NanoMem: Iterative Evidence Synthesis for Temporal-Causal Reasoning in Agent Memory

A Master's Thesis on Long-Term Memory for LLM Agents

Master's Thesis in Physics

Youliang Zhu

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden May 2026

Master's thesis 2026

NanoMem: Iterative Evidence Synthesis for Temporal-Causal Reasoning in Agent Memory

A Master's Thesis on Long-Term Memory for LLM Agents

Youliang Zhu

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden May 2026

NanoMem: Iterative Evidence Synthesis for Temporal-Causal Reasoning in Agent Memory
A Master's Thesis on Long-Term Memory for LLM Agents
Youliang Zhu

Supervisor: Mats Granath and Sanidhya Kashyap
Examiner: Mats Granath

Master thesis 2026
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden May 2026

NanoMem: Iterative Evidence Synthesis for Temporal-Causal Reasoning in Agent Memory
A Master's Thesis on Long-Term Memory for LLM Agents
Youliang Zhu
Department of Physics
Chalmers University of Technology

Abstract

Long-term conversational agents require memory systems that can recover evidence from personal histories spanning multiple sessions. A central challenge is the evidence addressability gap: the retrieval cue needed for the final answer may not be present in the original query, and must instead be inferred from intermediate evidence discovered in earlier retrieval rounds. Existing approaches either conflate session time with the time at which the described event occurred, or lack a mechanism to carry temporal inferences across retrieval steps to reach indirectly addressable evidence.

This thesis investigates how a long-term conversational memory system can construct compact and temporally grounded evidence at query time, without relying on heavy write-time processing that fixes interpretations before the future query is known. I introduce NanoMem, a memory evidence synthesis framework that reformulates temporal memory search as iterative evidence construction. The core mechanism is a Temporal Evidence Pool that accumulates structured event records with separate event-time and session-time fields. At ingestion, NanoMem applies lightweight deterministic normalization to resolve relative temporal expressions. At search time, a Planner decomposes the query into retrieval cues, and a trainable Synthesizer extracts temporally grounded events from retrieved sessions and judges whether the accumulated evidence is sufficient to answer the query. When it is not, the inferred events feed back to the Planner to drive the next retrieval round.

The Synthesizer is trained with GRPO using multi-level rewards covering answer correctness, evidence compactness, output format, and a per-round sufficiency verdict that provides direct process supervision for the stop-or-continue decision. A new diagnostic benchmark, TimeMemEval, is introduced to evaluate hidden-dependency temporal bridge search, where the answer-bearing evidence is reachable only after an intermediate temporal anchor has been established.

NanoMem is evaluated on LoCoMo, LongMemEval, and TimeMemEval. On all three benchmarks, NanoMem achieves the best accuracy among all evaluated methods, improving over the strongest prior baselines by 12.5, 11.8, and 31.0 percentage points, respectively. GRPO training also reduces returned evidence length by 59.5% and 66.3% on LoCoMo and LongMemEval relative to the untrained variant, while simultaneously improving accuracy. The results support the thesis position that query-time evidence synthesis, structured temporal grounding, and reward-aligned sufficiency judgment together address the evidence addressability gap more effectively than full-context prompting or write-time memory construction.

Keywords: agent memory, large language models, evidence synthesis, conversational

memory, iterative retrieval, temporal reasoning, reinforcement learning, GRPO.

Acknowledgements

I would first like to express my sincere gratitude to Mats Granath and Per Thoren for their support throughout my master's thesis work. Mats Granath, my supervisor and examiner, provided academic guidance, responded to my questions, and supported my decisions during the research and writing process. Per Thoren offered important administrative and procedural support throughout the programme. Both were generous with their time and help, and I am grateful to each of them for making the thesis process manageable while I was working in Switzerland.

I am also very grateful to my supervisor Sanidhya Kashyap at EPFL RS3Lab. His support provided the experimental conditions that made this work on agent memory possible, and his guidance during the research and paper writing process helped shape the direction of the thesis. I would also like to thank EPFL and RS3Lab for providing a stimulating research environment in which this project could be carried out.

I owe special thanks to Yueyang Pan, a PhD student at RS3Lab, with whom I worked closely on the agent memory project that became the foundation of this thesis. Together, we pushed through the most intense weeks before the NeurIPS deadline, supporting each other and completing the work under significant pressure. I will not forget those three weeks of writing, experiments, debugging, and revisions. I am equally thankful to Haodong Zheng, who worked with us on the paper and contributed to bringing the project to completion.

I would also like to thank my girlfriend, Yang Li, for her constant care and support. During the periods when I was anxious, overwhelmed, or extremely busy, she was always there for me, listened to me, and helped me regain balance. Her quiet support in the background has been one of the most important sources of strength during this thesis.

Finally, I am deeply grateful to my family. Although we have been separated by a great distance while I studied abroad, their unwavering support has always been with me. Being able to study and work with peace of mind in another country would not have been possible without them.

Youliang Zhu, Gothenburg, May 2026

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AI	Artificial Intelligence
BM25	Best Matching 25
BLEU	Bilingual Evaluation Understudy
F1	Harmonic mean of precision and recall
FSDP	Fully Sharded Data Parallel
GPU	Graphics Processing Unit
GRPO	Group Relative Policy Optimization
JSON	JavaScript Object Notation
KL	Kullback–Leibler
LLM	Large Language Model
PPO	Proximal Policy Optimization
QA	Question Answering
RAG	Retrieval Augmented Generation
RL	Reinforcement Learning
SGLang	Structured Generation Language
VERL	Volcano Engine Reinforcement Learning
XML	Extensible Markup Language

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables used throughout this thesis.

Indices

i, j, k	Indices for sessions, chunks, turns, evidence items, or sampled trajectories
r, t	Indices for iterative retrieval rounds; t is also used in timestamp notation when explicitly subscripted
y	Index for an unmasked Synthesizer token in the GRPO objective

Sets

$\mathcal{M}$	Long-term conversational memory bank
$\mathcal{B}$	Indexed memory database used by the Retriever
$\mathcal{D}$	Dataset or distribution of queries used for training or evaluation
$\mathcal{S}_q$	Sessions selected for query q by a conventional retriever
$\widetilde{\mathcal{S}}_t$	Normalized sessions retrieved at round t
$\widetilde{\mathcal{S}}_{\leq t}$	Accumulated normalized sessions retrieved up to round t
$\mathcal{S}^{\text{ret}}_{\leq t}$	Session identifiers covered by accumulated retrieved sessions up to round t
$\mathcal{G}_q$	Gold evidence session set for query q
C_t	Retrieval cues generated by the Planner at round t
$C_{\leq t}$	Retrieval cues accumulated up to round t
$\mathcal{E}_t$	Evidence events emitted by the Synthesizer at round t
H_t	Temporal Evidence Pool after previous retrieval rounds

$\Gamma_i, \Gamma_q, \Gamma_{\leq t}$	Deterministic temporal hints for session i, query q, or retrieved context up to round t
$\mathcal{T}$	Group of sampled GRPO trajectories for one query
$\mathcal{Y}^i$	Unmasked Synthesizer tokens in trajectory τ^i
$\mathcal{D}_{\text{dist}}$	Distractor events in a TimeMemEval example

Parameters

N	Number of sessions in the memory bank
L_i	Number of conversational units in session S_i
K	Retrieval budget or number of sampled trajectories, depending on context
G	GRPO group size, i.e., the number of trajectories sampled for one query
T	Number of retrieval rounds in a trajectory
$T_{\max}$	Maximum number of retrieval rounds
B	Length budget used in the compactness reward
w_o, w_v, w_l	Reward weights for outcome correctness, verdict correctness, and compactness
c	Fixed penalty assigned to a format-invalid trajectory
ϵ	Small constant for numerical stability in advantage normalization
ϵ_c	GRPO clipping parameter
β	KL regularization coefficient in the GRPO objective; also used as a bridge-scope variable in the TimeMemEval appendix when locally defined
δ	Temporal offset used to construct a TimeMemEval bridge window
$[\alpha_{ij}, \beta_{ij}]$	Normalized time span for temporal phrase ϕ_{ij}

Variables

$q, \widetilde{q}$	User query and temporally normalized query
t_q	Query time anchoring relative expressions in the current query
$S_i, \widetilde{S}_i$	Historical session and temporally normalized session

c_{ij}	Conversational unit j in session S_i
t_i	Session timestamp of S_i
ϕ_{ij}	Temporal phrase extracted from session S_i
t_{session}	Time when a cited session was stored or observed
t_{event}	Time or interval when the described event happened or was valid
$e, h(e)$	Evidence event and its structured Temporal Evidence Pool record
$\text{text}(e)$	Textual content of evidence event e
$s(e)$	Source session identifier for evidence event e
$\hat{v}_t$	Binary Synthesizer sufficiency verdict at round t
r_t	Synthesizer reasoning text at round t
h_t	Retrieval context given to the Synthesizer at round t
z_t	Structured Synthesizer action at round t
Norm	Deterministic temporal normalization function
Index	Indexing function that builds the memory database
P	Frozen Planner that generates retrieval cues
Ret	Frozen Retriever that searches the indexed memory database
τ	Multi-round rollout trajectory
$f(\tau)$	Format-validity gate for trajectory τ
$\mathbf{v}$	Sequence of round-level verdict correctness signals
v_t	Verdict correctness signal at round t, valued in $\{-1, +1\}$ when used in the reward appendix
$o(\tau)$	Final answer correctness indicator
$\ell(\tau)$	Compactness score for a trajectory
$m(z_t)$	Token length of the Synthesizer output at round t
$V(\tau)$	Accumulated verdict score across a trajectory
$R(\tau)$	Total scalar reward assigned to trajectory τ
R_{ver}	Round-level verdict reward
R_{out}	Terminal outcome reward
$\hat{a}, a^\star$	Predicted final answer and gold answer
A, J	Frozen downstream answerer and frozen answer judge
π_θ	Trainable Synthesizer policy
$\pi_{\theta_{\text{old}}}$	Behavior policy used to sample GRPO trajectories

π_{ref}	Frozen reference policy used for KL regularization
$\hat{A}^i$	Group-normalized advantage for trajectory τ^i
$\rho_y(\theta)$	Per-token probability ratio in the GRPO objective
D_{KL}	Kullback–Leibler divergence term
$\mathbb{I}[\cdot]$	Indicator function
x	TimeMemEval example
e_s, e_d	Source event and destination event in TimeMemEval
W_b	Temporal bridge window in TimeMemEval
g	Function that applies a temporal relation or offset to produce W_b

Contents

List of Figures

List of Tables

1
Introduction

Large language models (LLMs) have become a foundation for systems that can produce text, follow instructions, use external tools, and participate in multi-turn interaction. When such models are embedded in agent systems, they are no longer used only as single-turn text generators. They may plan subtasks, call tools, inspect external resources, and adapt their behavior over a longer interaction. This shift makes state important. An agent that helps a user over days or weeks needs access to information that was introduced outside the current prompt.

Long-term memory is therefore a central component of LLM agents. A memory system can preserve user preferences, plans, corrections, decisions, and events that occurred across previous sessions. Without such a component, an agent is limited by the current context window and must either forget earlier interactions or ask the user to repeat information. Recent memory systems for LLM agents study how to store, organize, retrieve, and update long-term conversational information [4, 21, 35, 40, 41].

Figure 1.1 illustrates the basic motivation for such a memory component. A naive long-context approach appends the message history and the current query directly to the prompt. This can overflow the context window, increase token cost and latency, and still leave the model to find the relevant parts of the history by itself. A memory-augmented approach instead uses a memory system to select information that is relevant to the current query before the prompt is sent to the LLM. The goal is not only to shorten the prompt, but also to make the historical context more focused for downstream answering.

This thesis focuses on conversational agent memory. Unlike a static document collection, a conversation history is incremental, personal, and time-sensitive. The same user may mention a future plan in one session, revise it later, and ask about it after several unrelated conversations. The memory source therefore contains not only facts, but also event order, relative dates, stale information, and references to earlier or later moments. A useful memory system must recover evidence from this history even when the user does not remember the exact wording or the session in which the evidence appeared.

Temporal reasoning is especially important in this setting. A user may ask about what happened before a trip, which book they were reading during the week of a promotion, or what decision changed after a meeting. Such questions are not only asking for stored facts. They require the memory system to distinguish when something was mentioned from when it actually happened, and to use temporal information to decide which part of the history should be inspected. Prior work and benchmarks show that LLMs remain brittle on temporal reasoning tasks, including

event ordering, relative dates, and long conversational histories [5, 9, 22].

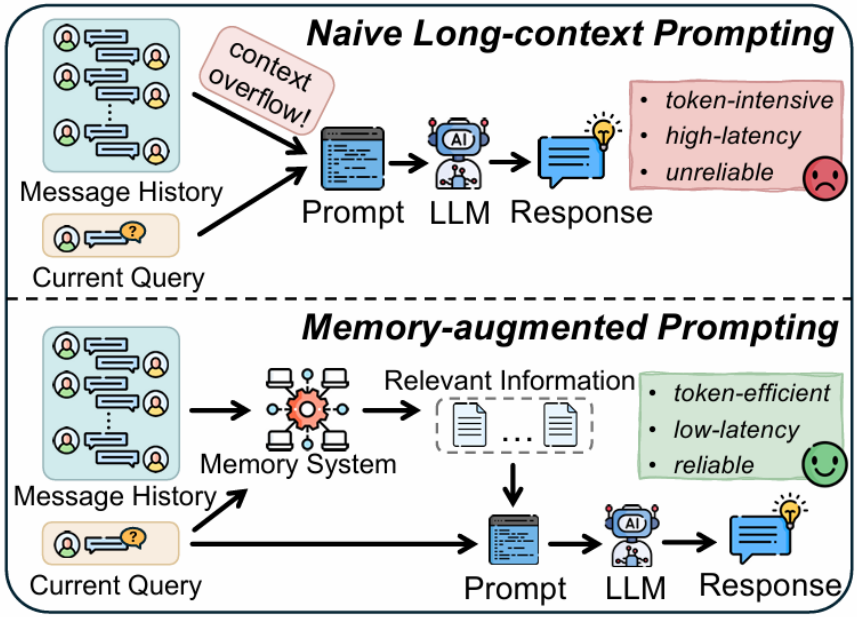

Figure 1.1: Naive long-context prompting compared with memory-augmented prompting. Long-context prompting sends the message history directly to the LLM, whereas memory-augmented prompting first uses a memory system to select relevant information for the current query.

The position of this thesis is that long-term agent memory should be studied as an evidence-providing component. The memory module should not simply return all retrieved sessions to a downstream answer model, because that makes the answer model perform noise filtering, temporal reasoning, evidence selection, and final answer generation at once. The memory module should also not hide the retrieval and synthesis process behind an opaque final answer. Instead, it should provide compact, source-grounded, temporally grounded evidence that a downstream answer model can inspect and use.

1.1 Significance of the Study

This study is significant because conversational memory is becoming a basic requirement for long-running AI assistants rather than a peripheral research idea. Modern conversational agents are expected to remember preferences, past decisions, plans, corrections, and personal context across sessions. This changes the role of the assistant: it is no longer only a model that answers the current prompt, but a system that must maintain continuity over a user's history. If the memory layer is unreliable, the agent may forget important context, retrieve stale information, or force the user to repeatedly restate facts that should already be available.
The technical significance is that long-term conversational memory exposes reasoning problems that are not solved by storing more text. Benchmarks such as LongMemEval and LoCoMo treat temporal reasoning, multi-session reasoning, knowledge

updates, and multi-hop reasoning as core memory abilities [34, 22]. These abilities are central to real conversations because users often refer to events by relative time, implicit dependencies, or remembered circumstances rather than by exact session identifiers. The problem is therefore not only recall. A memory system must decide which pieces of history count as evidence for the current question and how temporal relationships should guide later retrieval.

The system significance is that existing memory infrastructure leaves an important design space between raw-context prompting and heavy write-time memory construction. Systems such as MemGPT, Mem0, and Zep show that persistent memory, context management, and temporal knowledge organization are valuable for agents [24, 4, 25]. At the same time, write-time extraction, summarization, or graph construction must decide what is important before the future query is known. This thesis instead emphasizes query-time evidence construction: preserving the original conversational memory more lightly, then selecting, grounding, and synthesizing evidence when a user question arrives.

The practical significance is that compact and source-grounded evidence is a more inspectable memory interface than either passing all retrieved sessions to an answer model or returning only a final answer. Long personal and workplace histories can be noisy, sensitive, and expensive to process. A memory module that produces concise evidence with source grounding can reduce the burden on the downstream answer model and make failures easier to diagnose. In this sense, NanoMem is not presented as a complete solution to long-term agent memory, but as a study of an important interface: query-time, temporally grounded evidence construction for conversational agents.

1.2 Problem Description

The problem addressed in this thesis is that evidence needed for long-term conversational memory is often not directly addressable from the user's query. A query may contain a relative time expression, an event description, a causal consequence, or a remembered attribute, while the session that contains the answer can only be found after another piece of evidence has first been recovered. This thesis refers to this as the evidence addressability gap.

The first part of the problem is temporal reasoning. Even strong LLMs can make errors when reasoning about event order, relative dates, durations, and updates over time [5, 9]. In conversational memory, this difficulty is compounded by the difference between session time and event time. Session time records when the user said something. Event time records when the described event happened or was valid. If a user says on April 2 that they travelled to Europe last week, April 2 is the session time, but the travel event belongs to the previous week. A memory system that conflates these two times can retrieve or interpret the wrong evidence.

The second part is hidden-dependency retrieval. Many useful memory questions cannot be answered by one retrieval step. For example, in the question "What book was I reading during the week I got promoted?", the query does not directly identify the session where the book was mentioned. The system must first find evidence about the promotion, determine the relevant time window, and then use

that window to locate the reading-related evidence. The first retrieved evidence is not the answer; it supplies the address needed for the next retrieval step.
The third part is the amount and noise of retrieved memory. Long-term conversational memory can return many candidate sessions, and those sessions may contain unrelated events, outdated information, and near-miss evidence. Passing all of this raw context to a downstream LLM turns the task into a needle-in-a-haystack problem over a long and noisy input. This is expensive, can dilute attention over the relevant evidence, and still leaves the downstream model responsible for temporal reasoning and multi-hop evidence selection. Repeated multi-round retrieval can make this problem worse if each round keeps accumulating uncompressed sessions.
Together, these issues make long-term conversational memory more than a top-k retrieval problem. The system must find evidence whose retrieval address may not be present in the original query, and it must distinguish retrieved memory from evidence that is compact and reliable enough for downstream answering.

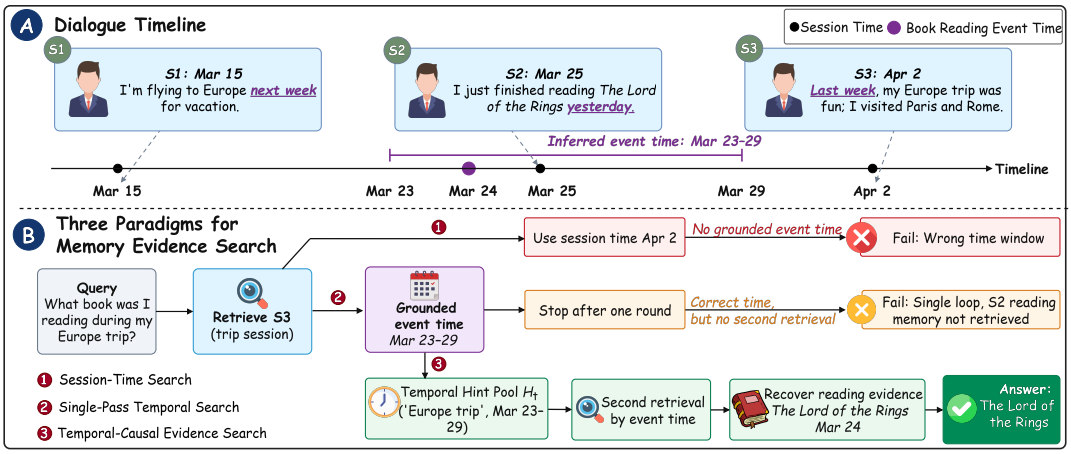

Figure 1.2: Evidence addressability gap in temporal conversational memory. The answer-bearing evidence may not be directly retrievable from the original query: an earlier piece of evidence must first expose a temporal bridge, while raw retrieved sessions still need to be reduced to usable evidence.

