1,177
TOMC (full)	38.70	2,570	28.22	2,851	55.62	2,687

E.1 Repaired NIAH paired ablation

The repaired rerun uses the same 100 NIAH examples. It reserves the source block and header, then packs complete compiled records in their original priority order; the removal arm deletes only the source block, keeping the compiled records byte-identical, without refilling the budget. All 600 answers (two conditions, three readers) are valid. Mean F1 is 87.28 for full TOMC and 82.26 without extra source, and removal saves 43.69% input (Table 12). On the 99 needle-route examples, removing extra source lowers F1 by 5.02 points for GPT-5.1 and 7.15 points for Rednote preview, whereas Flash scores 100 in both arms. Query selection and head–tail were not rerun, so Table 2A retains the historical comparison.

Table 12: Repaired NIAH paired ablation: 100 common valid examples per reader. Full and w/o extra source retain identical compiled records. F1 and EM are percentages; input includes API prompt overhead.

Reader	Token F1		Exact match		Mean input tokens	
	Full	w/o source	Full	w/o source	Full	w/o source
GPT-5.1	85.00	79.03	80.00	74.00	3,399	1,911
Rednote preview	76.84	68.76	72.00	63.00	3,650	2,046
DeepSeek 4.1 Flash	100.00	99.00	100.00	99.00	3,471	1,967

F BEAM Compilation Ablation

TOMC w/o compilation removes the compiled records and their accompanying instructions from each frozen TOMC memory, keeps the source-text suffix byte for byte, and does not refill the freed budget; routing and selection are unchanged. It changes the 432 compiled memories and leaves the 168 raw-text-route memories identical (Appendix B.1), reducing mean memory length from 5,192 to 4,724 tokens. GPT-5.1 and Rednote preview answer both arms anew on the 600 questions, and pairs are scored with the paired DeepSeek V4 Pro protocol of Appendix D.4 (597 and 595 valid pairs). With conversations as independent units, full TOMC minus TOMC w/o compilation is +1.20 points for GPT-5.1 (missingness-expanded 95% interval $[-1.45, +3.88]$) and -0.91 for Rednote preview ($[-3.43, +1.63]$); both Holm-adjusted p-values are 0.575. On the 168 unchanged memories, where both arms receive identical input, differences of +0.69 and -1.02 reflect answer and judging variation (Figure 4b).

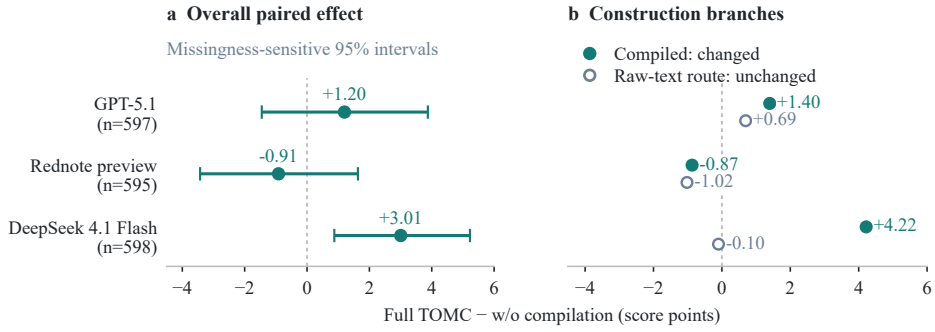

Figure 4: BEAM compilation ablation. (a) Full TOMC minus TOMC w/o compilation, with missingness-expanded 95% conversation-bootstrap intervals. (b) Differences on the 432 changed compiled memories and the 168 unchanged raw-text-route memories; the latter reflect answer and judging variation alone. Subgroups are descriptive; Flash comes from a separate batch.

DeepSeek 4.1 Flash answers the same memory pairs in a separate batch scored with the same Pro protocol (598 valid pairs): full TOMC scores 52.74 versus 49.74, a difference of +3.01 points (95% interval $[+0.88, +5.23]$; sign-flip $p = 0.00365$), with +4.22 on changed and -0.10 on unchanged memories. These Flash answers were first scored by a Flash judge and then rejudged by Pro for protocol compatibility with the other readers; Pro lowers the difference by 0.25 points.

Ability-level changes. Across all history tiers, temporal reasoning and instruction following are the only abilities that improve for all three readers when compiled records and instructions are restored: by 8.33, 7.50, and 4.86 points and by 0.28, 1.53, and 7.08 points for GPT-5.1, Rednote preview, and Flash (60 valid pairs each). Event ordering is the only ability that declines for all three (-0.92, -0.79, and -1.73 points). These ability-level differences are descriptive.

G Construction Resources

Table 13 reports construction time and memory, excluding reader inference and judging.