1.3 Purpose of the Study

The purpose of this study is to investigate how a long-term conversational agent memory system can construct compact and temporally grounded evidence for questions whose supporting information is distributed across multiple sessions and is not directly addressable from the original query.
The first objective is to study query-time memory evidence construction. This thesis does not treat memory primarily as a write-time process in which the system must fully interpret, summarize, or restructure a conversation before knowing the future query. Instead, the study focuses on how a memory agent can model the relationship between a user query and the stored history when the query arrives. The ingestion stage should preserve the original memory structure and provide only lightweight processing, indexing, and temporal support, so that future query-specific

interpretations are not fixed prematurely.
The second objective is to study complex temporal reasoning and multi-hop evidence search in long-term conversation. The target setting is not only general long-term recall, but temporal-causal evidence search: cases where the answer evidence is distributed across sessions and the cue needed for a later retrieval step must be discovered from earlier evidence.
The third objective is to study how the evidence construction component can be trained for this setting. NanoMem uses a Synthesizer to construct evidence from retrieved sessions and to decide whether the current evidence is sufficient for downstream answering. A central purpose of the study is to investigate whether reinforcement learning can align this component with the behavior required by the framework: useful temporal grounding, compact evidence synthesis, and sufficiency judgment.
These objectives lead to three research questions:
RQ1. How can a long-term conversational memory system construct evidence at query time while preserving the original memory structure and avoiding heavy query-agnostic interpretation during ingestion?
RQ2. How can a memory agent recover temporally grounded and multi-hop evidence when the retrieval cue needed for the answer is not directly present in the original query?
RQ3. How can reinforcement learning be used to train the Synthesizer to produce useful, compact, temporally grounded evidence and to judge whether the current evidence is sufficient for downstream answering?
NanoMem is developed in this thesis as a query-time memory evidence provider for long-term conversational agents. The detailed system design, training procedure, and evaluation are presented in the following chapters.

1.4 Contributions

This thesis makes the following contributions to the study of long-term conversational agent memory:

1. **Problem formulation.** A formal definition of query-time temporal evidence-state construction as the memory task, distinguishing session time, event time, and query time, and identifying the evidence addressability gap as the central difficulty.
2. **NanoMem system.** The NanoMem architecture, comprising lightweight temporal normalization at ingestion time, a Temporal Evidence Pool for source-grounded event records, and an iterative Planner–Retriever–Synthesizer loop controlled by a binary sufficiency verdict.
3. **Training procedure.** A GRPO-based training procedure for the Synthesizer that combines an outcome reward, a per-round verdict reward, a compactness reward, and a format gate, designed to align the Synthesizer with the evidence-state interface without training the broader retrieval environment.
4. **TimeMemEval benchmark.** A diagnostic benchmark for hidden-dependency temporal bridge search in long-term conversational memory, where the answer-bearing evidence becomes addressable only after an intermediate temporal

anchor has been established.

5. **Empirical evaluation.** A controlled comparison of NanoMem against full-context prompting, write-time memory systems, and search-time retrieval agents on LoCoMo, LongMemEval, and TimeMemEval, accompanied by architecture and reward ablations that isolate the contribution of each design choice.

1.5 Scope and Delimitations

This thesis investigates one component of the long-term memory problem for LLM agents: query-time evidence construction for conversational memory. Several related areas are outside the scope of this study and are important to make explicit.

Conversational memory, not document RAG. This work focuses on the memory of an agent interacting with a single user over multiple sessions. It does not cover document retrieval systems, static corpora, or knowledge bases. The session structure, personal context, and temporal sensitivity of conversational history create the specific problem that NanoMem addresses. Systems designed for document RAG face a different problem and are treated only as technical neighbors in Chapter 2.

Synthesizer training only. NanoMem trains only the Synthesizer component. The Planner, Retriever, downstream answerer, and judge are treated as frozen components of the retrieval environment. Training the Planner or Retriever jointly with the Synthesizer is a natural extension but is left for future work.

Three benchmark datasets. The evaluation is limited to LoCoMo, LongMemEval, and TimeMemEval. These benchmarks were chosen because they cover complementary question types relevant to the research questions. The study does not claim that the reported performance generalises to all conversational memory scenarios or question distributions.

English language only. All datasets and prompts used in this thesis are in English. Multilingual conversational memory, non-English time expressions, and cross-lingual personal memory are outside the evaluation scope.

No deployment infrastructure. This thesis does not address real-time latency, privacy-preserving storage, user consent management, or production-scale memory services. These aspects are discussed as directions for future work in Chapter 5.

1.6 Thesis Outline

Chapter 2 reviews the literature on LLM agents, long-term agent memory, retrieval-augmented generation, temporal reasoning, and reinforcement learning for agent memory. Chapter 3 formulates the memory evidence construction problem and presents the NanoMem architecture, including temporal evidence representation, the Planner–Retriever–Synthesizer loop, GRPO (Group Relative Policy Optimization) training, reward design, and the construction of TimeMemEval. Chapter 4 reports the experimental setup, main benchmark results, comparative analysis, architecture ablations, and reward ablations. Chapter 5 answers the research questions and discusses implications, limitations, future work, and ethical considerations.

2

Related Work

This chapter positions NanoMem within prior work on LLM agents, long-term agent memory, retrieval-augmented generation, temporal reasoning, and reinforcement learning for memory systems. The goal is not to present these areas as competing lines of work, but to clarify their system boundaries. NanoMem builds on retrieval and memory-augmented agent research, while focusing on a narrower problem: constructing compact, source-grounded, temporally grounded evidence at query time for long-term conversational memory.

2.1 LLM Agents

Large language model agents extend language models from single-turn text generators into systems that can act over multiple steps. In this setting, an LLM is used as a controller that may decompose tasks, call tools, inspect external observations, update its plan, and produce responses based on both the current instruction and accumulated state. Work such as ReAct and Toolformer helped establish the idea that language models can interleave reasoning with external actions or tool calls rather than only generating a final answer in one pass [38, 26].

Although agent systems vary in implementation, they commonly combine several functional components: planning, tool use, retrieval, memory, and feedback or reflection. These components are useful because real tasks often require more than the information present in the current prompt. An agent may need to retrieve a document, remember a user preference, revise a plan after an observation, or decide that more information is needed before answering.

Memory becomes central when the agent is expected to serve the same user, project, or workflow over a long period of time. A conversational agent cannot rely only on the current context window if the relevant information was mentioned days or months earlier. It needs a mechanism for storing and re-accessing preferences, facts, corrections, plans, events, and task state across sessions. The remainder of this chapter focuses on that memory problem, especially in long-term conversational settings.

2.2 Long-Term Agent Memory

Long-term agent memory addresses the mismatch between persistent user history and the limited active context available to an LLM. In a long-running interaction,

useful information is distributed across many sessions and may include user preferences, personal facts, project decisions, future plans, outdated statements, corrections, and events mentioned only once. Directly passing the entire history to the model is expensive, slow, and often unreliable. A memory system therefore turns past interactions into a resource that can be stored, maintained, retrieved, and reused when a future query requires it.

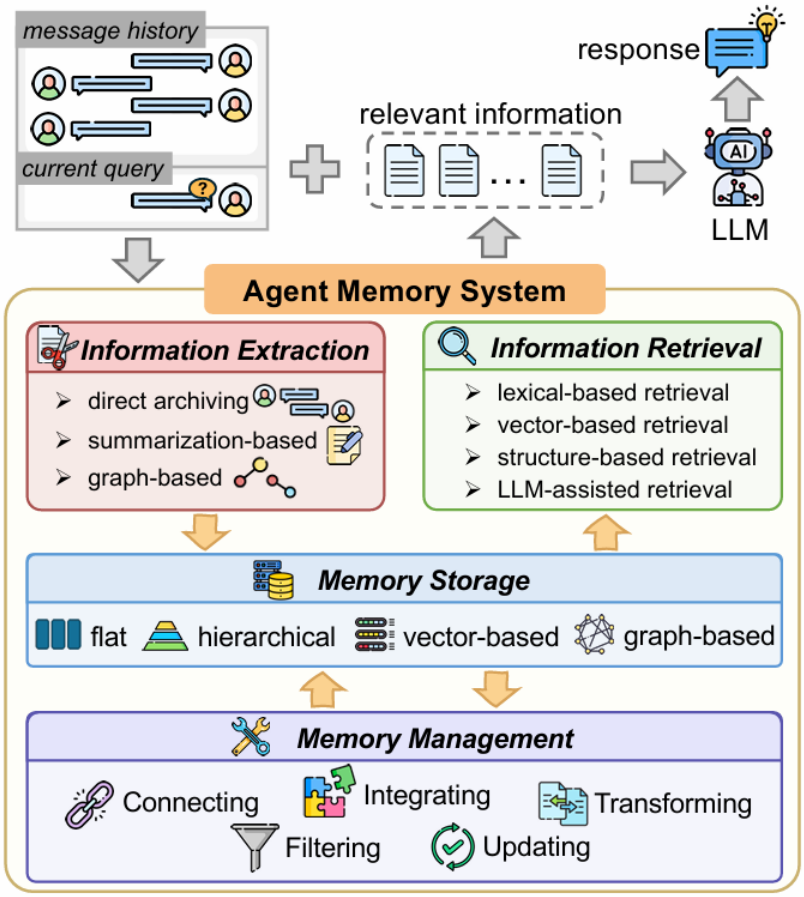

Figure 2.1: A generic long-term agent memory framework. Historical messages and the current query are processed by an agent memory system that extracts, stores, manages, and retrieves information before passing relevant memory content to the LLM.

Figure 2.1 summarizes a common view of long-term agent memory. On the write side, information extraction decides how interaction history enters memory. Some systems directly archive conversations, others summarize them into compact records, and graph-based systems extract entities, relations, or events. Memory storage then determines how the resulting memory units are represented and indexed, for example as flat records, hierarchical structures, vector stores, or graph stores. Memory management maintains the memory over time by connecting, integrating, transforming, filtering, and updating records. On the query side, information retrieval selects relevant memory content through lexical, vector-based, structure-based, or

LLM-assisted methods.
This framework is useful because it separates two design choices that are often blurred. A memory system may do heavy write-time processing, attempting to summarize, normalize, and structure information before a future query is known. It may also do query-time processing, where the current user question determines which parts of memory should be retrieved and how they should be interpreted. NanoMem does not reject write-time processing. It uses lightweight extraction, temporal enhancement, and indexing so that memories remain searchable. However, it avoids making the memory writer responsible for heavy interpretation of future query-dependent relations. The main reasoning step is moved to query time, where evidence can be constructed with the actual question in view.
Prior long-term memory systems cover several parts of this framework. MemGPT frames memory as a virtual context management problem, where an agent decides which information should stay in active context and which should be moved to external storage [24]. Production-oriented systems such as Mem0 and MemOS organize histories into persistent memory units and expose memory as a reusable service for agents [4, 21]. Structured memory systems enrich the memory substrate itself: Zep uses a temporal knowledge graph, A-Mem constructs dynamic agentic notes, GMemory studies hierarchical memory for multi-agent systems, HyperMem uses hypergraph memory for long conversations, and MemPalace represents a local-first memory management direction [25, 35, 40, 39, 23]. Compression-oriented systems such as LightMem and R3MEM reduce the cost of retaining long histories [8, 31].
These systems show that agent memory is not simply a vector database attached to a chatbot. It is a long-lived subsystem with extraction, storage, management, and retrieval responsibilities. This thesis focuses on a narrower case within that space: conversational agent memory. Conversation history has a session structure, contains updates and corrections, and frequently includes relative time expressions. The time at which a user mentions an event can differ from the time at which the event happened. As a result, long-term conversational memory requires more than storing more history; it requires query-time interpretation of how past memories relate to the current question.

2.3 Retrieval Augmented Generation

Retrieval Augmented Generation (RAG) is the standard paradigm for combining a language model with external information. Given a query, a retriever selects relevant documents or passages from an external corpus, and a generator uses the retrieved context to produce an answer. Classic work such as REALM, DPR, RAG, FiD, and Atlas established important variants of this retrieve-then-read pipeline, including learned retrieval, dense passage retrieval, sequence generation conditioned on retrieved documents, fusion over multiple passages, and retrieval-augmented few-shot learning [12, 19, 20, 14, 15].
Recent RAG research has moved beyond one-shot retrieval. Self-RAG trains a model to decide when to retrieve and critique its own generations, while SmartRAG jointly learns RAG-related behaviors from environmental feedback [2, 10]. Search-R1 and RAG-RL study reinforcement learning for search and retrieval behavior [17, 13].

Other work optimizes retrieval preferences, faithfulness, and structured search-agent behavior [36, 30, 16, 43]. These directions are important for NanoMem because they show that query-time retrieval can be adaptive rather than a fixed top-k operation. However, RAG and long-term conversational agent memory are not the same task. RAG is usually defined over an external corpus such as documents, webpages, papers, or search results. The corpus is often treated as already existing, and the main problem is to retrieve useful context for the current information need. Conversational agent memory is produced by the interaction between a user and an agent. It is personal, multi-session, updateable, and part of the agent's state. The goal is not only to answer a knowledge question, but to preserve continuity, personalization, and consistency across time.

This distinction becomes sharper for temporal questions. A RAG document may have metadata timestamps, but standard RAG does not normally separate the time at which a memory was mentioned from the time at which the event described by that memory occurred. In long-term dialogue, that separation is often essential. As illustrated in Section 1.2, the time at which a user mentions an event in a session frequently differs from the time at which the event actually occurred. A later question may depend on the event's occurrence date rather than the session in which it was mentioned. Treating the conversation history as an ordinary document corpus leaves this temporal relation implicit.

NanoMem therefore uses RAG as a technical neighbor, not as the task definition. It inherits the query-time retrieval idea and is influenced by reflective RAG and search-agent RL. At the same time, it also has a write-side memory component: memories are lightly extracted, timestamped, temporally enhanced, and indexed without aggressively rewriting the original conversational structure. The central contribution is then placed on the search side. At query time, NanoMem synthesizes compact, source-grounded evidence with explicit temporal fields and a binary sufficiency verdict, rather than returning only raw retrieved passages or a final answer.

2.4 Temporal Reasoning in Agent Memory

Temporal reasoning is a core difficulty in long-term agent memory. Dialogue contains absolute dates, relative expressions, event order, durations, plans, updates, and implicit dependencies across sessions. Users rarely formulate queries as exact session lookups. They ask about "the week after my promotion", "the trip I mentioned last time", or "the meeting before I changed my plan". Such questions require the system to identify a temporal anchor, normalize or infer the relevant time window, and often use that window to retrieve additional evidence.

Benchmarks support this problem framing. TimeBench and Test of Time evaluate general temporal reasoning abilities and show that LLMs remain brittle when handling temporal relations and event ordering [5, 9]. LoCoMo evaluates very long-term conversational memory and includes temporal and multi-hop question types [22]. LongMemEval benchmarks chat assistants on long-term interactive memory, including temporal reasoning, multi-session reasoning, knowledge updates, and abstention [34]. These benchmarks make temporal reasoning a central capability for memory systems rather than an optional add-on.

Existing memory systems incorporate time in several ways. Zep encodes temporal information into a temporal knowledge graph, making time part of the structured memory substrate [25]. TReMu builds time-aware memory with timeline summaries and combines them with neuro-symbolic reasoning for multi-session temporal questions [11]. Beyond Dialogue Time argues for temporal semantic memory that goes beyond the dialogue timestamp alone [29]. Memory-T1 uses reinforcement learning to improve temporal reasoning in multi-session agents, especially through temporally relevant session selection and reasoning [7].
NanoMem is aligned with these works in treating time as essential, but it uses time differently. Rather than only storing time as metadata, summarizing it into a timeline, or using it as a selection signal, NanoMem treats time as part of the query-time search state. Each synthesized evidence event separates the source session time from the inferred event time. The inferred event time can then become a new retrieval cue for the next round. For example, a first round may find when a user was promoted; the resulting time window can then be used to search for what the user was reading during that week. This turns temporal grounding from a passive attribute into an active bridge for multi-hop memory search.

2.5 Reinforcement Learning for Agent Memory

Reinforcement learning is relevant to agent memory because many memory decisions are policy decisions rather than simple next-token prediction problems. A system may need to learn what to write, what to update, what to retrieve, whether current evidence is sufficient, whether another search round is needed, and how compact the intermediate evidence should be. These decisions are naturally evaluated by downstream usefulness, temporal grounding, faithfulness, and stop-or-continue behavior.
One line of work applies RL or learnable policies to write-side memory construction and management. Memory-R1 trains agents to manage and utilize memories through reinforcement learning, while MemReader and Mem-α study active or learnable memory construction during the ingestion stage [37, 18, 32]. These systems show that memory operations do not need to be purely hand-written heuristics. Their focus, however, is primarily on memory construction, extraction, or lifecycle management before or during storage.
A second line trains search-side behavior for RAG and retrieval agents. Search-R1, RAG-RL, RPO, SSFO, TreeGRPO, and Stratified GRPO optimize retrieval, query rewriting, tool use, faithfulness, or multi-step search behavior [17, 13, 36, 30, 16, 43]. MMOA-RAG and EMG-RAG also connect retrieval-augmented generation with multi-agent RL or editable memory graphs [3, 33]. These works are relevant because they demonstrate that search behavior can be trained, but their rewards are often attached to final answer quality, retrieval success, or generic faithfulness signals rather than to structured memory evidence.
Memory-T1 is closest to NanoMem in that it combines reinforcement learning with multi-session temporal memory [7]. The difference is the trained object and the intermediate output. Memory-T1 primarily improves temporal session selection and answer behavior. NanoMem trains the Synthesizer as an evidence construction

policy. Its output is not merely an answer or a selected session, but a set of structured evidence events together with a binary verdict, `sufficient` or `insufficient`. This verdict is important because hidden-dependency temporal questions often require an intermediate bridge: the first evidence may be useful because it reveals the next retrieval cue, even though it is not yet enough to answer the user query.
This framing motivates the method in the next chapter. NanoMem uses reinforcement learning to optimize intermediate evidence synthesis, with signals for parseable format, compactness, downstream usefulness, temporal grounding, and sufficiency behavior. The resulting policy is trained to decide not only what evidence to state, but also whether the current evidence should stop the search or guide another retrieval round.

3

Methods

This chapter describes NanoMem as a query-time memory evidence provider. The central claim is that long-term conversational memory should not only retrieve raw sessions, but should construct a compact, source-grounded, and temporally grounded evidence state for a downstream answer model. The chapter opens with the research approach, then formalizes the evidence-state construction problem, and presents the NanoMem architecture, the Temporal Evidence Pool, temporal normalization, the Planner–Retriever–Synthesizer loop, GRPO training data, GRPO training, the reward design, and the construction of TimeMemEval. Experimental settings, baselines, data splits, and results are deferred to Chapter 4.

3.1 Research Approach

This thesis takes a design science research approach: the primary artefact is NanoMem, a memory system that is designed, implemented, and evaluated against empirical benchmarks. The study does not derive the system from a single existing theory. Instead, it identifies a gap in long-term conversational memory, proposes a design motivated by that gap, implements the design, and evaluates it with controlled baselines and ablations that isolate each design choice.

The evaluation proceeds in three phases. First, the evidence-state construction problem is formulated, making explicit the gap between raw session retrieval and the temporally grounded evidence that a downstream answerer needs. Second, the NanoMem architecture, training procedure, and reward design are specified and implemented based on this formulation. Third, the system is evaluated on three benchmarks using a fixed judge and standardised baselines.

This approach is appropriate for the research questions defined in Chapter 1. RQ1 and RQ2 concern design properties that can only be assessed empirically, since there is no established ground truth for what constitutes a sufficient evidence state in long-term conversational memory. RQ3 concerns whether a specific training procedure aligns an LLM policy with a defined behavioural interface, which is similarly answered through measurement rather than analytical proof. Benchmark experiments and ablations therefore form the primary evidence base throughout this thesis.

3.2 Problem Formulation

Let the long-term conversational memory bank be

$$\mathcal{M} = \{(S_i, t_i)\}_{i=1}^{N},$$

where S_i is the i-th historical session and t_i is its session timestamp. Each session is an ordered sequence of conversational units,

$$S_i = (c_{i1}, c_{i2}, \ldots, c_{iL_i}),$$

where a unit may be a message, turn, or chunk. Given a user query q issued at query time t_q, a conventional memory retriever selects a subset of sessions

$$\mathcal{S}_q \subseteq \mathcal{M}$$

and passes those raw sessions to a downstream model. This objective is too weak for temporal and hidden-dependency questions. The answer-bearing session may not be directly addressable from the surface query; the system may first need to recover a bridge event, infer its event time, and only then search for the final evidence.
NanoMem therefore reformulates memory access as temporal evidence-state construction. For a query q, the system incrementally builds a Temporal Evidence Pool

$$H_T = \{h_1, h_2, \ldots, h_m\}$$

such that a frozen downstream answerer A can produce an answer from the query and the evidence pool:

$$\hat{a} = A(q, H_T).$$

NanoMem itself does not directly output $\hat{a}$. Its output is the structured evidence state H_T and a binary sufficiency verdict indicating whether the memory module has gathered enough evidence for answering.
This formulation distinguishes three time variables. The *session time* t_{session} records when a session was stored or observed. The *event time* t_{event} records when the described event actually happened or was valid. The *query time* t_q anchors relative expressions in the current query. These are not interchangeable: a user can say on April 2 that a trip happened "last week," so the session time is April 2 while the event time is the preceding week. NanoMem's method is designed around this separation.

3.3 NanoMem Overview

NanoMem has two stages. The first stage is lightweight memory preparation. It normalizes relative time expressions in historical sessions, preserves the original dialogue structure, and indexes the resulting session or chunk text in the memory database. This is intentionally not a heavy write-time interpretation stage: the system does not try to precompute every future fact or relationship that a later query might need.

The second stage is query-time evidence construction. A query is normalized with respect to t_q, a Planner generates retrieval cues, a Retriever returns candidate sessions, and a trainable Synthesizer converts retrieved material into structured events. The loop continues until the Synthesizer emits `sufficient` or a terminal condition is reached. The final pool is then given to a downstream answerer.

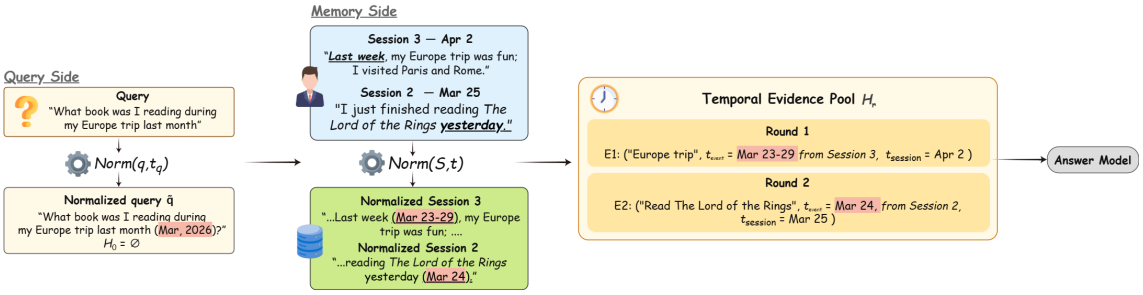

Figure 3.1: Temporal evidence-state construction from query and memory sides. Query-side and memory-side normalization resolve relative time expressions against different anchors, while the Temporal Evidence Pool stores source-grounded events that distinguish event time from session time before passing compact evidence to the answer model.

Figure 3.1 illustrates why the architecture is built around a structured evidence state. A session timestamp records when a memory was observed, while an event timestamp records when the described event actually happened. In the example, the Europe trip is mentioned on April 2 but is normalized to March 23–29; the reading event is mentioned on March 25 but is normalized to March 24. NanoMem keeps these distinctions explicit in the Temporal Evidence Pool, rather than forcing the downstream answerer to recover them from raw dialogue text.
The component boundary is important. The Planner generates search cues but does not answer. The Retriever returns source material and is treated as a frozen environment component. The Synthesizer constructs events, reasons about event time, and judges sufficiency. The answerer consumes the final evidence pool but is outside the NanoMem memory module. The judge is used only in training or evaluation to score final answers. A full architecture overview is provided in Appendix A.1; the main text separates the evidence-state construction and multi-round search views for readability.

3.4 Temporal Evidence Representation and Normalization

3.4.1 Temporal Evidence Pool

The Temporal Evidence Pool H_t is the policy state accumulated across retrieval rounds. Each evidence item is represented as

$$h(e) = \Big(\text{text}(e), t_{\text{event}}(e), t_{\text{session}}(e), s(e)\Big),$$

where text(e) is the evidence statement, $t_{\text{event}}(e)$ is the time or interval when the event happened, $t_{\text{session}}(e)$ is the timestamp of the cited session, and $s(e)$ is the source session identifier. The implementation exposes these fields through event text and the attributes `session_id`, `session_time`, and `event_time`. The sufficiency verdict is binary:

$$\hat{v}_t \in \{\texttt{sufficient}, \texttt{insufficient}\}.$$

After the Synthesizer emits a set of newly constructed events $\mathcal{E}_t$, the pool is updated as

$$H_{t+1} = H_t \cup \{h(e) \mid e \in \mathcal{E}_t\}.$$

In the prompt interface, the `events` block may be rendered cumulatively: the Synthesizer keeps still-valid previous events and appends newly extracted events. Conceptually, however, the state update is the pool update in subsection 3.4.1. The system does not restrict pool entries to cases where $t_{\text{event}} \neq t_{\text{session}}$. Events whose event time and session time coincide can still contain entities, locations, activities, or state changes that are useful for the next retrieval round.
Each event is source grounded. A record cites one source session rather than mixing several sessions into one evidence claim. If an answer requires several pieces of evidence, the pool should contain several records. This design keeps cross-session reasoning visible and makes failures easier to diagnose: errors can arise from retrieval, event selection, event-time inference, source attribution, or final answering.

3.4.2 Temporal Normalization and Hints

NanoMem uses deterministic temporal normalization to handle relative time expressions that can be resolved without learned reasoning. For a session and a query, normalization is written as

$$\widetilde{S}_i = \text{Norm}(S_i, t_i), \qquad \widetilde{q} = \text{Norm}(q, t_q).$$

The function Norm preserves the original phrase and inserts an absolute date or interval when possible. For example, a session phrase such as “last week” is resolved against the session timestamp t_i, while the same phrase in the query is resolved against t_q. The normalized sessions are indexed in the memory database $\mathcal{B}$:

$$\mathcal{B} = \text{Index}\Big(\{\widetilde{S}_i\}_{i=1}^{N}\Big).$$

It is useful to separate normalized text from temporal hints. Let

$$\Gamma_i = \{(\phi_{ij}, [\alpha_{ij}, \beta_{ij}])\}_j$$

denote the deterministic hints extracted from session S_i, where ϕ_{ij} is a temporal phrase and $[\alpha_{ij}, \beta_{ij}]$ is its resolved span. The query-side hints Γ_q are defined analogously from (q, t_q). These hints are parser aids for the Synthesizer; they are not the Temporal Evidence Pool. The pool comes from Synthesizer output, while temporal hints come from deterministic preprocessing.

Table 3.1: Core fields in temporal evidence construction.

Field or object	Meaning	Source
Event text	Concise statement of a query-relevant event or fact	Synthesizer output
`session_id`	Identifier of the single source session supporting the event	Retriever/session metadata
`session_time`	Timestamp of the cited source session	Session metadata
`event_time`	Time or interval when the described event happened or was valid	Temporal normalization and Synthesizer reasoning
`verdict`	Binary sufficiency decision: `sufficient` or `insufficient`	Synthesizer output
Temporal hints	Phrase-level parser aids such as a relative expression and its resolved span	Deterministic normalization
Temporal Evidence Pool H_t	Cross-round set of source-grounded evidence records	Accumulated Synthesizer events

This design turns part of temporal reasoning into lexical and semantic matching over explicit time spans. If a session says "next week" and the query says "two weeks ago," both expressions may normalize to the same absolute interval. BM25 and dense retrieval can then exploit a cue that was absent from the original surface text. At the same time, NanoMem avoids treating normalization as a complete temporal reasoner: complex event interpretation and sufficiency judgment remain the Synthesizer's responsibility.

3.5 Planner–Retriever–Synthesizer Loop

At round t, the Planner produces a set of retrieval cues C_t. In the first round it uses the normalized query; in later rounds it also receives previous cues, the previous Synthesizer reasoning, and the current pool:

$$C_t = \begin{cases} P(\widetilde{q}), & t = 0, \\ P(\widetilde{q}, C_{<t}, r_{t-1}, H_t), & t > 0. \end{cases}$$

The inclusion of H_t is deliberate. The Planner needs to know not only what is missing, but also what has already been established. An anchor event, event time, location, person, or state in the pool can become the address for the next retrieval query.

The Retriever searches the indexed memory database for the generated cues:

$$\widetilde{\mathcal{S}}_t = \mathrm{Ret}(C_t, \widetilde{q}; \mathcal{B}), \qquad \widetilde{\mathcal{S}}_{\le t} = \bigcup_{k=0}^{t} \widetilde{\mathcal{S}}_k.$$

The method is compatible with lexical, dense, or hybrid retrieval; concrete top-k, reranking, and model settings are reported in Chapter 4 when they are needed to interpret experiments.

The Synthesizer policy π_θ consumes the normalized query, cumulative retrieved sessions, the current evidence pool, and the deterministic temporal hints rendered in the prompt:

$$(r_t, \mathcal{E}_t, \hat{v}_t) \sim \pi_\theta\Big(\cdot \mid \widetilde{q}, \widetilde{\mathcal{S}}_{\le t}, H_t, \Gamma_q, \Gamma_{\le t}\Big).$$

When temporal hints are not written explicitly, they should be understood as part of the normalized query and normalized session representation. The output contains reasoning r_t, events $\mathcal{E}_t$, and the verdict $\hat{v}_t$. If $\hat{v}_t = \texttt{sufficient}$, the system stops and passes H_{t+1} to the answerer. If $\hat{v}_t = \texttt{insufficient}$, the reasoning and updated pool condition the next Planner call. Other terminal reasons, such as malformed XML, no new evidence, context budget exhaustion, or maximum rounds, are recorded as trajectory outcomes rather than discarded.

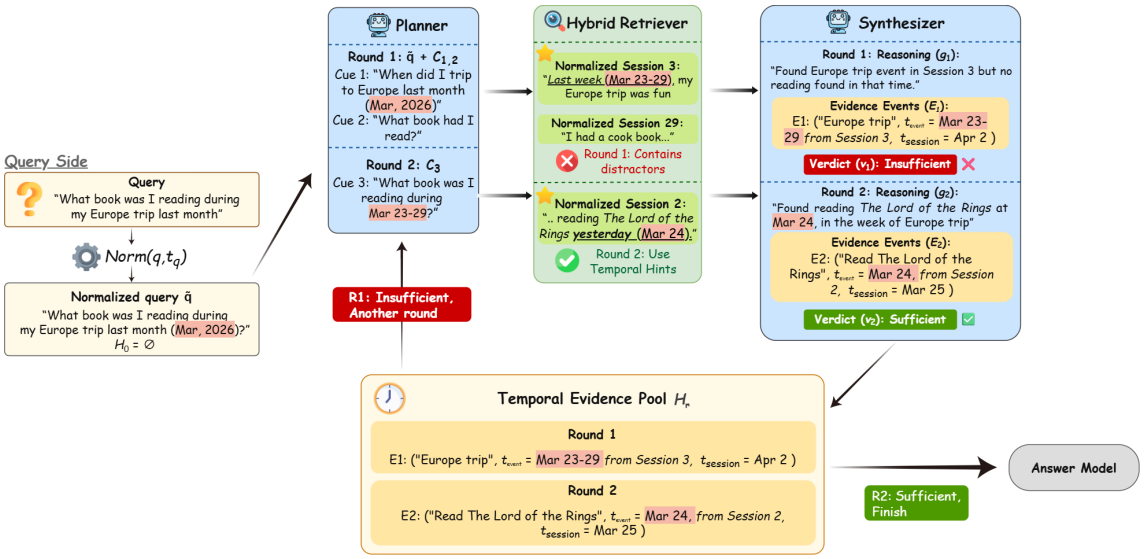

Figure 3.2: Planner–Retriever–Synthesizer loop for hidden-dependency temporal search. The first round recovers a temporal anchor but remains insufficient; the second round uses the accumulated evidence state as a retrieval hint, finds the answer-bearing memory, and terminates with a sufficient verdict.

Figure 3.2 shows how the evidence pool is used as a cross-round state rather than as a final-only summary. The first retrieval round can establish an anchor event without answering the query. Once that event is stored in H_t, the Planner can generate a more specific cue for the next round, such as searching within the inferred time window. The Synthesizer's binary verdict controls this process: `insufficient` keeps the loop active, while `sufficient` stops retrieval and exposes the accumulated pool to the answerer.

3.6 GRPO Training Data

GRPO training requires examples that define both a question and the searchable memory environment in which the policy must act. Each training instance therefore contains a query, an allowed search scope, and gold supervision for the final answer and evidence. The search scope specifies the user-side memory database or session set that the Retriever can access during rollout. The gold label includes the answer and the evidence sessions or turns used to evaluate retrieval coverage, verdict correctness, and final answer correctness.

Figure 3.3: Example structure of one GRPO training instance. Each sample contains the user query, the allowed search scope over memory sessions, and gold supervision for the answer and evidence used by the reward function.

Figure 3.3 illustrates the training-data interface. The policy is not trained on a fixed retrieved context; instead, each rollout calls the memory-search environment, retrieves sessions, constructs events, and receives rewards from the resulting trajectory. This structure is important because the task is not merely to answer from a supplied context, but to learn how to search, synthesize evidence, and decide when the evidence is sufficient. Detailed data splits and filtering rules are reported in Appendix A.2.

3.7 GRPO Training of the Synthesizer

NanoMem trains only the Synthesizer/Compressor policy. The Planner, temporal preprocessor, Retriever, answerer, judge, and reference policy are frozen. This choice keeps credit assignment focused: a failed final answer is treated as a signal about the Synthesizer's event construction and sufficiency behavior, rather than being confounded with simultaneous Planner or Retriever updates.
Let

$$h_t = (\widetilde{q}, \widetilde{\mathcal{S}}_{\le t}, C_{\le t}, H_t)$$

be the retrieval context at round t. The trainable action is the Synthesizer's structured XML output,

$$z_t = (r_t, \mathcal{E}_t, \hat{v}_t) \sim \pi_\theta(\cdot \mid h_t).$$

A rollout trajectory is

$$\tau = \{(h_1, z_1), (h_2, z_2), \ldots, (h_T, z_T)\}.$$

Only the tokens generated by the Synthesizer are optimized. Planner outputs, retrieved sessions, prompts, and other environment text are masked out from the policy loss.
For each query, GRPO samples a group of G trajectories

$$\mathcal{T} = \{\tau^i\}_{i=1}^{G} \sim \pi_{\theta_{\text{old}}}(\cdot \mid q),$$

assigns a scalar reward $R(\tau^i)$ to each trajectory, and computes the group-normalized advantage

$$\hat{A}^i = \frac{R(\tau^i) - \text{mean}_{j=1}^{G} R(\tau^j)}{\text{std}_{j=1}^{G} R(\tau^j) + \epsilon}.$$

The clipped GRPO objective over unmasked Synthesizer tokens is

$$J_{\text{GRPO}}(\theta) = \mathbb{E}_{q\sim\mathcal{D},\, \mathcal{T}\sim\pi_{\theta_{\text{old}}}(\cdot|q)} \Bigg[\frac{1}{G} \sum_{i=1}^{G} \frac{1}{|\mathcal{Y}^i|} \sum_{y\in\mathcal{Y}^i} \Big[\min\left(\rho_y(\theta)\hat{A}^i, \text{clip}(\rho_y(\theta), 1-\epsilon_c, 1+\epsilon_c)\hat{A}^i\right) - \beta D_{\text{KL}}[\pi_\theta(\cdot \mid \text{ctx}_y) \,\|\, \pi_{\text{ref}}(\cdot \mid \text{ctx}_y)] \Big] \Bigg],$$

where $\mathcal{Y}^i$ is the set of unmasked Synthesizer tokens in trajectory τ^i and

$$\rho_y(\theta) = \frac{\pi_\theta(y \mid \text{ctx}_y)}{\pi_{\theta_{\text{old}}}(y \mid \text{ctx}_y)}.$$

This objective trains the Synthesizer inside an online memory-search environment. Non-ideal trajectories are intentionally retained. A rollout that ends because of no new evidence, invalid XML, maximum rounds, or an `insufficient` terminal state still enters reward computation. Filtering such trajectories away would teach the model only what successful retrieval looks like, not how to behave when evidence is missing, noisy, or malformed.

The high-level GRPO computation is shown in Figure 3.4. GRPO was introduced in the DeepSeekMath line of work as a group-relative alternative to value-model-based policy optimization [27]. In NanoMem, the policy model is the Synthesizer inside the Planner–Retriever–Synthesizer loop. For the same input q, the policy samples a group of complete evidence-construction trajectories, the reward function scores each trajectory, group computation normalizes these scores into advantages, and a KL term regularizes the policy against a frozen reference model. The figure follows common RL terminology and labels the scoring block as a "Reward Model"; in this thesis, that block corresponds to the reward function defined in section 3.8, not to a separately trained reward-model network.

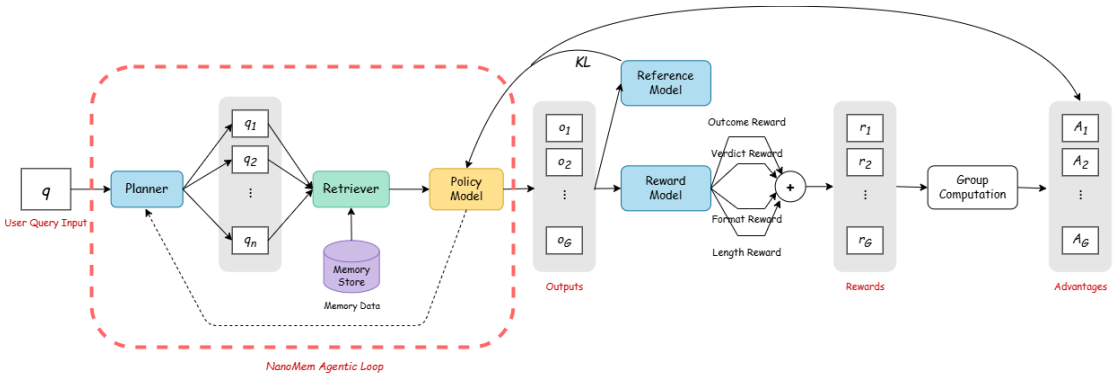

Figure 3.4: GRPO training for NanoMem. The trainable policy is the Synthesizer in the NanoMem agentic loop; the reward function combines outcome, verdict, format, and length rewards, after which group computation produces advantages for policy optimization under KL regularization.