TOMC. Six fresh-process CPU replays, two per history tier, reproduced their constructed memories exactly. Construction calls took 0.076–7.739 s (mean 4.21 s), excluding imports, input loading, and tokenizer warm-up; the peak whole-process RSS was 3.75 GiB. The replays ran on Linux x86-64 with CUDA disabled.

Baselines. BEAM-RAG built 60 conversation indexes in 23.94 s each on CPU, including chunking, embedding, and FAISS; queries took 9.71 ms each. The 600 original LongLLMLingua and BRIEF-Pro compression calls averaged 72.07 and 47.94 s, excluding model loading and candidate construction. LLMLingua-2 cache files record 9.16 s (direct) and 10.19 s (hierarchical) per conversation; the direct mean includes the 20 1M conversations whose memories are empty. LIGHT's 600

Table 13: Recorded construction time and local-stage memory use. Timing rows retain their original stages and units; memory replay coverage is described in the text.

(a) Recorded construction time

Method	Stage	Coverage	Mean or batch time
TOMC	Constructor replay	6 calls	4.21 s/call
BEAM-RAG	Index construction	60 conversations	23.94 s/conversation
BEAM-RAG	Query retrieval	600 queries	9.71 ms/query
LongLLMLingua	Compression	600 calls	72.07 s/call
BRIEF-Pro	Compression	600 calls	47.94 s/call
LIGHT	Observed construction span	600 memories	41.53 h/batch
LLMLingua-2-D	Compression	60 conversations	9.16 s/conversation
LLMLingua-2-H	Compression	60 conversations	10.19 s/conversation

(b) Observed local-stage memory peaks (GiB)

Method	Measured local stage	CUDA allocated	Host RSS
TOMC	None	0 (CPU)	3.75
BEAM-RAG	BGE-small	0 (CPU)	1.45
LongLLMLingua	7B compressor	18.50	10.24
BRIEF-Pro	3B compressor	11.81	5.45
LIGHT (worker)	32B AWQ writer/filter	80.82	6.04
LLMLingua-2-D	BERT compressor	1.55	1.85
LLMLingua-2-H	BERT compressor	1.55	1.62

Timing stages differ across methods; these are not matched latency measurements. Compressor memory values are maxima over six fresh-process replays per configuration; TOMC/BEAM-RAG retain their CPU records. Host RSS is a lifetime process high-water mark for the compressors, TOMC and BEAM-RAG; LIGHT reports the maximum across sampled worker-process-tree totals, which may miss brief peaks. LIGHT replays 128 writer/filter requests on one worker; its CUDA allocation includes the configured KV pool, not a minimum hardware requirement. BRIEF candidate construction and LIGHT's BGE/API-summary stages are outside these memory replays.

memories spanned 41.53 h of wall time from the first writer dispatch to the final memory, with 68,484 writer, 81,480 filter, and 334 summarization calls on two GPU workers.

Memory replays. Each compressor configuration is replayed in fresh processes on the six histories and questions used for TOMC, on an NVIDIA RTX PRO 6000 Blackwell Max-Q GPU; CUDA peaks are read inside the model process and host memory is the lifetime RSS high-water mark. The memory columns report the largest value across the six samples. LIGHT's memory comes from replaying 128 archived writer/filter requests on one vLLM worker with its original settings; its CUDA peak includes the preallocated KV pool and is not a minimum-VRAM estimate.

H Task-State Size

For questions asking only for the final values of a fixed set of variables, executing the updates leaves one value per variable, regardless of the number of updates. A counting argument makes this precise. Let $\mathcal{M}$ be the set of histories, $\mathcal{Q}$ a fixed question set, and $g(M, q)$ the reference answer for $M \in \mathcal{M}$ and $q \in \mathcal{Q}$. Call two histories equivalent, $M \sim_{\mathcal{Q}} M'$, if $g(M, q) = g(M', q)$ for every $q \in \mathcal{Q}$, and let $N_{\mathcal{Q}} \geq 1$ be the finite number of equivalence classes. Consider a deterministic encoder m that maps each history to a b-bit string before the question is known, and a decoder that must recover $g(M, q)$ from $m(M)$ and q for every M and q. Histories in different classes cannot share an encoding, because some question requires different answers from identical decoder inputs. Since b bits provide at most 2^b encodings,

$$b \geq \lceil \log_2 N_{\mathcal{Q}} \rceil , \tag{4}$$

and assigning one codeword per class attains the bound. If the questions depend only on the final values of n known variables with values in a finite set $\mathcal{V}$, all final assignments are possible, and the questions distinguish any two assignments, then $N_{\mathcal{Q}} = |\mathcal{V}|^n$ and Equation 4 gives $\lceil n \log_2 |\mathcal{V}| \rceil$ bits, independent of the number of updates. TOMC instead constructs M_c for the given question and represents the computed values as text, so variable names, source IDs, and retained source text take additional space (Section 3.4).