3.8 Reward Design

The final reward used in this thesis contains a trajectory format gate, a round-level verdict reward, a terminal outcome reward, and a compactness term. Let a trajectory be abstracted as

$$\tau = (f, T, \mathbf{v}, o, \ell),$$

where $f \in \{0, 1\}$ is format validity, T is the number of retrieval rounds, $\mathbf{v} = (v_1, \ldots, v_T)$ is the sequence of verdict correctness signals, $o \in \{0, 1\}$ is final answer correctness, and $\ell \in [0, 1]$ is compactness.

3.8.1 Format Gate

Format validity is a gate rather than an additive per-round score. Let

$$f(\tau) = \prod_{t=1}^{T} \mathbb{I}[z_t \text{ is parseable and satisfies the XML schema}] .$$

The schema requires a reasoning field, an events block, a binary verdict, and for each event a source session id, session time, event time, and non-empty event text.

If $f(\tau) = 0$, the trajectory receives the fixed penalty $c < 0$ and the remaining reward terms are not evaluated.

3.8.2 Verdict Reward

Verdict reward supervises the stop-or-continue decision using the objective retrieval state, not the sessions selected by the Synthesizer. Let $\mathcal{G}_q$ be the gold evidence session set for query q, and let $\mathcal{S}^{\text{ret}}_{\leq t}$ be the sessions retrieved up to round t. The round-level verdict reward is

$$R_{\text{ver}}(\hat{v}_t; \mathcal{S}^{\text{ret}}_{\leq t}, \mathcal{G}_q) = \begin{cases} +1, & \mathcal{G}_q \subseteq \mathcal{S}^{\text{ret}}_{\leq t} \text{ and } \hat{v}_t = \texttt{sufficient}, \\ +1, & \mathcal{G}_q \not\subseteq \mathcal{S}^{\text{ret}}_{\leq t} \text{ and } \hat{v}_t = \texttt{insufficient}, \\ -1, & \text{otherwise.} \end{cases}$$

The trajectory-level verdict score is accumulated:

$$V(\tau) = \sum_{t=1}^{T} R_{\text{ver}}(\hat{v}_t; \mathcal{S}^{\text{ret}}_{\leq t}, \mathcal{G}_q).$$

Using retrieved sessions rather than Synthesizer-cited sessions prevents the policy from ignoring already retrieved gold evidence merely to justify another round.

3.8.3 Outcome Reward

Outcome reward is computed once after the trajectory terminates. The frozen answerer consumes the final accumulated Temporal Evidence Pool and returns

$$\hat{a} = A(q, H_{T+1}).$$

A frozen judge compares $\hat{a}$ with the gold answer $a^\star$:

$$R_{\text{out}}(\tau) = o(\tau) = \mathbb{I}[J(\hat{a}, a^\star) = 1] \,.$$

This term measures whether the constructed evidence is useful for answering; it does not make NanoMem itself an answer generator.

3.8.4 Length and Compactness Reward

Compactness is rewarded only among correct trajectories. Let $m(z_t)$ be the token length of the Synthesizer output at round t, and let B be a length budget. A compactness score can be written as

$$\ell(\tau) = o(\tau) \cdot \left[1 - \min\left(1, \frac{1}{BT} \sum_{t=1}^{T} m(z_t)\right)\right].$$

The factor $o(\tau)$ is important: without it, the model could learn to produce short but useless evidence. With this gate, compactness means "shorter among the trajectories that already answer correctly."

3.8.5 Total Reward

The total reward is

$$R(\tau) = \begin{cases} c, & f(\tau) = 0, \\ w_o o(\tau) + w_v \sum_{t=1}^{T} R_{\text{ver}}(\hat{v}_t; \mathcal{S}_{\le t}^{\text{ret}}, \mathcal{G}_q) + w_l \ell(\tau), & f(\tau) = 1. \end{cases}$$

The final coefficient setting is

$$(w_o, w_v, w_l) = (0.50, 0.10, 0.05), \qquad c = -0.20.$$

The hierarchy is intentional: answer correctness is the dominant task-level signal, verdict correctness provides process-level supervision, and length optimizes compactness only after usefulness is achieved. NanoMem does not use a separate event-time reward. If the Synthesizer infers the wrong event time, the error can mislead later retrieval and reduce final answer correctness, which is captured through the outcome reward.

For clarity, the coefficient logic can be summarized by the margin conditions used in the final reward analysis. With maximum round count $T_{\max} = 3$, the worst format-valid wrong-answer trajectory has reward $-3w_v$, and the best wrong-answer trajectory has reward $3w_v$. Outcome dominance requires

$$-3w_v + w_o > 3w_v \quad \Longrightarrow \quad w_o > 6w_v.$$

A single verdict correction changes the cumulative verdict score by 2, so its reward change is

$$\Delta_{\text{ver}} = 2w_v.$$

To keep verdict quality more important than the full compactness range, the design requires

$$2w_v > w_l.$$

Finally, the format-failure penalty is placed below the worst valid trajectory:

$$c < 3w_v - w_o.$$

The chosen values satisfy these inequalities and keep the reward aligned with the intended preference ordering.

3.9 TimeMemEval Construction

LoCoMo and LongMemEval evaluate long-term memory and multi-session reasoning [22, 34], but many questions can still be solved by one high-recall retrieval step. TimeMemEval is designed as a diagnostic stress test for hidden-dependency temporal search. Its key invariant is that the answer-bearing evidence becomes addressable only after an intermediate temporal bridge has been recovered.

Each TimeMemEval example contains a source event e_s, a bridge temporal window W_b, a destination event e_d, and distractors $\mathcal{D}_{\text{dist}}$:

$$x = (q, e_s, W_b, e_d, \mathcal{D}_{\text{dist}}, a^\star).$$

The query makes e_s retrievable, but the answer is stored in e_d. The bridge window is derived from the event time of the source event:

$$W_b = g(t_{\text{event}}(e_s), \delta),$$

where g applies the sampled temporal relation or offset δ. A successful memory system must recover e_s, infer W_b, retrieve e_d, and synthesize a sufficient evidence pool for the answerer.

The construction process is event-graph-first. A persona or theme is sampled, then a source event is generated. A temporal offset defines the bridge window, and a destination event containing the gold answer is placed relative to that window. Hard distractors are added near the same topic, time period, entity, or answer type. The event graph is then realized as multi-session dialogue with calendar-based, relative, or indirect temporal phrasing. Automatic checks remove invalid generations, temporal inconsistencies, answer ambiguity, and split leakage.

TimeMemEval is not intended to model every real memory failure. It isolates the case where a system must use a temporal bridge to make the final evidence retrievable. That makes it especially useful for evaluating whether the Temporal Evidence Pool and iterative loop solve a problem that ordinary one-shot retrieval does not directly target.

4

Experiment and Analysis

This chapter evaluates NanoMem as a memory evidence provider. The central question is whether the system can construct compact, source-grounded, and temporally grounded evidence for long-term conversational memory questions. The chapter first describes the experimental setup, including benchmarks, baselines, metrics, and training/evaluation boundaries. It then reports the main benchmark results on LoCoMo, LongMemEval, and TimeMemEval, followed by a comparative analysis across baseline families. The final sections analyze architecture and reward ablations to explain which parts of NanoMem account for the observed behavior.

4.1 Experimental Setup

4.1.1 Benchmarks

I evaluate NanoMem on two public long-term memory benchmarks and one diagnostic benchmark constructed for this thesis. The three benchmarks are complementary: LoCoMo and LongMemEval establish whether NanoMem remains effective on standard long-term conversational memory tasks, while TimeMemEval isolates the hidden-dependency temporal bridge setting that motivates the method.

LoCoMo evaluates long-term multi-session dialogue memory with single-hop, multi-hop, temporal reasoning, and open-domain questions [22]. It is useful for testing cross-session evidence localization and whether a memory method remains accurate when relevant facts are distributed across long dialogue histories.

LongMemEval focuses on long-term memory for chat assistants and contains single-session user/assistant questions, knowledge updates, temporal reasoning, and multi-session reasoning [34]. Compared with LoCoMo, its histories are noisier and closer to assistant-style long-memory usage, making it suitable for testing whether compact evidence construction can outperform direct full-context prompting.

TimeMemEval is the diagnostic benchmark introduced in this thesis. Each instance requires a model to find a source event, derive a target temporal window, retrieve a destination event in that window, and answer a terminal question about the destination event. The benchmark is not intended to cover all temporal reasoning phenomena. Instead, it specifically tests the evidence addressability gap: the original query does not directly contain the retrieval cue needed to find the final answer-bearing evidence.

The training and evaluation split is reported in Appendix A.2. In brief, GRPO training uses 500 examples from LoCoMo and TimeMemEval, while LongMemEval

is reserved for zero-shot evaluation.

4.1.2 Baselines

I compare NanoMem with four groups of baselines. Full-context baselines feed the complete available history directly to GPT-5.1 or Qwen3.5-9B. These rows serve as strong long-context references, but they are not memory systems and do not represent the lowest-cost deployment option. Write-time memory systems, including Mem0 [4] and A-Mem [35], construct or organize memory records before the future query is known. Search-time retrieval agents, including MemR^3 [6] and Search-R1 [17], retrieve or search at query time but do not maintain NanoMem's structured temporal evidence pool. The temporal baseline is Memory-T1-Qwen [7]. Since the official trained Memory-T1 checkpoint was not available, this row is a Memory-T1-style implementation using Qwen3.5-9B rather than official Memory-T1 checkpoint performance.

NanoMem is reported in two variants. NanoMem-Vanilla uses the same Planner–Retriever–Synthesizer framework without GRPO training. NanoMem-GRPO uses the trained Synthesizer. The comparison between the two variants separates the contribution of the framework from the learned evidence construction policy.

4.1.3 Metrics and Evaluation Protocol

The main metric is answer accuracy. A fixed GPT-5.1 judge compares each predicted answer with the gold answer under the same judge prompt across all methods. I do not use BLEU or F1 as the primary metric because long-term memory answers often admit semantically equivalent natural-language variants, especially for temporal expressions and event descriptions.

Token counts are reported as output-side compactness metrics and must be read carefully. For direct-answerer methods, tokens denote answer-output tokens. For evidence-provider methods such as NanoMem, Mem0, A-Mem, and MemR^3, tokens denote retrieved evidence or memory context returned downstream. They are therefore useful for comparing evidence compactness, but they are not identical to full input cost for every method.

For NanoMem-specific diagnostics, I additionally report retrieval rounds and verdict accuracy. Retrieval rounds measure how often the iterative loop is used. Fewer rounds are not automatically better: the desired behavior is to stop when the evidence is sufficient and continue when a missing bridge must be recovered. Verdict accuracy measures whether the Synthesizer's sufficient/insufficient decision agrees with gold evidence coverage.

During training, the Qwen3.5-9B Synthesizer is optimized with full-parameter GRPO using VERL/HybridFlow [28]. The Planner, NanoMem retriever, Qwen3.5-9B downstream answerer, GPT-5.1 judge, and reference policy are frozen environment components. Each prompt samples $K = 8$ trajectories with at most three retrieval rounds. At inference time, NanoMem uses the same three-round limit and retrieves the top-7 memories in each round. The final run uses reward weights $(w_o, w_v, w_l) = (0.50, 0.10, 0.05)$ and a format-failure penalty $c = -0.20$. Training

Group	Method	LoCoMo			LongMemEval			TimeMemEval		
		Multi-hop	Temporal	Overall	TR	MR	Overall	Normal	Hard	Overall
Full Context	GPT-5.1	88.7	76.0	87.7	39.8	44.4	63.6	25.0	7.0	16.0
Full Context	Qwen3.5-9B	84.0	55.5	80.8	39.8	45.1	63.6	25.0	16.0	20.5
Write	Mem0	79.3	43.0	71.0	54.2	46.2	59.2	2.0	1.0	1.5
Write	A-Mem	78.4	65.1	76.1	45.0	50.0	72.0	5.0	10.0	7.5
Search	MemR^3	67.4	64.8	73.4	55.0	50.0	68.0	13.0	11.0	12.0
Search	Search-R1	47.2	23.1	44.5	10.5	3.0	22.8	–	–	6.5
Temporal	Memory-T1-Qwen	75.2	52.6	74.7	27.1	24.8	45.1	3.0	4.0	3.5
Ours	**NanoMem-Vanilla**	84.0	76.6	85.0	75.8	70.7	78.7	47.0	39.0	43.0
Ours	**NanoMem-GRPO**	89.0	87.5	88.6	86.4	71.4	83.8	57.0	47.0	51.5

Table 4.1: Summary of main benchmark accuracy. All entries are accuracy percentages; detailed category-level results and token counts are reported in section 4.2.

is conducted on four NVIDIA GH200 96GB GPUs.

4.2 Benchmark Results

Table 4.1 summarizes the main accuracy results across LoCoMo, LongMemEval, and TimeMemEval. NanoMem-GRPO obtains the best overall accuracy on all three benchmarks: 88.6% on LoCoMo, 83.8% on LongMemEval, and 51.5% on TimeMemEval. Compared with NanoMem-Vanilla, GRPO raises the overall score from 85.0% to 88.6% on LoCoMo, from 78.7% to 83.8% on LongMemEval, and from 43.0% to 51.5% on TimeMemEval. The largest relative gain appears on TimeMemEval, where evidence is not directly addressable from the original query and the system must recover an intermediate temporal bridge before continuing search.

4.2.1 LoCoMo

On LoCoMo, NanoMem-GRPO reaches 88.6% overall accuracy, slightly above the GPT-5.1 full-context reference at 87.7%. The category-level pattern is more informative than the overall score alone. NanoMem-GRPO improves over NanoMem-Vanilla on multi-hop questions from 84.0% to 89.0%, and on temporal questions from 76.6% to 87.5%. It also remains strong on single-hop questions at 91.4%, showing that the temporal and multi-hop optimization does not break ordinary memory recall. The open-domain score is 65.2%, slightly below NanoMem-Vanilla's 67.4%, so the correct interpretation is not that every category improves uniformly. The gain is concentrated in the multi-hop and temporal categories targeted by the method.

The token counts also show that the improvement is not caused by returning a larger context. NanoMem-GRPO returns 107 tokens on average, compared with 264 for NanoMem-Vanilla, 650 for Mem0, and 12.3k for A-Mem. On LoCoMo, the trained Synthesizer therefore improves accuracy while producing substantially more compact evidence.

Group	Method	Single-hop		Multi-hop		Temporal		Open		Overall	
		Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.
Full Context	GPT-5.1	94.2	25.0	88.7	32.1	76.0	26.8	67.7	36.0	87.7	27.4
Full Context	Qwen3.5-9B	92.2	18.2	84.0	23.6	55.5	15.5	56.2	39.4	80.8	19.9
Write	Mem0	79.5	675	79.3	650	43.0	650	67.1	675	71.0	650
Write	A-Mem	81.0	12.5k	78.4	12.4k	65.1	11.9k	63.5	12.0k	76.1	12.3k
Search	MemR^3	82.5	94.8	67.4	118.2	64.8	182.8	41.7	168.5	73.4	122.2
Search	Search-R1	52.9	5.2	47.2	4.1	23.1	2.8	35.4	3.1	44.5	4.4
Temporal	Memory-T1-Qwen	86.3	44.6	75.2	49.0	52.6	43.1	44.8	48.7	74.7	45.4
Ours	**NanoMem-Vanilla**	90.5	234	84.0	364	76.6	225	67.4	375	85.0	264
Ours	**NanoMem-GRPO**	91.4	99	89.0	138	87.5	88	65.2	152	88.6	107

Table 4.2: LoCoMo results by category. Acc. is LLM-judge accuracy (%), Tok. is output tokens, and k denotes thousands.

Group	Method	SSU		SSA		KU		TR		MR		Overall	
		Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.
Full Context	GPT-5.1	95.7	30.8	96.4	47.9	84.6	29.1	39.8	52.2	44.4	51.6	63.6	42.3
Full Context	Qwen3.5-9B	97.1	76.2	100.0	74.0	79.5	147.4	39.8	254.7	45.1	189.0	63.6	172.3
Write	Mem0	82.3	209	89.5	207	47.4	220	54.2	220	46.2	220	59.2	217
Write	A-Mem	95.0	18.7k	95.0	15.7k	75.0	18.7k	45.0	18.6k	50.0	18.7k	72.0	18.1k
Search	MemR^3	90.0	50.4	95.0	86.2	50.0	101.7	55.0	175.2	50.0	140.4	68.0	110.8
Search	Search-R1	38.6	28.6	28.6	19.3	33.3	13.6	10.5	24.9	3.0	26.7	22.8	22.6
Temporal	Memory-T1-Qwen	64.3	49.7	64.3	55.0	44.9	49.9	27.1	62.2	24.8	63.9	45.1	56.1
Ours	**NanoMem-Vanilla**	94.3	104	94.6	245	71.8	357	75.8	417	70.7	462	78.7	353
Ours	**NanoMem-GRPO**	95.7	81	98.2	94	79.5	109	86.4	124	71.4	149	83.8	119

Table 4.3: LongMemEval results. SSU/SSA: single-session user/assistant; KU: knowledge update; TR: temporal reasoning; MR: multi-session reasoning; k: thousands.

4.2.2 LongMemEval

The gap between full-context prompting and evidence construction becomes larger on LongMemEval. NanoMem-GRPO reaches 83.8% overall accuracy, compared with 63.6% for GPT-5.1 full context. This should not be read as a general claim that GPT-5.1 is weak. Instead, it shows that in long and noisy histories, directly asking the answer model to process the whole context can be less effective than selecting and synthesizing the relevant evidence first.

The temporal reasoning category is the clearest result. NanoMem-GRPO reaches 86.4% on LongMemEval-TR, compared with 54.2% for Mem0, 55.0% for MemR^3, and 27.1% for Memory-T1-Qwen. NanoMem-GRPO also remains strong on single-session categories, reaching 95.7% on SSU and 98.2% on SSA. The multi-session reasoning score is 71.4%, slightly above NanoMem-Vanilla at 70.7%.

LongMemEval also confirms the compactness pattern. NanoMem-GRPO returns 119 tokens on average, compared with 353 for NanoMem-Vanilla, 217 for Mem0, and 18.1k for A-Mem. Thus, the improvement over full-context prompting and write-

Group	Method	TimeMemEval-Normal						TimeMemEval-Hard						Overall	
		En		Cond		Time		En		Cond		Time		Acc.	Tok.
		Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.		
Full Context	GPT-5.1	30.0	54.1	26.7	57.0	16.7	74.3	10.0	41.5	6.7	44.8	3.3	77.9	16.0	57.2
Full Context	Qwen3.5-9B	27.5	321.6	16.7	249.6	30.0	671.5	25.0	678.4	6.7	615.4	13.3	835.5	20.5	555.8
Write	Mem0	2.5	180.5	3.3	164.1	0.0	172.3	2.5	169.4	0.0	160.5	0.0	181.3	1.5	171.7
Write	A-Mem	5.0	3.7k	6.7	3.8k	3.3	3.8k	15.0	3.8k	10.0	3.8k	3.3	3.8k	7.5	3.8k
Search	MemR^3	17.5	495.8	13.3	300.1	6.7	1336.8	20.0	872.7	3.3	560.5	6.7	405.9	12.0	664.2
Search	Search-R1	10.5	103.5	6.7	78.4	3.3	58.2	12.5	67.8	3.3	59.4	3.3	58.9	6.5	62.1
Temporal	Memory-T1-Qwen	2.5	37.5	6.7	43.4	0.0	44.5	7.5	31.8	0.0	43.7	3.3	32.8	3.5	38.5
Ours	NanoMem-Vanilla	42.5	318	53.3	312	46.7	267	40.0	258	33.3	239	43.3	330	43.0	287
Ours	NanoMem-GRPO	52.5	221	66.7	266	53.3	226	47.5	216	46.7	256	46.7	232	51.5	234

Table 4.4: TimeMemEval results by difficulty and terminal type. En, Cond, and Time denote entity, condition, and explicit-time questions; Tok. is output tokens.

time memory systems is paired with shorter evidence rather than larger returned context.

4.2.3 TimeMemEval

TimeMemEval is harder in absolute terms, which is expected because it is a diagnostic temporal-bridge benchmark. NanoMem-GRPO reaches 51.5% overall accuracy, compared with 43.0% for NanoMem-Vanilla. The normal split reaches 57.0% and the hard split reaches 47.0%. Full-context baselines remain much lower: GPT-5.1 full context reaches 16.0%, and Qwen3.5-9B full context reaches 20.5%. Existing memory and retrieval baselines are also far below NanoMem on this diagnostic task, with Mem0 at 1.5%, A-Mem at 7.5%, MemR^3 at 12.0%, Search-R1 at 6.5%, and Memory-T1-Qwen at 3.5%.

The absolute accuracy remains far from saturated, which is an important part of the result. TimeMemEval is intentionally difficult: the final evidence is hidden behind an intermediate event, time window, or condition. NanoMem does not solve this problem completely, but it improves substantially over both full-context and existing memory baselines. This supports the central claim that temporal bridge search requires more than one-shot semantic matching.

4.3 Comparative Analysis

4.3.1 Full-context Prompting versus Memory Evidence Construction

Full-context prompting is a useful reference because it gives the answer model access to the full available history. However, the results show that access is not the same as usable evidence. On LoCoMo, NanoMem-GRPO is close to the GPT-5.1 full-context reference and slightly exceeds it overall, 88.6% versus 87.7%, while returning compact evidence rather than relying on the complete dialogue history. On

LongMemEval, the gap is larger: NanoMem-GRPO reaches 83.8%, while GPT-5.1 full context reaches 63.6%. On TimeMemEval, GPT-5.1 full context reaches only 16.0%, indicating that even when the relevant sessions are available in the prompt, the model can fail to recover the temporal bridge that connects source and destination evidence.

The implication is not that long-context models are unhelpful. Rather, long-term conversational memory requires evidence construction. A memory system should identify what the downstream answerer needs to see, preserve source grounding, and reduce irrelevant context. This is the role NanoMem is designed to play.

4.3.2 Write-time Memory versus Query-time Evidence Synthesis

Write-time memory systems such as Mem0 and A-Mem construct memory records before the future query is known. This is appropriate for stable user facts, preferences, and profile-style memories. The limitation appears when the relevant relation becomes visible only after the query is asked. TimeMemEval exposes this limitation directly: Mem0 reaches 1.5% and A-Mem reaches 7.5%, while NanoMem-GRPO reaches 51.5%.

The token profile also shows a practical trade-off. A-Mem returns transcript- level evidence in these experiments, with 12.3k tokens on LoCoMo, 18.1k on LongMemEval, and 3.8k on TimeMemEval. This may preserve recall, but it shifts a large burden to the downstream answerer. NanoMem instead performs lightweight write-time processing and uses query-time evidence synthesis to decide what is relevant for the current question.

4.3.3 Search-time Retrieval versus Structured Temporal Evidence State

Search-time retrieval agents are closer to NanoMem because they react to the current query. MemR^3 and Search-R1 can retrieve or search at query time, but they do not maintain NanoMem's structured Temporal Evidence Pool or the binary sufficient/insufficient verdict loop. This difference is most visible on LongMemEval temporal reasoning and TimeMemEval. MemR^3 reaches 55.0% on LongMemEval-TR and 12.0% on TimeMemEval, while NanoMem-GRPO reaches 86.4% and 51.5%, respectively.

Search-R1 should be interpreted carefully. Its public checkpoint was trained for open-domain search QA, not timestamped long-term conversational memory. The reported row therefore evaluates a search-style baseline under a memory adapter, not the upper bound of the Search-R1 method family. Still, the comparison illustrates a key distinction: NanoMem's search is not only about issuing more queries. It converts retrieved sessions into a temporally grounded evidence state that can guide the next retrieval step.

4.3.4 Accuracy and Compactness

Compactness is not a cosmetic metric in this thesis. NanoMem is designed as an evidence provider, so the downstream answer model should see a concise evidence state rather than a large bag of raw sessions. Table 4.5 compares NanoMem-Vanilla with NanoMem-GRPO. GRPO improves accuracy while reducing returned evidence length on all three benchmarks. On LoCoMo, output length drops from 264 to 107 tokens; on LongMemEval, from 353 to 119 tokens; on TimeMemEval, from 287 to 234 tokens.

Table 4.5: NanoMem-Vanilla versus NanoMem-GRPO accuracy and returned evidence compactness.

Benchmark	Vanilla acc.	Vanilla tok.	GRPO acc.	GRPO tok.	Acc. gain	Token reduction
LoCoMo	85.0	264	88.6	107	+3.6	59.5%
LongMemEval	78.7	353	83.8	119	+5.1	66.3%
TimeMemEval	43.0	287	51.5	234	+8.5	18.5%

This result should be interpreted as returned evidence compactness, not total system cost. The input cost of each method depends on its retrieval procedure, history length, and implementation. Nevertheless, the pattern is important: NanoMem-GRPO improves the answer outcome without passing more evidence to the answerer.

4.4 Architecture Ablation

The architecture ablation isolates three design choices: temporal text normalization, the Temporal Evidence Pool, and multi-round retrieval. The full row uses NanoMem-Vanilla rather than NanoMem-GRPO so that the ablation measures framework components without mixing in the effect of the trained policy.

Variant	LoCoMo-MH	LoCoMo-Temp	LME-TR	TimeMemEval	Avg. Rounds
Full NanoMem-Vanilla	84.0	76.6	75.8	43.0	1.25
w/o text normalization	82.3	71.3	64.9	33.5	1.29
w/o temporal evidence pool	77.0	67.6	60.6	25	1.45
Single-round retrieval	78.0	78.5	78.0	32.0	1.0

Table 4.6: Architectural ablation of NanoMem. MH, Temp, TR, and Avg. Rounds denote multi-hop, temporal, temporal reasoning, and mean retrieval rounds.

Text normalization mainly supports temporal grounding. Removing it barely affects LoCoMo-MH, which drops from 84.0% to 82.3%, but it causes larger drops on LoCoMo-Temp, LongMemEval-TR, and TimeMemEval: 5.3, 10.9, and 9.5 points, respectively. This indicates that explicit normalization is most useful when the query or memory contains relative time expressions, mismatched session/event times, or a temporal bridge.

The Temporal Evidence Pool is more broadly important. Removing it lowers every reported task score. The largest drops appear on LongMemEval-TR and TimeMemEval, by 15.2 and 18.0 points. It also increases average rounds from 1.25 to 1.45. This

means the loop continues searching, but without a structured state of temporally resolved events, later rounds become less effective.
Single-round retrieval shows why the value of iteration is task-dependent. It hurts LoCoMo-MH and TimeMemEval, where first-round retrieval is often incomplete, but it slightly improves LoCoMo-Temp and LongMemEval-TR. Many temporal questions in those categories are answerable from first-round evidence, and extra search can add noise. NanoMem's goal is therefore not to increase the number of rounds, but to learn when evidence is sufficient and when another search step is necessary.

4.5 Reward Ablation

The reward ablation evaluates whether GRPO changes the Synthesizer's evidence construction behavior. The full reward combines answer outcome, verdict accuracy, compactness, and the format gate described in Chapter 3. Two variants remove different parts of this signal: one removes the format and length rewards, and the other removes the verdict reward.

Variant	LoCoMo-MH	LoCoMo-Temp	LME-TR	TimeMemEval	Verdict Acc.	Avg. Rounds	Avg. Tok.
Full reward	89.0	87.5	86.4	51.5	84.0	1.1	110
- remove R_{fmt} and R_{len}	87.6	85.4	84.1	48	84.9	1.1	359
- remove $R_{verdict}$	80.5	89.1	90.2	41.5	72.9	1.25	361

Table 4.7: Reward ablation for GRPO training. Verdict Acc. is per-round sufficient/insufficient accuracy against gold evidence coverage.

The full reward reaches 89.0% on LoCoMo-MH, 87.5% on LoCoMo-Temp, 86.4% on LongMemEval-TR, and 51.5% on TimeMemEval. It also keeps verdict accuracy high at 84.0%, with only 1.10 average rounds and 110 average output tokens.
Removing the format and length rewards produces a compactness failure. Accuracy drops moderately and consistently, but the more visible effect is output length: average output increases from 110 to 359 tokens, a 3.3 times increase. These rewards are therefore not merely cosmetic formatting constraints. They regularize the Synthesizer against satisfying the outcome objective by passing large, redundant evidence to the downstream answerer.
Removing the verdict reward exposes a different failure mode. The outcome-only variant improves on LoCoMo-Temp and LongMemEval-TR, but it drops sharply on LoCoMo-MH and TimeMemEval, from 89.0% to 80.5% and from 51.5% to 41.5%, respectively. Verdict accuracy also falls from 84.0% to 72.9%, average rounds increase from 1.10 to 1.25, and average tokens increase to 361. This pattern is consistent with the task structure: when first-round evidence is often sufficient, outcome-only training can be strong; when a multi-hop or temporal bridge is missing, the policy needs direct supervision for the sufficient/insufficient decision.

4.6 Summary of Findings

The experiments support four findings. First, NanoMem-GRPO obtains the best overall accuracy on LoCoMo, LongMemEval, and TimeMemEval, with gains concen-

trated in multi-hop, temporal reasoning, and hidden-dependency temporal bridge settings. Second, the comparative analysis shows that full-context prompting, write-time memory, and generic search-time retrieval do not fully address temporal evidence addressability. NanoMem's advantage comes from query-time evidence synthesis, the Temporal Evidence Pool, and the structured verdict loop. Third, the compactness analysis shows that GRPO improves accuracy without context bloat: NanoMem-GRPO returns substantially shorter evidence than NanoMem-Vanilla on all three benchmarks. Fourth, the ablations show that the main components play distinct roles: text normalization supports temporal grounding, the Temporal Evidence Pool supports cross-round state transfer, multi-round retrieval supports hidden-dependency search, and the reward components shape compactness and sufficiency judgment.

The next chapter uses these findings to answer the research questions and discuss limitations, including automatic judge dependence, baseline adaptation boundaries, non-equivalent token metrics, the diagnostic scope of TimeMemEval, and deployment concerns for long-term personal memory systems.

5

Conclusion

This chapter closes the thesis by returning to the research questions introduced in Chapter 1. Chapter 4 showed that NanoMem improves accuracy and returned-evidence compactness on LoCoMo, LongMemEval, and TimeMemEval, with the largest relative gain on the diagnostic temporal-bridge setting. The role of this chapter is not to repeat those tables, but to explain what the results imply for long-term conversational agent memory, where the current method remains limited, and what ethical and deployment issues remain before such a system can be used as personal memory infrastructure.

5.1 RQ1: Query-Time Evidence Construction

RQ1. How can a long-term conversational memory system construct evidence at query time while preserving the original memory structure and avoiding heavy query-agnostic interpretation during ingestion?

NanoMem answers this question by treating memory access as query-time evidence construction rather than write-time memory rewriting. The system does not rely on a single precomputed memory record as the final representation of a past conversation. Instead, it keeps the original session structure available, uses lightweight indexing and temporal support at ingestion time, and delays the main interpretation step until a concrete user query is available. This design is important because the future question determines which part of a memory is evidence. A detail that looks unimportant at write time may become the key bridge for a later temporal or multi-hop question.

The Temporal Evidence Pool is the main mechanism that makes this answer concrete. It stores compact events rather than raw sessions. Each event is tied to a source session and carries both session time and event time. The memory module therefore returns an inspectable evidence state to the downstream answer model instead of a long, noisy history dump. This preserves source attribution while reducing the burden on the answerer: the downstream model no longer has to perform retrieval validation, evidence selection, temporal grounding, compression, and final answer generation all at once.

The experimental results support this design. Compared with NanoMem-Vanilla, NanoMem-GRPO improves accuracy while reducing the amount of returned evidence on all three benchmarks. On LoCoMo, returned evidence drops from 264 to 107 tokens; on LongMemEval, from 353 to 119 tokens; and on TimeMemEval, from 287 to 234 tokens. These numbers should be interpreted as returned-evidence

compactness rather than full serving cost, but they show that query-time evidence synthesis can improve answer usefulness without simply passing more context downstream.
The answer to RQ1 is therefore that a long-term conversational memory system can preserve the original memory structure by keeping ingestion lightweight and moving the main interpretation into a query-time evidence provider. The memory module should not be forced to choose, at write time, a single permanent summary of what a session means. It can instead construct source-grounded, temporally grounded evidence when the user query makes the relevant relation visible.

5.2 RQ2: Multi-Hop Temporal Evidence Recovery

RQ2. How can a memory agent recover temporally grounded and multi-hop evidence when the retrieval cue needed for the answer is not directly present in the original query?
The central difficulty behind RQ2 is the evidence addressability gap. In many long-term memory questions, the original query does not contain the phrase, date, entity, or condition needed to retrieve the final answer-bearing session. For example, a question may ask about what happened during the week of another event. The system must first find the anchor event, derive a temporal or semantic cue from it, and only then search for the destination evidence.
The system addresses this problem through an iterative Planner–Retriever–Synthesizer loop. The important point is not merely that it performs multiple retrieval rounds. The important point is that each round updates the evidence state. Synthesized events from earlier rounds enter the Temporal Evidence Pool, and the following round can use this pool as part of the search state. This turns intermediate evidence into a new retrieval address. When the first round finds a bridge event rather than the final answer, the system has a structured way to carry that bridge forward.
Temporal grounding is essential to this process. Long-term conversational memory has at least two relevant times: the time when a session occurred and the time when the described event happened. These are often different. If a user says on April 2 that a trip happened last week, the session time is April 2, but the event time is the previous week. NanoMem keeps this distinction in the evidence state, allowing event time to guide later retrieval rather than remaining only as unstructured text for the answer model.
The ablation results show that this evidence state is not an optional detail. Removing the Temporal Evidence Pool lowers performance across the reported tasks, with especially large drops on LongMemEval temporal reasoning and TimeMemEval. Forcing single-round retrieval also hurts settings where the relevant answer requires an intermediate bridge. At the same time, the results show that iteration is not automatically beneficial for every question. Some questions are answerable from first-round evidence, and extra search can introduce noise. This is why the binary `sufficient/insufficient` verdict is a core part of the design: the system must learn not only how to continue searching, but also when to stop.

The answer to RQ2 is therefore that temporally grounded multi-hop evidence can be recovered by maintaining a structured evidence state across retrieval rounds. The memory agent needs to transform partial evidence into the next search cue, while using event time and sufficiency judgments to avoid both premature stopping and unnecessary over-search.

5.3 RQ3: Reinforcement Learning for Synthesizer Training

RQ3. How can reinforcement learning be used to train the Synthesizer to produce useful, compact, temporally grounded evidence and to judge whether the current evidence is sufficient for downstream answering?

The role of reinforcement learning in NanoMem is to align the Synthesizer with the memory-side responsibilities defined by the architecture. The Synthesizer does not directly answer the user. It extracts and compresses evidence, grounds events in source sessions and event times, and predicts whether the current evidence is sufficient. GRPO is used to train this behavior inside the same iterative environment in which the Synthesizer will be used at inference time.

The reward design reflects this separation of responsibilities. The outcome reward measures whether the accumulated evidence helps a frozen downstream answerer produce the correct answer. The verdict reward trains the binary sufficiency decision against gold evidence coverage. The length reward encourages compactness when the answer outcome is correct. The format gate enforces parseability at the trajectory level: if a trajectory is malformed, it receives the fixed format penalty and no other reward terms are added. This design avoids treating format as just another additive preference.

The reward ablation supports the need for these components. Removing the format and length rewards causes returned evidence to become much longer, showing that compactness does not emerge reliably from answer accuracy alone. Removing the verdict reward produces a different failure mode: some temporal categories can remain strong, but multi-hop and hidden-bridge settings degrade, verdict accuracy drops, and the system tends to search more while returning longer evidence. This pattern is consistent with the task structure. When first-round evidence is enough, outcome-only training can be effective; when the system must decide whether a bridge is still missing, sufficiency supervision becomes more important.

The answer to RQ3 is therefore that reinforcement learning is useful when it is matched to the memory interface. GRPO does not replace the need for an evidence-state architecture. Instead, it trains the Synthesizer to operate within that architecture: produce parseable evidence, keep it compact, preserve temporal grounding, and decide whether the accumulated evidence is enough for downstream answering.

5.4 Implications

These implications extend and contextualise the answers to RQ1, RQ2, and RQ3. The first implication concerns how long-term memory systems should be evaluated, following from the RQ1 finding that query-time evidence construction outperforms storage-capacity-based approaches. The second concerns the output boundary of the memory module, drawing on both RQ1 and RQ2. The third treats event time as active search state, which is central to the RQ2 finding on temporal evidence recovery. The fourth discusses the complementarity of write-time and query-time memory approaches, integrating the results across all three research questions.

The first implication is that long-term agent memory should not be evaluated only by storage capacity or context length. A larger context window can expose more history to the model, but it does not guarantee that the model will find the right evidence, resolve the right time relation, or ignore irrelevant sessions. The results on LongMemEval and TimeMemEval show that full-context prompting can fail even when the answer-bearing information is present. The hard part is not merely access to history, but construction of usable evidence.

The second implication is that the output boundary of a memory module matters. If the memory module returns raw sessions, the answerer has to solve too many subproblems at once. If the memory module directly returns a final answer, the evidence construction process becomes hard to inspect. NanoMem takes a narrower position: memory returns evidence and a verdict, while final answer generation remains outside the memory module. This boundary makes errors easier to diagnose. A failure can be attributed more specifically to retrieval recall, temporal grounding, premature sufficiency, evidence bloat, or downstream answer generation.

The third implication is that event time should be treated as search state, not only as metadata. In conversational memory, time is often the thing that makes a hidden dependency addressable. A past event may be mentioned after it occurred, or an answer-bearing session may be reachable only through a time window derived from another event. Keeping event time in the Temporal Evidence Pool allows time to participate in retrieval, rather than appearing only in the final prompt as text for the answerer to reason over.

The fourth implication is that query-time and write-time memory methods should be seen as complementary rather than mutually exclusive. Write-time memory is useful for stable user preferences, profiles, and frequently reused facts. The results of this thesis suggest that it is less suitable as the only mechanism for questions whose relevance depends on a future query. Query-time evidence synthesis gives the system a way to interpret the memory in relation to the current information need, while still preserving source grounding.

5.5 Limitations and Future Work

5.5.1 Limitations

The first limitation is retrieval recall. NanoMem can only synthesize evidence from sessions that are retrieved. If the Planner forms the wrong cue, the retriever misses

the answer-bearing session, or the candidate budget excludes a relevant memory, the Temporal Evidence Pool cannot recover information that was never observed. This is a general upper bound for search-based memory systems.

The second limitation is temporal normalization and event-time grounding. Deterministic temporal normalization helps with common relative-time expressions, but it does not cover all natural language time references, fuzzy intervals, recurring events, time zones, cultural calendars, or changing future plans. The Synthesizer can also make grounding errors by confusing session time with event time or by compressing a broad interval into a narrow date. Because event time is used as search state, these errors can propagate into later retrieval rounds.

The third limitation is the expressiveness of the structured schema. The final schema uses source-grounded events and a binary verdict. This makes parsing, training, and evaluation more stable, but it gives limited room for uncertainty, conflict, partial support, negative evidence, or multi-source inference. The schema is therefore a pragmatic research interface, not a final representation for all personal memory systems.

The fourth limitation is evaluation dependence. Final answer accuracy measures whether the evidence helps a fixed downstream answerer produce the right answer, but it is not a direct measure of evidence faithfulness. A correct evidence state can still lead to a wrong answer, and a strong answerer can sometimes compensate for incomplete evidence. The GPT-5.1 judge provides a consistent automatic evaluation boundary, but any LLM judge may prefer certain wordings, miss equivalent answers, or fail to detect source-grounding errors.

The fifth limitation is benchmark coverage. LoCoMo and LongMemEval are important public long-term memory benchmarks, and TimeMemEval isolates the temporal-bridge problem targeted by this thesis. However, these benchmarks do not cover every realistic personal memory scenario. Real assistants must handle user correction, memory deletion, stale preferences, cross-application data, privacy policies, adversarial queries, and long-term drift. The present work should therefore be read as a research prototype and diagnostic evaluation, not as a complete deployed memory service.

The sixth limitation is cost. NanoMem reduces the evidence passed to the downstream answerer, but the loop itself may require multiple retrieval rounds, Synthesizer calls, and answerer or judge calls during evaluation. The token compactness results in Chapter 4 measure returned evidence length, not full end-to-end latency, cost, or energy use.

5.5.2 Threats to Validity

Beyond the specific limitations listed above, the following threats to the validity of the conclusions are worth making explicit.

Internal validity. The main internal threat is evaluation dependence on a fixed GPT-5.1 judge. All methods share the same judge and the same judge prompt, so relative comparisons between systems are internally consistent. However, the absolute accuracy scores may shift under a different judge or a different judging protocol. A secondary internal threat is the Memory-T1-Qwen baseline, which is

a re-implementation rather than an official trained checkpoint and may understate the best achievable Memory-T1 performance on these benchmarks.

Construct validity. Token counts are used as a proxy for evidence compactness, but they measure output length rather than information content, redundancy, or total serving cost. Verdict accuracy is measured against retrieved-session coverage of gold evidence sessions, which approximates sufficiency rather than directly measuring evidence quality. Final answer accuracy is itself a proxy for evidence-state quality: a strong downstream answerer may compensate for incomplete evidence, and correct evidence may still produce a wrong answer under a weak answerer.

External validity. The benchmarks used are English-language datasets with controlled question types. The results may not generalise to non-English languages, multimodal sessions, real user conversations with unconstrained question forms, or high-stakes domains such as medical or legal contexts. TimeMemEval was constructed using the same temporal-bridge design principles as the NanoMem framework, which means it may advantage NanoMem's approach more than a fully independent benchmark would.

5.5.3 Future Work

One direction is to train the Planner and Retriever. This thesis focuses on training the Synthesizer while keeping the rest of the environment fixed. Future work could train a planner to generate better temporal and multi-hop cues, or train a retrieval policy that dynamically allocates budget across dense retrieval, lexical retrieval, temporal constraints, and source constraints.

A second direction is to extend the evidence schema. Future systems could add uncertainty, conflict, negative evidence, relation type, source confidence, and multi-source support. Such fields should be added only when matching supervision and evaluation are available; otherwise, the schema may become more complex without becoming more reliable.

A third direction is stronger temporal reasoning. Rule-based normalization can be expanded to more languages, time zones, fuzzy intervals, recurring events, and calendar constraints. More robust variants could combine LLM-based evidence synthesis with symbolic temporal reasoning or constraint checking, so that generated event times are verified before they influence later retrieval.

A fourth direction is direct evidence evaluation. Future work should measure source faithfulness, event-time correctness, evidence coverage, minimality, verdict reliability, and uncertainty calibration directly. This would separate retrieval success, evidence synthesis success, and downstream answer success more cleanly than final answer accuracy alone.

A fifth direction is real longitudinal deployment evaluation. A realistic personal memory system must support update, inspection, correction, deletion, retention, and user control. Long-running user studies or high-fidelity simulated agent environments are needed to test whether users can understand, trust, and correct structured memory evidence over time.

A final direction is multilingual and cross-domain memory. The current evaluation is centered on English benchmarks. Relative time, reference, and calendar conven-

tions differ across languages and domains. Testing Chinese, Swedish, multilingual conversations, workplace agents, educational tutors, code agents, and research assistants would clarify which parts of the evidence addressability gap are general and which are domain-specific.

5.6 Ethics

Long-term conversational memory is privacy-sensitive by design. A system that stores and searches user histories can accumulate information about health, finances, relationships, work plans, location patterns, beliefs, and personal vulnerabilities. The risk is not limited to storing individual sessions. Query-time evidence synthesis can connect information across sessions and reveal patterns that were never stated in one place.

Source-grounded evidence has both benefits and risks. On the positive side, NanoMem makes memory use more auditable by returning source identifiers, session times, and event times. This can help users and developers inspect why an answer was produced. On the negative side, returning evidence can expose private material if access control is weak or if the Synthesizer includes irrelevant details. Compact evidence reduces exposure compared with raw history injection, but it does not replace consent, access control, or policy-aware retrieval.

Users should be able to control the memory lifecycle. A deployed system should support consent, inspection, correction, deletion, retention limits, and clear explanations of when memory is used. This is especially important because personal memory can become stale. A preference, relationship, address, medical detail, or work plan may change, and an agent that retrieves old evidence without temporal qualification can give misleading or harmful advice.

High-risk domains require additional safeguards. In medical, legal, financial, employment, or mental-health settings, memory evidence should be treated as supporting context rather than authoritative truth. Premature sufficiency, event-time drift, and source-attribution errors can cause real harm. Systems using NanoMem-like evidence synthesis in such settings should require human verification, domain-specific policies, and conservative refusal or escalation behavior when evidence is incomplete.

The experiments in this thesis use publicly available benchmark datasets — LoCoMo, LongMemEval, and TimeMemEval — which contain no real personal user information. Training and inference were conducted in a closed computing environment using open-source Qwen models, with no data transmitted to external services. Under the EU Artificial Intelligence Act, the current system does not fall into a high-risk category: it does not perform biometric identification, does not operate in safety-critical infrastructure, and does not make legally binding decisions about individuals. This assessment applies to the research prototype described in this thesis; deployment in a high-risk context would require a separate compliance review.

Sustainability is also part of responsible deployment. NanoMem can reduce the evidence tokens passed downstream, but this does not prove lower end-to-end energy use. Training with GRPO, multi-round retrieval, LLM-based answering, and automatic judging all consume computational resources. A deployed system should

therefore use budget-aware control, caching where appropriate, smaller or local models for low-risk steps when possible, and transparent reporting of model calls and evaluation costs.

The thesis text was written independently by the author. After completing a full draft, ChatGPT was used for language polishing and phrasing improvements in selected passages. The development of experimental code, data processing pipelines, and evaluation scripts made use of Claude Code as a programming assistant. The author is responsible for all content, experimental descriptions, analysis, and conclusions in this thesis.

Overall, this thesis shows that long-term agent memory can be designed as query-time evidence construction rather than raw history injection or opaque answer generation. NanoMem makes progress toward compact, source-grounded, and temporally grounded memory access, especially for temporal and multi-hop queries. At the same time, reliable personal memory requires more than benchmark accuracy. It also requires stronger retrieval, better evidence verification, user control, privacy governance, and careful deployment in settings where memory errors can affect real decisions.

Bibliography

[1] David Abel, Will Dabney, Anna Harutyunyan, Mark K. Ho, Michael L. Littman, Doina Precup, and Satinder Singh. On the expressivity of markov reward. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

[2] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *International Conference on Learning Representations*, 2024.

[3] Yiqun Chen, Lingyong Yan, Weiwei Sun, Xinyu Ma, Yi Zhang, Shuaiqiang Wang, Dawei Yin, Yiming Yang, and Jiaxin Mao. Improving retrieval-augmented generation through multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, 2025.

[4] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. *CoRR*, abs/2504.19413, 2025.

[5] Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Haotian Wang, Ming Liu, and Bing Qin. TimeBench: A comprehensive evaluation of temporal reasoning abilities in large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1204–1228, Bangkok, Thailand, 2024. Association for Computational Linguistics.

[6] Xingbo Du, Loka Li, Duzhen Zhang, and Le Song. MemR^3: Memory retrieval via reflective reasoning for LLM agents. *CoRR*, abs/2512.20237, 2025.

[7] Yiming Du, Baojun Wang, Yifan Xiang, Zhaowei Wang, Wenyu Huang, Boyang Xue, Bin Liang, Xingshan Zeng, Fei Mi, Haoli Bai, Lifeng Shang, Jeff Z. Pan, Yuxin Jiang, and Kam-Fai Wong. Memory-T1: Reinforcement learning for temporal reasoning in multi-session agents. In *International Conference on Learning Representations*, 2026.

[8] Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Huajun Chen, and Ningyu Zhang. LightMem: Lightweight and efficient memory-augmented generation. *CoRR*, abs/2510.18866, 2025.

[9] Bahare Fatemi, Seyed Mehran Kazemi, Anton Tsitsulin, Karishma Malkan, Jinyeong Yim, John Palowitch, Sungyong Seo, Jonathan Halcrow, and Bryan Perozzi. Test of time: A benchmark for evaluating LLMs on temporal reasoning. In *International Conference on Learning Representations*, 2025.

[10] Jingsheng Gao, Linxu Li, Ke Ji, Weiyuan Li, Yixin Lian, Yuzhuo Fu, and Bin Dai. SmartRAG: Jointly learn RAG-related tasks from the environment feedback. In *International Conference on Learning Representations*, 2025.

[11] Yubin Ge, Salvatore Romeo, Jason Cai, Raphael Ren-Yian Shu, Monica Sunkara, Yassine Benajiba, and Yi Zhang. TReMu: Towards neuro-symbolic temporal reasoning for LLM-agents with memory in multi-session dialogues. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 18974–18988, 2025.

[12] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. REALM: Retrieval-augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, pages 3929–3938. PMLR, 2020.

[13] Jerry Huang, Siddarth Madala, Risham Sidhu, Cheng Niu, Hao Peng, Julia Hockenmaier, and Tong Zhang. RAG-RL: Advancing retrieval-augmented generation via RL and curriculum learning. *CoRR*, abs/2503.12759, 2025.

[14] Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 874–880, Online, 2021. Association for Computational Linguistics.

[15] Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. Few-shot learning with retrieval augmented language models. In *International Conference on Learning Representations*, 2023.

[16] Yuxiang Ji, Ziyu Ma, Yong Wang, Guanhua Chen, Xiangxiang Chu, and Liaoni Wu. Tree search for LLM agent reinforcement learning. In *International Conference on Learning Representations*, 2026.

[17] Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan O. Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Conference on Language Modeling*, 2025.

[18] Jingyi Kang, Chunyu Li, Ding Chen, Bo Tang, Feiyu Xiong, and Zhiyu Li. MemReader: From passive to active extraction for long-term agent memory. *CoRR*, abs/2604.07877, 2026.

[19] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 6769–6781, Online, 2020. Association for Computational Linguistics.

[20] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474, 2020.

[21] Zhiyu Li, Shichao Song, Chenyang Xi, Hanyu Wang, Chen Tang, Simin Niu, Ding Chen, Jiawei Yang, Chunyu Li, Qingchen Yu, Jihao Zhao, Yezhaohui

Wang, Peng Liu, Zehao Lin, Pengyuan Wang, Jiahao Huo, Tianyi Chen, Kai Chen, Kehang Li, Zhen Tao, Huayi Lai, Hao Wu, Bo Tang, Zhenren Wang, Zhaoxin Fan, Ningyu Zhang, Linfeng Zhang, Junchi Yan, Mingchuan Yang, Tong Xu, Wei Xu, Huajun Chen, Haofen Wang, Hongkang Yang, Wentao Zhang, Zhi-Qin John Xu, Siheng Chen, and Feiyu Xiong. MemOS: A memory OS for AI system. *CoRR*, abs/2507.03724, 2025.

[22] Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Bangkok, Thailand, 2024. Association for Computational Linguistics.

[23] MemPalace. MemPalace: Local-first AI memory. `https://github.com/MemPalace/mempalace`, 2026. Open-source software repository.

[24] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *CoRR*, abs/2310.08560, 2023.

[25] Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A temporal knowledge graph architecture for agent memory. *CoRR*, abs/2501.13956, 2025.

[26] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[27] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300, 2024.

[28] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, 2025.

[29] Miao Su, Yucan Guo, Zhongni Hou, Long Bai, Zixuan Li, Yufei Zhang, Guojun Yin, Wei Lin, Xiaolong Jin, Jiafeng Guo, and Xueqi Cheng. Beyond dialogue time: Temporal semantic memory for personalized LLM agents. *CoRR*, abs/2601.07468, 2026.

[30] Xiaqiang Tang, Yi Wang, Keyu Hu, Rui Xu, Chuang Li, Weigao Sun, Jian Li, and Sihong Xie. SSFO: Self-supervised faithfulness optimization for retrieval-augmented generation. *CoRR*, abs/2508.17225, 2025.

[31] Xiaoqiang Wang, Suyuchen Wang, Yun Zhu, and Bang Liu. R^3Mem: Bridging memory retention and retrieval via reversible compression. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 4541–4557, Vienna, Austria, 2025. Association for Computational Linguistics.

[32] Yu Wang, Ryuichi Takanobu, Zhiqi Liang, Yuzhen Mao, Yuanzhe Hu, Julian McAuley, and Xiaojian Wu. Mem-α: Learning memory construction via reinforcement learning. *CoRR*, abs/2509.25911, 2025.

[33] Zheng Wang, Zhongyang Li, Zeren Jiang, Dandan Tu, and Wei Shi. Crafting personalized agents through retrieval-augmented generation on editable memory graphs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 4891–4906, Miami, Florida, USA, 2024. Association for Computational Linguistics.

[34] Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking chat assistants on long-term interactive memory. *CoRR*, abs/2410.10813, 2024.

[35] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, 2025.

[36] Shi-Qi Yan, Quan Liu, and Zhen-Hua Ling. RPO: Retrieval preference optimization for robust retrieval-augmented generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5228–5240, Vienna, Austria, 2025. Association for Computational Linguistics.

[37] Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z. Pan, Hinrich Schütze, Volker Tresp, and Yunpu Ma. Memory-R1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *CoRR*, abs/2508.19828, 2025.

[38] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

[39] Juwei Yue, Chuanrui Hu, Jiawei Sheng, Zuyi Zhou, Wenyuan Zhang, Tingwen Liu, Li Guo, and Yafeng Deng. HyperMem: Hypergraph memory for long-term conversations. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, 2026. ACL 2026 Main.

[40] Guibin Zhang, Muxin Fu, Guancheng Wan, Miao Yu, Kun Wang, and Shuicheng Yan. G-Memory: Tracing hierarchical memory for multi-agent systems. In *Advances in Neural Information Processing Systems*, 2025.

[41] Zeyu Zhang, Quanyu Dai, Luyu Chen, Zeren Jiang, Rui Li, Jieming Zhu, Xu Chen, Yi Xie, Zhenhua Dong, and Ji-Rong Wen. MemSim: A bayesian simulator for evaluating memory of LLM-based personal assistants. In *Advances in Neural Information Processing Systems*, 2025.

[42] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. *CoRR*, abs/2312.07104, 2023.

[43] Mingkang Zhu, Xi Chen, Bei Yu, Hengshuang Zhao, and Jiaya Jia. Stratified GRPO: Handling structural heterogeneity in reinforcement learning of LLM search agents. *CoRR*, abs/2510.06214, 2025.

A

Supplementary Experimental Details

This appendix records the experimental details that are compressed in Chapter 4. The goal is to make the benchmark construction, model configuration, training setup, and baseline adaptations explicit enough for the results to be audited and reproduced.

A.1 Supplementary Architecture Overview

The main text separates NanoMem's evidence-state construction view and multi-round search view in Figures 3.1 and 3.2. For completeness, Figure A.1 shows the original full architecture in one diagram.

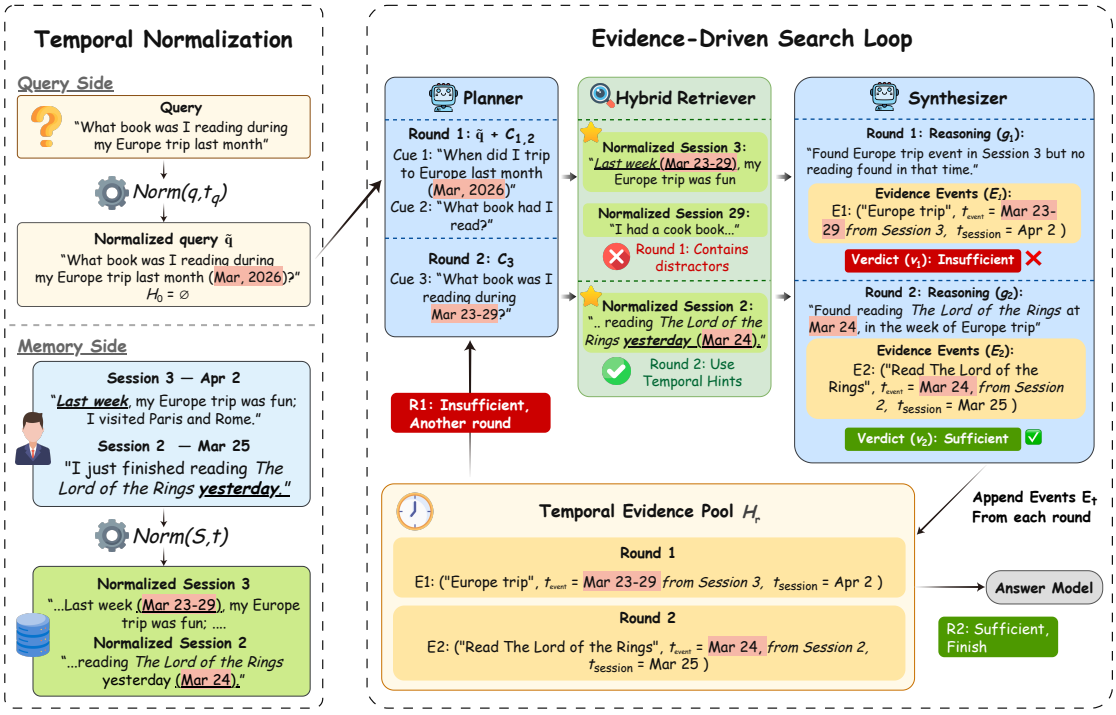

Figure A.1: Full NanoMem architecture for iterative temporal evidence-state construction. The diagram combines temporal normalization, query-time planning, retrieval, synthesis, verdict prediction, and final evidence delivery to a downstream answer model.

A.2 Training and Evaluation Split

The GRPO training data is constructed from three sources. LoCoMo and LongMemEval provide public multi-session dialogue-memory questions with gold answers and evidence-session annotations. The diagnostic TimeMemEval benchmark contributes temporal-bridge records whose final answer evidence is not directly addressable from the original query.
I start from 1,986 LoCoMo questions, 500 LongMemEval questions, and 500 raw TimeMemEval candidates. For LoCoMo, category-5 adversarial questions are excluded. For LongMemEval, single-session preference questions are excluded. For TimeMemEval, the generation pipeline removes invalid attempts, duplicates, split leakage, incomplete metadata, temporal inconsistencies, and non-unique answers. This produces 1,536 LoCoMo questions, 469 LongMemEval questions, and 374 TimeMemEval records in the filtered candidate pool.
The final GRPO training set contains 500 examples. To avoid conversation-level leakage, LoCoMo is partitioned by dialogue rather than by individual question: six conversations form the LoCoMo training pool, and four conversations are reserved for evaluation. LongMemEval is not used for GRPO training; all 469 filtered LongMemEval questions are used for zero-shot evaluation. For TimeMemEval, 99 filtered records are sampled for training, and the remaining 275 records are used for evaluation. The evaluation partitions are not used for checkpoint selection.

Table A.1: Training and evaluation split used for GRPO training and reporting.

Data stage	LoCoMo	LongMemEval	TimeMemEval	Total	Use
Original questions	1,986	500	500	2,986	Source pool
Filtered questions	1,536	469	374	2,379	Candidate pool
Training	401	0	99	500	GRPO training
Evaluation	655	469	275	1,399	Reporting only

Each training example stores lightweight supervision and metadata: the user query, gold answer, gold evidence sessions, source identifier, user or speaker metadata, source bucket, difficulty label, and evaluation mode. It does not store fixed retrieved memories or precomputed rollout traces. During GRPO, the sample metadata is used to call the corresponding online memory-search environment, retrieve sessions, synthesize events, produce verdicts, and score the final answer. The policy is therefore trained on online memory-search behavior rather than on a static retrieved context.

A.3 Model and Baseline Configurations

Table A.2 reports the model configuration used by each method in the main result tables. Every method has a model that produces the final answer. The distinction is whether the memory component returns evidence to a separate downstream answerer, as NanoMem, Mem0, and A-Mem do, or whether the same model directly produces the answer as part of the method, as in full-context prompting and direct-answerer retrieval agents. GPT-5.1 is used as the fixed judge for all methods and is not the answerer unless the row explicitly names GPT-5.1 as the final-answer model.

Table A.2: Model combinations used in the main evaluation tables. The final-answer model is the model whose output is judged for answer accuracy. GPT-5.1 is the fixed judge for all methods, not the answerer for all methods.

Group	Method	Output mode	Memory/search or policy model	Final-answer model
Full context	GPT-5.1	Direct answer	Full history prompt	GPT-5.1
Full context	Qwen3.5-9B	Direct answer	Full history prompt	Qwen3.5-9B
Write-time memory	Mem0	Evidence provider	Qwen3.5-9B	Qwen3.5-9B
Write-time memory	A-Mem	Evidence provider	Qwen3.5-9B	Qwen3.5-9B
Search-time retrieval	MemR^3	Direct answer	Qwen3.5-9B	Qwen3.5-9B
Search-time retrieval	Search-R1	Direct answer	Public checkpoint	Public checkpoint
Temporal memory	Memory-T1-Qwen	Direct answer	Qwen3.5-9B	Qwen3.5-9B
Ours	NanoMem-Vanilla	Evidence provider	Qwen3.5-9B	Qwen3.5-9B
Ours	NanoMem-GRPO	Evidence provider	Qwen3.5-9B GRPO	Qwen3.5-9B

A.4 Training Implementation Details

Policy and rollout. The trainable policy is the Qwen3.5-9B Synthesizer, trained with full-parameter GRPO using VERL/HybridFlow [28]. The Planner, NanoMem retriever, Qwen3.5-9B answerer, GPT-5.1 judge, and reference policy are frozen environment components. For each prompt, the actor samples $K = 8$ trajectories with at most three retrieval rounds. At inference time, NanoMem uses the same maximum of three retrieval rounds and retrieves the top-7 memories in each round. The rollout context allows 49,152 prompt tokens, 2,048 response tokens, and a 65,536-token SGLang model context [42].

Optimization. I optimize with AdamW using learning rate 2×10^{-6}, weight decay 0.01, betas $(0.9, 0.999)$, and Adam epsilon 10^{-8}. The learning-rate schedule is constant with a 3% warmup, corresponding to 9 warmup steps over 300 global steps. Each GRPO update uses one prompt and eight rollout trajectories, with PPO mini-batch size 1 and micro-batch size 1. I use PPO clipping ratio 0.2, actor KL loss coefficient 0.001 with low-variance KL, gradient clipping 1.0, bf16 training, and FSDP parameter and optimizer offload. The final run resumes from global step 170 and continues to step 300, saving checkpoints every 10 steps. Training is conducted on four NVIDIA GH200 96GB GPUs.

Reward setting. The final run uses reward weights

$$(w_o, w_v, w_l) = (0.50, 0.10, 0.05), \qquad c = -0.20.$$

Appendix B.1.2 defines the reward components, and Appendix B.1.3 analyzes the coefficient choice.

A.5 Evaluation Protocol

The main metric is answer accuracy. In all experiments, GPT-5.1 is used as the fixed LLM judge for automatic evaluation, and the judge prompt is kept unchanged

across compared methods. BLEU and token-level F1 are not used as primary metrics because many long-term memory answers are semantically equivalent despite surface variation, especially for dates, event descriptions, and short natural-language answers.
LoCoMo is a two-speaker conversational memory benchmark. The evaluation follows the conversation-memory setting and searches the relevant speaker histories as required by each question. LongMemEval and TimeMemEval are treated as single-user query settings; each query is evaluated independently over its associated history or generated session set.
Token counts are output-side compactness metrics and are not identical to full serving cost. For direct-answerer methods, tokens denote final answer-output tokens. For evidence-provider methods, tokens denote the retrieved evidence or memory context returned downstream. NanoMem-specific diagnostics additionally include retrieval rounds and verdict accuracy. Retrieval rounds measure how often the loop is used, while verdict accuracy measures whether the Synthesizer's `sufficient/insufficient` decision agrees with gold evidence coverage.

A.6 Baseline Implementation Notes

This section records implementation adaptations and caveats not captured by Table A.2. For stateful memory systems, users and benchmark files are isolated to avoid cross-example leakage. Local Qwen baselines use SGLang [42] with thinking disabled. Embedding-based baselines use `text-embedding-3-large` with hidden dimension 3072 unless otherwise noted. The intended evidence retrieval budget is top-$k = 7$, except for method-specific budgets or implementation caveats below.
Full context. Full context does not call a memory system. LoCoMo uses each conversation file as one full-history prefix. LongMemEval uses each query-specific haystack, and TimeMemEval provides all sessions in a sample as context. This is a strong reference baseline but not a deployable memory system.
Mem0. I run a local OSS Mem0 codebase [4] with lightweight server/client adaptations. Histories are ingested at the session level, and Mem0 internally extracts fact-style memory notes; search therefore returns Mem0 notes rather than raw transcripts. The local backend uses pgvector without graph-store retrieval and adds synchronous ingestion, JSON repair and retry, custom fact/update prompts, and top-k search fixes required by the local evaluation harness.
A-Mem. I evaluate A-Mem [35] with the released codebase. Each session is inserted through one `add_note()` call with a timestamped header, so the stored `MemoryNote.content` is the full transcript rather than an extracted atomic fact. The vector store is ChromaDB, and the wrapper keeps the released add/search/evolution path intact. A-Mem evolution updates tags, context, and links, but does not rewrite the stored transcript into compact facts; its evidence token counts therefore reflect transcript-level evidence.
MemR^3. I evaluate MemR^3 [6] with the released codebase and its original retrieve/-generate/reflect/answer workflow. Each benchmark item is converted into MemR^3's conversation-plus-QA format: the conversation is flattened as timestamped speaker text, chunked, embedded, and queried by the multi-step retrieval loop. The con-

figuration uses chunk size 256, top-$k = 7$, maximum 5 iterations, and no reranker. I do not train MemR^3. Disabling the reranker may understate its best retrieval performance, and its score reflects retrieval plus answer generation rather than pure retrieval accuracy.

Search-R1. I evaluate Search-R1 [17] with the public Hugging Face checkpoint `PeterJinGo/SearchR1-nq_hotpotqa_train-qwen2.5-7b-em-ppo`. The model is used as a direct answerer with the original `<think>`/`<search>`/`<answer>` protocol. It can issue up to three search calls, receives retrieved memory text in `<information>` blocks, and the final text inside `<answer>` is used as the benchmark response. The checkpoint was trained for open-domain search QA rather than timestamped long-term conversational memory; the adapter does not fine-tune it or extract atomic memory facts during ingestion.

Memory-T1-Qwen. I do not run an official trained Memory-T1 checkpoint because public weights were not available for the trained model at the time of the experiments. I therefore report a Memory-T1-style baseline using Qwen3.5-9B rather than official Memory-T1 checkpoint performance. Session-level SimpleBM25 retrieves candidate session transcripts; the prompt asks Qwen to select memory IDs and produce a JSON answer. The retrieval budget is top-$k = 7$, the prompt budget is 15,500 tokens, temperature is 0, and malformed JSON outputs receive at most two retries. Session-level BM25 can miss one endpoint of a temporal bridge, and Qwen3.5-9B is not trained for Memory-T1-style long-term temporal dialogue reasoning.

B

Algorithm and Reward Details

B.1 Algorithm Details

B.1.1 NanoMem Training Procedure

This subsection summarizes the online GRPO training loop used to optimize only the Synthesizer/Compressor policy in NanoMem. Planner, preprocessor, retriever, answerer, judge, and reference policy are frozen throughout training.

Algorithm 1 GRPO Training for Multi-turn Memory Evidence Synthesis

Input:

Training set $\mathcal{D}_{\text{train}}$; each sample contains query q, gold answer $a^\star$, gold evidence sessions $\mathcal{E}^\star$, source DB path, user id, and analysis metadata
Trainable compressor policy π_θ initialized from Qwen3.5-9B
Frozen planner, retriever, answerer, judge, and reference policy π_{ref}
Rollout group size $G = 8$, maximum search rounds $R_{\max} = 3$

Ensure:

Optimized compressor policy $\pi_{\theta'}$

for each training step **do**
  Sample one training example $x = (q, a^\star, \mathcal{E}^\star, \mathrm{DB}, u, \mathrm{meta})$ from $\mathcal{D}_{\mathrm{train}}$
  Stage 1: Input and initialization
    Load the sample-specific memory DB and user id; keep planner, retriever, answerer, judge, and π_{ref} frozen
  Stage 2–3: Online multi-turn group rollout
    for rollout $g = 1, \ldots, G$ **do**
      Initialize previous events $\mathcal{M}^{(0)} \leftarrow \emptyset$, previous cues $\mathcal{C}^{(0)} \leftarrow \emptyset$, and terminal flag stop $\leftarrow$ false
      for round $r = 1, \ldots, R_{\max}$ **do**
        Use the frozen planner to generate retrieval cues $\mathcal{C}^{(r)}$ from q, $\mathcal{C}^{(r-1)}$, and compressor reasoning when available
        Use the frozen retriever to search sessions $\mathcal{S}^{(r)}$ from the sample-specific memory DB
        Construct temporal hints online from the query and retrieved sessions
        Sample compressor XML $z_g^{(r)} \sim \pi_\theta(\cdot \mid q, \mathrm{hints}, \mathcal{M}^{(r-1)}, \mathcal{S}^{(r)})$
        Parse $z_g^{(r)}$ into reasoning, events $\mathcal{M}^{(r)}$, and verdict $v^{(r)}$
        if $v^{(r)} =$ sufficient, no new evidence is found, or XML is invalid **then** set stop $\leftarrow$ true and break
      end for
      Use the frozen answerer to generate final answer $\hat{a}_g$ from q and terminal events $\mathcal{M}^{(r)}$
      Store rollout trace τ_g: retrieved sessions, compressor XML, reasoning, events, verdicts, terminal reason, and answer
    end for
  Stage 4: Reward computation
    for rollout $g = 1, \ldots, G$ **do**
      Compute format reward from XML validity and schema compliance
      Compute verdict reward by comparing sufficient/insufficient decisions with gold evidence coverage $\mathcal{E}^\star$
      Use the frozen judge to compare $\hat{a}_g$ with $a^\star$ and assign outcome reward
      Compute length reward from compressor output compactness, mainly active when the outcome is correct
      Combine format, verdict, outcome, and length rewards into scalar reward R_g
    end for
  Stage 5: GRPO policy update
    Compute group-relative advantages from $\{R_g\}_{g=1}^{G}$ for the same query
    Increase likelihood of higher-reward compressor outputs and decrease likelihood of lower-reward outputs
    Apply KL regularization against frozen reference policy π_{ref}
    Update only compressor policy parameters θ
end for
Select the final checkpoint using training reward only; held-out sets are used for reporting
return optimized compressor policy $\pi_{\theta'}$

B.1.2 Reward Design

The reward is defined over a complete rollout trajectory $\tau = \{(h_1, z_1), \ldots, (h_T, z_T)\}$, where h_t is the retrieval context and $z_t = (r_t, \mathcal{E}_t, \hat{v}_t)$ is the Synthesizer action at round t. For coefficient analysis, the same trajectory is abstracted as $\tau = (f, T, \mathbf{v}, o, \ell)$, where f is format validity, T is the number of retrieval rounds, $\mathbf{v}$ is the sequence of verdict correctness signals, o is final answer correctness, and ℓ is evidence compactness.

Format reward. Let $f(\tau) \in \{0, 1\}$ indicate whether all Synthesizer outputs in the rollout are parseable under the required XML schema. Format validity acts as a gate: if $f(\tau) = 0$, all other reward terms are not evaluated and the trajectory receives the fixed reward $c < 0$. Equivalently, the format-failure penalty magnitude is $-c > 0$.

Length reward. Let $m(z_t)$ be the token length of the Synthesizer output at round t. The compactness score $\ell(\tau) \in [0, 1]$ is activated only when the final answer is

correct:

$$\ell(\tau) = \mathbb{I}\{R_{\text{out}}(\tau) = 1\} \cdot \text{clip}\left(1 - \frac{1}{T m_{\max}} \sum_{t=1}^{T} m(z_t), 0, 1\right),$$

where $m_{\max}$ is the length threshold. This term prefers shorter evidence among trajectories that already solve the task.

Verdict reward. The Synthesizer must decide whether the current retrieved evidence covers the query or whether a missing evidence gap remains. Let $\mathcal{G}_q$ denote the gold evidence session set for query q, and let $\mathcal{S}_{\le t}^{\text{ret}}$ denote the session identifiers covered by the accumulated retrieved sessions $\widetilde{\mathcal{S}}_{\le t}$. The round-level verdict reward is

$$v_t = \begin{cases} +1, & \mathcal{G}_q \subseteq \mathcal{S}_{\le t}^{\text{ret}} \text{ and } \hat{v}_t = \texttt{sufficient}, \\ +1, & \mathcal{G}_q \not\subseteq \mathcal{S}_{\le t}^{\text{ret}} \text{ and } \hat{v}_t = \texttt{insufficient}, \\ -1, & \text{otherwise.} \end{cases}$$

This term evaluates whether the policy correctly decides to stop or continue under the objective retrieval state. Since it is computed independently at each round, the trajectory-level verdict score is $V = \sum_{t=1}^{T} v_t$, providing fine-grained process supervision for multi-round search.

Outcome reward. The outcome reward ties the final Temporal Evidence Pool to task success. Let H_{T+1} be the Temporal Evidence Pool after the final round. The frozen answer model generates $\hat{a} = \text{Ans}(q, H_{T+1})$, and the evaluator checks semantic equivalence with the gold answer $a^\star$:

$$o(\tau) = \mathbb{I}\{\text{Eval}(\hat{a}, a^\star) = 1\}.$$

This component measures whether the constructed evidence state is sufficient for answering the original query.

Combining these components, the total reward is

$$R(\tau) = \begin{cases} c, & f(\tau) = 0, \\ w_o o(\tau) + w_v \sum_{t=1}^{T} v_t + w_l \ell(\tau), & f(\tau) = 1, \end{cases}$$

where $w_o, w_v, w_l > 0$ are the weights for outcome, verdict, and compactness. Thus, o and ℓ are trajectory-level signals, while v_t is accumulated over retrieval rounds and format validity gates the whole trajectory. No separate event-time reward is introduced: incorrect event-time reasoning is reflected in $o(\tau)$ through retrieval drift and final answer failure.

B.1.3 Reward Coefficient Analysis

Problem Definition. A rollout trajectory is abstracted as $\tau = (f, T, \mathbf{v}, o, \ell)$, where $f \in \{0, 1\}$ denotes whether the Synthesizer output is parseable under the required XML schema, $T \in \{1, \ldots, T_{\max}\}$ is the number of retrieval rounds, and $\mathbf{v} = (v_1, \ldots, v_T)$ records round-level verdict correctness with $v_t \in \{-1, +1\}$. The final answer correctness is $o \in \{0, 1\}$, and $\ell \in [0, 1]$ measures evidence compactness, with $\ell = 0$ when $o = 0$ because compactness is only rewarded for correct outcomes. Let $V = \sum_{t=1}^{T} v_t$ be the cumulative verdict score. Format acts as a gate: if $f = 0$,

verdict, outcome, and length terms are not evaluated and the rollout receives a fixed penalty c. Otherwise, the reward is:

$$R(\tau) = \begin{cases} c, & f = 0, \\ w_v V + w_o o + w_l \ell, & f = 1, \end{cases}$$

where $c \in \mathbb{R}$ is the format-failure penalty and $w_v, w_o, w_l > 0$ are reward weights for verdict correctness, answer correctness, and compactness, respectively. With $T_{\max} = 3$, the wrong-outcome reward range under format success is $[-3w_v, 3w_v]$, while the correct-outcome reward range is $[-3w_v + w_o, 3w_v + w_o + w_l]$.

The reward is required to satisfy six ordering goals. (G1) *Format gate*: every format-valid trajectory should exceed format failure. (G2) *Outcome dominance*: any correct-outcome trajectory should exceed any wrong-outcome trajectory, regardless of verdict history or compactness. (G3) *Verdict discrimination*: for fixed outcome and compactness, a larger cumulative verdict score should receive larger reward. (G4) *Multi-round credit*: a trajectory with more correct verdict rounds should receive more verdict credit. (G5) *Compactness preference*: for fixed outcome and verdict, greater compactness should receive larger reward. (G6) *Verdict over length*: a single verdict correction should dominate the full compactness range, so process supervision cannot be overridden by the length term.

Four guaranteed margins are also defined to express the intended priority among learning signals. Δ_1 is the *format-gate margin*: it asks how far the worst format-success trajectory remains above format failure. Format failure receives the fixed reward c, while the worst parseable trajectory has all other signals minimized, namely $V = -3$, $o = 0$, and $\ell = 0$:

$$R^{\inf}_{f=1} = w_v(-3) + w_o \cdot 0 + w_l \cdot 0 = -3w_v,$$

$$\Delta_1 = R^{\inf}_{f=1} - c = -3w_v - c.$$

Thus, $\Delta_1 > 0$ guarantees that any parseable output is preferred to a format failure. Δ_2 is the *outcome-discrimination margin*: under format success, it compares the worst correct-answer trajectory with the best wrong-answer trajectory. The former has $V = -3$ and $\ell = 0$, while the latter has $V = 3$:

$$R^{\inf}_{o=1} = -3w_v + w_o, \quad R^{\sup}_{o=0} = 3w_v,$$

$$\Delta_2 = R^{\inf}_{o=1} - R^{\sup}_{o=0} = w_o - 6w_v.$$

The term $6w_v$ is the maximal swing introduced by cumulative verdict scoring; $\Delta_2 > 0$ means that answering correctly remains better than answering incorrectly even under this extreme verdict contrast. Δ_3 is the *single-step verdict signal*: changing one round-level verdict from incorrect to correct changes v_t from -1 to $+1$, increasing V by 2:

$$w_v(V + 2) - w_v V = 2w_v,$$

$$\Delta_3 = 2w_v.$$

This margin is independent of T, o, and ℓ. Finally, Δ_4 is the *compactness range*: since $\ell \in [0, 1]$ and the length term is $w_l \ell$, the maximum reward variation induced by compactness is

$$w_l \cdot 1 - w_l \cdot 0 = w_l,$$

$$\Delta_4 = w_l.$$

These margins satisfy $\Delta_1 > \Delta_2 > \Delta_3 > \Delta_4 > 0$ so that the guaranteed discrimination decreases along the intended priority order: format validity, answer correctness, verdict quality, and evidence compactness. This ordering ensures that GRPO receives policy-gradient signals aligned with the desired learning hierarchy.

Constraint Derivation. Constraints are first derived directly from the six design goals.

C1 (Format gate). The worst format-valid trajectory has $o = 0$ and $V = -T_{\max} = -3$, so Goal 1 requires it to beat the format-failure penalty:

$$-3w_v > c.$$

C2 (Outcome dominance). The worst correct trajectory must beat the best wrong trajectory:

$$R^{\inf}_{o=1} > R^{\sup}_{o=0} \quad \Longrightarrow \quad -3w_v + w_o > 3w_v \quad \Longrightarrow \quad w_o > 6w_v.$$

C3 (Verdict discrimination and multi-round credit). Since the reward is linear in V with slope w_v, verdict ordering and cumulative multi-round credit require

$$w_v > 0.$$

C4 (Compactness preference). Since compactness only affects correct trajectories through $w_l \ell$, compactness preference requires

$$w_l > 0.$$

C5 (Verdict over length). A single verdict correction changes V by 2, so it must dominate the entire compactness range:

$$2w_v > w_l.$$

Gap-Ordering Constraints. Constraints from the desired hierarchy are derived next: $\Delta_1 > \Delta_2 > \Delta_3 > \Delta_4 > 0$.

G1 follows from $\Delta_1 > \Delta_2$:

$$-3w_v - c > w_o - 6w_v \quad \Longrightarrow \quad c < 3w_v - w_o.$$

G2 follows from $\Delta_2 > \Delta_3$:

$$w_o - 6w_v > 2w_v \quad \Longrightarrow \quad w_o > 8w_v.$$

G3 follows from $\Delta_3 > \Delta_4$:

$$2w_v > w_l.$$

G4 follows from $\Delta_4 > 0$:

$$w_l > 0.$$

Combining the six ordering goals with the gap hierarchy gives the feasible coefficient region:

$$\mathcal{W} = \left\{ (c, w_v, w_o, w_l) \in \mathbb{R} \times \mathbb{R}^3_{>0} \;\middle|\; \begin{array}{l} w_v > 0, \\ w_o > 8w_v, \\ c < 3w_v - w_o, \\ 2w_v > w_l \end{array} \right\}.$$

This region keeps outcome correctness as the dominant task-level signal, preserves verdict correctness as process-level supervision, and treats format and length terms as stabilizing constraints rather than primary task objectives.

Coefficient Selection. The coefficients are selected in three steps. First, the design goals are instantiated as a four-level priority structure: answer correctness is the primary objective, verdict correctness provides process-level control over stopping and continued search, compactness improves efficiency, and format validity acts as a hard gate. Second, this priority induces ratio constraints among weights: w_o should dominate w_v, while the single-step verdict signal $2w_v$ should cover the full length range w_l, preventing the policy from sacrificing verdict quality for shorter evidence. Third, the default setting $(w_o, w_v, w_l) = (0.50, 0.10, 0.05)$ is chosen. This makes outcome the dominant signal in the rollout distribution, gives a verdict-step margin $2w_v = 0.20$ larger than the length range $w_l = 0.05$, and sets the format penalty by $c = 3w_v - w_o = -0.20$ so that format failure remains below the worst valid output.

Effectiveness Analysis. This construction can be viewed as an expressibility argument for a desired trajectory ordering. Following the perspective of Abel et al. [1], reward design is not merely assigning heuristic scalar scores: it asks whether a reward function can express a task-level preference relation. The six design goals define a partial order over rollout trajectories for temporal-causal evidence synthesis. For example, outcome dominance requires every correct-answer trajectory to outrank every wrong-answer trajectory, while the verdict and length goals impose finer preferences within trajectory subsets. The feasible region $\mathcal{W}$ then gives a constructive certificate that this ordering is expressible by the multi-level reward family. Moreover, the gap hierarchy $\Delta_1 > \Delta_2 > \Delta_3 > \Delta_4 > 0$ strengthens strict preference by requiring larger guaranteed margins for higher-priority distinctions, echoing the range-based view that acceptable behaviors should be separated from unacceptable ones by reward margins. This explains why the coefficient design is effective: it aligns the scalar GRPO learning signal with the intended trajectory-level preference hierarchy, rather than relying on unstructured reward mixing.

C

Dataset Details

C.1 Dataset Construction

C.1.1 TimeMemEval Benchmark Construction

Algorithm 2 TimeMemEval Benchmark Construction Procedure

Require:
Persona/theme pools, event-family library, bridge scopes $\mathcal{B}$, offset sampler Ω, terminal types $\mathcal{Y}$
Distractor modes $\mathcal{M}$, surface temporal styles $\mathcal{R}$, event/session generators, uniqueness judge $\mathcal{J}$, temporal validator $\mathcal{V}$, retry budgets

Ensure:
TimeMemEval records with `record.json`, `sessions.json`, generation traces, split metadata, and validation logs

function BuildTimeMemEval($\mathcal{B}, \Omega, \mathcal{Y}$)
  for each requested record **do**
    Sample persona/theme context and generate a source event e_s with a valid temporal anchor
    Sample bridge scope and offset (β, δ), where $\beta \in \mathcal{B}$ and $\delta \in \Omega$
    $W^\star \leftarrow$ ComputeTargetWindow(e_s.event_time, β, δ)
    Sample terminal type $y \in \mathcal{Y}$ and a compatible event family; generate destination event e_d inside $W^\star$
    Generate probe question q and answer a that require linking e_s to e_d through $W^\star$
    Generate hard distractors $\mathcal{D}$ using modes in $\mathcal{M}$, including same-activity, answer-alias, near-window, and source-anchor-alias distractors outside $W^\star$
    Generate filler events $\mathcal{U}$ that add dialogue noise without leaking the answer or temporal bridge
    if $\mathcal{J}(e_s, e_d, \mathcal{D}, \mathcal{U}, q, a)$ rejects uniqueness or answerability **then** retry the failed stage
    for each event $e \in \{e_s, e_d\} \cup \mathcal{D} \cup \mathcal{U}$ **do**
      Sample a surface temporal style $r \in \mathcal{R}$ and generate a multi-turn session s_e with relative, indirect, or calendar-based temporal phrasing
      if $\mathcal{V}(s_e, e, \beta)$ rejects temporal consistency **then** regenerate s_e or its dependent event
    end for
    Write record, sessions, traces, split metadata, and validation logs
  end for
end function

TimeMemEval is designed as a controlled benchmark for temporal-bridge retrieval in long-term memory, rather than covering all forms of temporal reasoning. Each instance requires a model to identify a source event, derive a target temporal window, retrieve a destination event in that window, and answer a terminal question about the destination event. This construction intentionally isolates interval-conditioned multi-hop retrieval from ordinary single-hop lookup, while keeping explicit metadata that allows analysis of the effect of bridge scope, offset, terminal type, distractor type, and surface temporal phrasing.

To reduce benchmark artifacts, generation is not tied to a single template. Records are sampled from persona/theme contexts, event-family libraries, terminal-question types, bridge scopes, offset ranges, and multiple distractor modes. Distractors are constructed to be semantically and structurally close to the destination event: for

example, they may share the same activity, preserve the same object or location skeleton when required by the terminal type, occur near the target window, or introduce competing source-like temporal anchors. The final judge enforces that the question is not answerable from the source alone or the destination alone, but remains uniquely answerable when all source, destination, distractor, and filler sessions are present.

Session generation is separated from event generation. After the event-level graph passes the uniqueness gate, each event is realized as a multi-turn memory session with a sampled temporal surface form. These forms include deterministic calendar references, relative references such as "last week" or "three months ago", and less calendar-like user-memory anchors such as life episodes, projects, trips, moves, medical events, habits, or emotionally salient periods. This is intended to make the benchmark less dependent on explicit date strings: the temporal validator checks whether the realized session still supports the intended day, week, or month bucket under the sampled bridge scope.

I report TimeMemEval as a targeted temporal-bridge retrieval stress test. It does not subsume broader temporal phenomena such as long causal chains, belief revision, recurring habits, partial temporal order, or contradictory memories. To make this limitation explicit, benchmark metadata records the temporal topology of every instance, and evaluation can be stratified by bridge scope, terminal type, distractor mode, surface temporal style, and whether temporal normalization is provided to the model. This allows distinction between performance gains from temporal normalization, retrieval policy, and reasoning over the generated memory evidence, and supports held-out splits over event families, surface templates, and temporal relation types.

C.1.2 Training and Held-out Evaluation Dataset Construction

I construct the GRPO data from two public long-term memory benchmarks, LoCoMo and LongMemEval, together with the generated TimeMemEval records. LoCoMo and LongMemEval provide real multi-session dialogue questions with gold answers and evidence-session annotations, while TimeMemEval contributes targeted temporal-causal questions whose answer evidence is not directly addressable from the original query. These sources play different roles: the public benchmarks preserve general long-memory coverage, whereas TimeMemEval increases the density of training and evaluation cases that require temporal bridge construction.

Source filtering. I start from 1,986 LoCoMo questions, 500 LongMemEval questions, and 500 raw TimeMemEval candidates. For LoCoMo, category-5 adversarial questions are excluded; for LongMemEval, single-session preference questions are excluded, yielding 1,536 and 469 usable public questions, respectively. For TimeMemEval, the generation pipeline first produces 500 raw candidates and then applies automatic validity checks for invalid generation attempts, duplicate examples, split leakage, incomplete metadata, temporal inconsistency among target windows, session times and event times, and answer non-uniqueness. This leaves 374 filtered TimeMemEval records that are eligible for training or held-out evaluation.

The resulting filtered pool contains 2,379 examples.

Training and held-out evaluation splits. The final GRPO training set contains 500 examples. To avoid conversation-level leakage in LoCoMo, I first partition LoCoMo by dialogue rather than by question: six conversations form the LoCoMo training pool and the remaining four conversations are reserved for held-out evaluation. I then sample 401 LoCoMo questions from the six training conversations. LongMemEval is not used for GRPO training; all 469 filtered LongMemEval questions are kept for zero-shot held-out evaluation. For TimeMemEval, I sample 99 training examples from the 374 filtered records and use the remaining 275 filtered records for held-out evaluation. The held-out evaluation sets are not used for checkpoint selection.

Example format. Each training example stores only lightweight supervision and metadata: the user query, gold answer, gold evidence sessions, source identifier, user or speaker metadata, source bucket, difficulty label, and evaluation mode. It does not store fixed retrieved memories or precomputed rollout traces. During GRPO, the system uses the sample metadata to dynamically call the corresponding NanoMem search environment, generate temporal hints, retrieve sessions, synthesize events, produce verdicts, and score the final answer. This ensures that the policy is trained on online memory-search behavior rather than a static retrieved context.

Data stage	LoCoMo	LongMemEval	TME	Total	Use
Original questions	1,986	500	500	2,986	Source pool
Filtered questions	1,536	469	374	2,379	Candidate pool
Train	401	0	99	500	GRPO training
Held-out evaluation	655	469	275	1,399	Reporting only

Table C.1: Distribution of query pools for GRPO training and held-out evaluation. LoCoMo is split by conversation; LME is zero-shot; TME uses filtered records.

D

Prompt Templates and Case Study

D.1 Prompt Templates

D.1.1 Prompt Template of Initial Planner

Prompt Template of Initial Planner: $P_{\text{plan}}^{(0)}$

```
You are a memory retrieval planner. Given a user query, decompose it into
targeted retrieval cues for searching the user's conversation history.

## Output format (XML only)

<cues>
  <cue>
    <text>retrieval query text</text>
  </cue>
</cues>

---

## Query

{query}

---

## Rules

Cue text:
- Decompose the query into one or more focused retrieval cues.
- Each cue should target a distinct aspect or entity of the query.
- Always generate at least one `<cue>`. If not sure what to generate,
  copy the original user query.
- Preserve named entities, dates, locations, event types, quantities, and
  temporal semantics.
- Preserve the user's intent as closely as possible. Do not broaden a
  specific question into a generic summary query.
- For questions asking "when", keep the cue as a "when" query about the
  same event.
- For dependent temporal questions, emit the original standalone cue plus
  support cues for the anchor event and target fact.
- For count, comparison, ordering, or duration questions, emit support cues
  that retrieve the underlying items, quantities, event times, or start/end
  anchors.
- Keep the original standalone intent as the first cue, then add supporting
  cues.
- Do not annotate temporal phrases. Temporal extraction is handled by
  deterministic preprocessing after cue generation.

Return XML only, no markdown fences, no text outside the tags.
```

Figure D.1: Prompt template used by the initial planner to decompose the user query into targeted retrieval cues.

D.1.2 Prompt Template of Planner Retry

Prompt Template of Planner Retry: $P_{\text{plan}}^{(\text{retry})}$

```
You are a memory retrieval planner. A previous retrieval round was
attempted but the synthesizer judged the evidence insufficient.
Your task is to generate new targeted cues that address the identified gap.

## Output format (XML only)

<cues>
  <cue>
    <text>retrieval query text</text>
  </cue>
</cues>

---

## Original query

{query}

## Previous cues

{previous_cues}

## Temporal Evidence Pool
{pool}

## Synthesizer reasoning from previous round

{previous_reason}

---

## Rules

Cue text:
- Generate 1-3 cues that target the missing evidence gap named in the
  synthesizer reasoning and its extracted temporal evidences.
- If reasoning found an anchor/bridge entity/place/time, reuse that entity to
  guide the new cue generation.
- Combine the anchor/bridge with the missing target question relation, e.g.
  added beside, requested from, mailed for, set up through, etc.
- Do not re-search only for the anchor/source if it was already found in reason;
  search for the downstream action/object involving the anchor.
- Turn "no evidence of X near/from/through Y" into a cue for "X
  near/from/through Y". It is your job to recover missing relations through new
  cues.
- Do not repeat previous cues unless the reasoning explicitly asks for
  the same topic with a different temporal anchor or framing.
- Preserve the user's intent as closely as possible. Do not broaden a
  specific question into a generic summary query.
- Keep cues short and search-like; avoid meta questions such as
  "is this linked" or "is this the same item".
- For questions asking "when", keep the cue as a "when" query about the
  same event.
- For count, comparison, ordering, or duration questions, emit support cues
  that retrieve the underlying items, quantities, event times, or start/end
  anchors.
- Do not annotate temporal phrases. Temporal extraction is handled by
  deterministic preprocessing after cue generation.

Return XML only, no markdown fences, no text outside the tags.
```

Figure D.2: Prompt template used by the retry planner to generate new cues from the previous Synthesizer reasoning.

D.1.3 Prompt Template of Synthesizer

Prompt Template of Synthesizer: P_{synth}

```
You are a memory evidence synthesizer. Given a user query, temporal hints,
previously extracted events, and retrieved conversation sessions, extract
structured evidence and judge whether retrieval is sufficient to answer the
query.

## Output format (XML only, this exact order)

<reasoning>
Concise verdict justification (<=200 words): what decisive evidence is
present, what is missing if any, and why the verdict follows.

</reasoning>

<events>
  <event
    session_id="source session ID; exactly one session per event"
    session_time="YYYY-MM-DDTHH:MM:SS.sssZ"
    event_time="YYYY-MM-DD or 'unknown'">
    A single self-contained evidence statement. Free text, one paragraph,
    no nested tags, no markdown.
  </event>
</events>

<verdict>sufficient|insufficient</verdict>

---

## Normalized Query

{query}

## Retrieved sessions

{sessions}

Retrieved session text preserves original temporal phrases and may annotate
them with parenthesized normalized times, such as `last night (during
2023-05-27 18:00:00 to 2023-05-28 05:59:59)`. These parenthesized values are
deterministic parser aids for interpreting time; they are not additional facts
or evidence beyond the session text.

## Temporal Evidence Pool

{previous_events}

Temporal Evidence Pool are evidence already extracted in earlier retrieval rounds.
They may summarize facts that also appear in the current retrieved sessions.
Use them as context for sufficiency and deduplication, but do not treat them
as new evidence or a substitute for facts grounded in retrieved sessions.
---

## Rules

`Verdict`:
`insufficient` - critical evidence clearly absent; loop continues.
`sufficient` - events reliably cover what the query needs; loop ends.

`event_time`: resolve relative expressions using session timestamps and
temporal hints. Each temporal hint contains the original query phrase and an
absolute UTC timestamp range in `iso_span`. If no explicit date is
recoverable, fall back to the session date. Use `unknown` only if even the
session date is unavailable.

Output rules:
- The final `<events>` output is cumulative. Include all useful
  `<previous_events>` first, unchanged when still valid, then append newly
  extracted events from the current retrieved sessions. Do not delete or
  rewrite previous events unless you find an error that can be corrected by
  current sessions.
- One `<event>` per distinct evidence item, grounded in exactly one session
  (`session_id` must be a single session ID from the input; one session may
  yield multiple events). Always emit partial events even when the verdict is
  `insufficient`. Use `<events/>` only if no useful evidence exists at all.
- If a fact is already covered by previous events and the current sessions add
  no new information, do not repeat it as a new event.
- Preserve temporal relations (before/after/during, duration anchors).
- Event text: free text, one paragraph, no nested tags, no markdown.
- Event text should avoid including temporal phrases when possible; represent
```

```
  time in the `event_time` attribute instead.
- Return XML only, no text outside the tags.
- All three attributes (`session_id`, `session_time`, `event_time`) are
  required on every `<event>`.
- For quantity, state, or collection queries, retain all update events
  in chronological order; do not drop later increments or corrections.
```

Figure D.3: Prompt template used by the Synthesizer to extract temporal evidence and decide whether retrieval is sufficient.

D.1.4 Case Study: Temporal Bridge Retrieval

Case Study: Temporal Bridge Retrieval

Case overview. This case illustrates how NanoMem recovers an answer when the decisive temporal anchor is not stated in the original query but must be inferred from a bridge session retrieved in an earlier round.

Query. What did Haruki read 4 months after sorting through old notebooks and setting aside pages that could become personal essays?

Round 1. The initial planner issues cues for the original question, the notebook-sorting event, and Haruki's reading activities. Retrieval returns several reading distractors from February, April, May, September, November, and December, together with the bridge session `S_E001`. The key bridge evidence is: *Haruki: I spent most of the afternoon on the living room floor with old notebooks and binders, and 2 months ago I made a separate pile of pages that could become personal essays instead of project records.*

Since `S_E001` is timestamped 2024-04-18, the normalized phrase "2 months ago" grounds the sorting event in February 2024. The Synthesizer then infers that four months after February 2024 is June 2024. However, because the retrieved sessions contain no June reading evidence, it returns `insufficient`.

Retry cues.

```
What did Haruki read in June 2024 (during 2024-06)?
What books did Haruki read between May 2024 and September 2024?
What reading activities did Haruki have in June 2024 (during 2024-06)?
```

Round 2. The second retrieval round finds the target session `S_H2_E001_01`, timestamped 2024-06-08: *Haruki: I read The Death and Life of Great American Cities this month, and it got me thinking about how to turn my old engineering observations into personal essays instead of reports.*

Final synthesis. The final Synthesizer output carries forward the bridge event, appends the June reading event, and marks the evidence as `sufficient`. The final answer is *The Death and Life of Great American Cities*, matching the ground-truth label. This example highlights the role of Synthesizer reasoning as the information bridge between rounds: it converts first-round bridge evidence into a concrete temporal cue that the retry planner can use to retrieve the otherwise hidden answer session.

Figure D.4: Case study showing how first-round bridge evidence is transformed into a second-round temporal retrieval cue.

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY