ForeDreamer: A Self-Evolving Dual-Agent Memory Architecture for Future Event Prediction

Linhao Zhong[1,2*] **Zongze Du**[1*] **Linyu Wu**[3] **Yu Bo**[1] **Hourong Li**[1]
Chenchen Jing[1,4] **Hao Chen**[1] **Yuling Xi**[1†] **Chunhua Shen**[1,2,4†]

[1]Zhejiang University, State Lab of CAD & CG [2]Ant Group
[3]National University of Singapore [4]Zhejiang University of Technology

Abstract

Open-web future event prediction requires agents to distill reliable signals from noisy, redundant, and incomplete evidence. Existing retrieval/memory mechanisms directly feed retrieved information to agents or rely on simple memory functions such as storing and reusing prior information for prediction, leaving them insufficient for open-web forecasting. We propose to transform raw web evidence into structured memory before prediction, enabling agents to reason over distilled, question-specific evidence rather than noisy retrieval results. This paper presents ForeDreamer, a self-evolving dual-agent framework for managing memory over open-web evidence. ForeDreamer separates factual memory, a question-specific evidence state for the current forecast, from experiential memory, persistent agent experience accumulated across forecasting episodes. It uses a main agent for search and prediction, and a memory-processing subagent to convert search results into factual memory with dedicated tools. ForeDreamer further evolves experiential memory through two tracks, improving both forecasting decisions and factual-memory construction. Experiments on Prophet Arena and FutureX demonstrate the effectiveness of ForeDreamer. **Project page:** `https://zhongzero.github.io/ForeDreamer`

1 Introduction

Large language model (LLM) (Vaswani et al., 2017; Brown et al., 2020; OpenAI, 2024; Yang et al., 2025a) agents are increasingly deployed in open-world settings that require tool use, web interaction, and adaptation across tasks. A central requirement in these settings is agent memory (Chhikara et al., 2025; Kang et al., 2025), which maintains task-relevant information that supports an agent's behavior within and across tasks. Future event prediction (Yang et al., 2025b; Zeng et al., 2025) places particular demands on agent memory because an agent must identify relevant open-web evidence, organize noisy and redundant observations into a coherent evidence basis, and produce a calibrated prediction from incomplete and uncertain signals.

Existing retrieval and agent-memory mechanisms (Chhikara et al., 2025; Kang et al., 2025; Fang et al., 2025) are not sufficient for this setting. In web-augmented forecasting, agents can access search results, but existing approaches often focus on how these results are supplied to the model. A simple long-context strategy largely passes retrieved information directly into the model context, whereas retrieval-augmented generation typically performs lightweight selection or aggregation before generation. Both strategies can remain unreliable when relevant signals are sparse, time-sensitive, and mixed with distracting or conflicting reports. Long-term agent-memory systems, by contrast, are often designed for dialogue personalization, static knowledge reuse, or accumulated user-specific context. Such systems usually emphasize relatively simple memory functions, including the storage, retrieval, or reuse of prior information, and are therefore less suited to the more complex setting of open-web future event prediction. Figure 1 illustrates this task mismatch. Existing memory mechanisms can support relatively clean, user-centered memory tasks (Maharana et al., 2024; Wu et al., 2024), but they do not directly address the need to process noisy and conflicting open-web evidence.

In this paper, we distinguish between two roles of agent memory in open-ended forecasting. Factual memory denotes a query-conditioned evidence artifact constructed from retrieved web observations for the current forecasting question. It serves as the processed evidence interface for the current forecast, with its scope restricted to the current

*Equal Contribution.
†Corresponding Authors.

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 3085–3112
October 24-29, 2026 ©2026 Association for Computational Linguistics

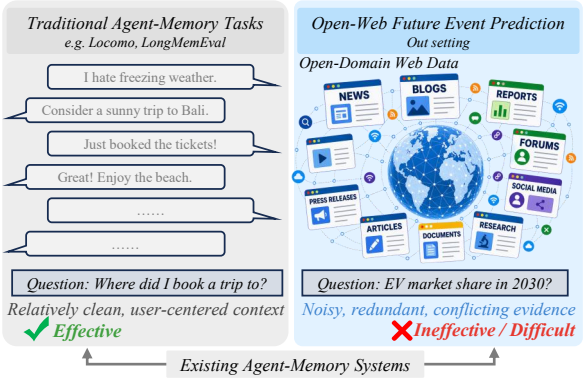


Figure 1: Task mismatch of existing agent-memory systems. Traditional memory tasks often involve relatively clean, user-centered context and can be addressed by storing, retrieving, or reusing prior information. Open-web future event prediction instead requires agents to handle heterogeneous, noisy, and conflicting web data, which existing memory mechanisms are insufficient to process into forecast-ready evidence.

question. Experiential memory denotes persistent agent memory that accumulates across forecasting episodes and guides future behavior. In open-web forecasting, experiential memory should support both forecasting decisions, including search planning and forecast calibration, and evidence-processing decisions that determine how noisy search artifacts are transformed into factual memory.

We aim to develop an agent-memory framework for open-ended future event prediction by addressing the following two research questions (RQs):

- **RQ1.** *How can an agent make reliable future event predictions from noisy open-web evidence?*
- **RQ2.** *How can feedback from forecasting episodes be used to evolve experiential memory that improves both factual-memory construction and forecasting decisions?*

Contribution 1. To answer RQ1, we formalize factual-memory management for open-ended future event prediction and propose **ForeDreamer**, a self-evolving dual-agent framework for managing factual memory over open-web evidence. ForeDreamer separates forecasting from evidence processing. The main agent searches for relevant evidence, integrates processed evidence, and produces the final forecast. A memory-processing subagent receives search results and constructs question-specific factual memory using a MemGuide to specify the evidence-processing workflow and executable MemTools to perform evidence-processing operations. This design exposes factual memory as the question-specific interface between raw evidence and the final forecast.

Contribution 2. To answer RQ2, we develop a dual-track evolution mechanism for experiential memory in ForeDreamer. One track updates textual forecasting experience that guides search planning, evidence integration, and forecast calibration. The other track updates procedural evidence-processing experience that guides how search results are converted into factual memory. We further analyze limitations of the initial evolution process, including redundant tool generation and over-concentrated exploration, and introduce two optimizations, compositional tool reuse and diversity-guided exploration scheduling. Experiments on Prophet Arena (Yang et al., 2025b) and FutureX (Zeng et al., 2025) demonstrate the effectiveness of ForeDreamer for open-web future event prediction.

2 ForeDreamer Framework

ForeDreamer is a self-evolving agent-memory framework for open-web future event prediction. Given a forecasting question, it retrieves temporally valid web evidence, converts raw search results into question-specific factual memory, and predicts from the processed evidence. It separates factual memory, the local evidence interface for the current question, from experiential memory, persistent guidance accumulated across forecasting episodes. Figure 2 summarizes the inference workflow and dual-track self-evolution.

2.1 Task Formulation

Each instance consists of a forecasting question q and a temporal cutoff τ_q specifying the information available at prediction time. The system outputs a forecast $\hat{y}$, either a probability distribution over listed outcomes or a discrete answer, depending on the benchmark. We denote the cutoff-constrained evidence space by $\mathcal{E}_{\leq \tau_q}$. Since open-web results are long, noisy, and only partially relevant, ForeDreamer inserts factual-memory construction between retrieval and forecasting.

2.2 Architecture Overview

As shown in Figure 2, ForeDreamer contains two agents. The main agent receives the question, consults the Experience Bank $\mathcal{B} = \{e_i\}_{i=1}^{|\mathcal{B}|}$,

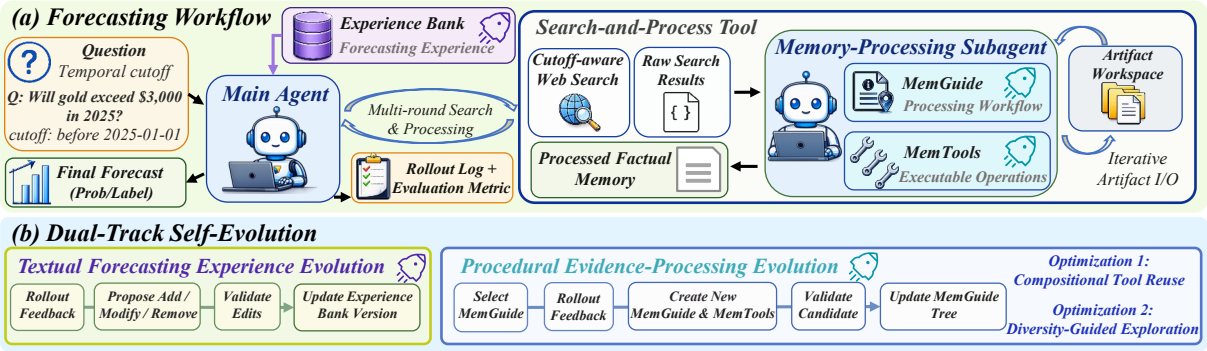


Figure 2: Overview of ForeDreamer. (a) During forecasting, the main agent uses forecasting experience from the Experience Bank, invokes a search-and-process tool to collect temporally valid evidence, and receives processed factual memory from a memory-processing subagent. The subagent follows a MemGuide and uses executable MemTools through an artifact workspace. (b) Rollout feedback drives dual-track self-evolution: textual forecasting experience evolves through validated edits to the Experience Bank, while procedural evidence-processing experience evolves through validated updates to the MemGuide tree and MemTools.

plans search actions, integrates factual memory, and produces the final prediction, where each e_i is a reusable textual forecasting experience. The memory-processing subagent converts raw search-result artifacts into factual memory under a MemGuide g and its associated MemTool set $\mathcal{T}_g$: the MemGuide specifies the evidence-processing workflow, and MemTools provide executable operations. This makes evidence processing explicit and inspectable, rather than leaving the forecasting model to interpret raw search results from a long prompt. Appendix B.3 provides additional details on the MemGuide–MemTool interface.

2.3 Forecasting Workflow

Panel (a) of Figure 2 illustrates the forecasting workflow. Given (q, τ_q), the main agent conditions on $\mathcal{B}$ and runs a multi-turn search-and-reasoning loop with budget T. At turn t, it may issue one query s_t through the search-and-process tool, and cutoff-aware web search retrieves

$$\mathcal{R}_t = \mathrm{Search}(s_t, \mathcal{E}_{\leq \tau_q}). \tag{1}$$

Each result $r_{t,k} \in \mathcal{R}_t$ is normalized and written to an artifact workspace. The memory-processing subagent then applies the selected MemGuide and MemTools to produce factual memory,

$$m_{t,k} = P_{g,\mathcal{T}_g}(r_{t,k}, q), \tag{2}$$

where $P_{g,\mathcal{T}_g}$ denotes guide-conditioned processing with executable tools. The subagent may iteratively read and write artifacts, and return $\mathcal{M}_t = \{m_{t,k} : r_{t,k} \in \mathcal{R}_t\}$ to the main agent.

Let K be the number of executed search-and-processing turns, with accumulated factual memory $\mathcal{M}_{1:K} = \bigcup_{t=1}^{K} \mathcal{M}_t$. On the final allowed turn, the main agent answers without calling the search tool:

$$\hat{y}, \rho = A(q, \mathcal{B}, \mathcal{M}_{1:K}), \tag{3}$$

where A is the main agent and ρ is the rationale. ForeDreamer records the rollout, including interactions, artifacts, ground truth, and metrics, as feedback for self-evolution. Additional details on search-and-process decomposition, factual-memory representation, and controlled tool execution are provided in Appendix B.1, Appendix B.2, and Appendix B.7, respectively.

2.4 Dual-Track Self-Evolution Interfaces

Panel (b) of Figure 2 shows how rollout feedback updates experiential memory through two tracks. The textual track applies validated add, modify, or remove operations to the Experience Bank, improving guidance for search planning, evidence integration, and calibration. The procedural track critiques rollout behavior, derives design requirements, generates candidate MemGuides and MemTools, and admits validated candidates into the guide tree. The next section details these algorithms and the two procedural optimizations: compositional tool reuse and diversity-guided exploration.

3 Dual-Track Experience Evolution

ForeDreamer evolves experiential memory on a small evolution-and-validation pool $\mathcal{D}_{\text{evo}}$, whose rollouts provide feedback for memory updates and

serve as the admission gate for candidates. Evolution has two tracks: textual forecasting experience for search planning, evidence integration, and calibration, and procedural evidence-processing experience for converting search results into factual memory. This section describes both tracks, with procedural evolution summarized in Figure 3.

3.1 Textual Forecasting Experience Evolution

Textual forecasting experience is stored as a compact experience bank $\mathcal{B}^{(v)} = \{e_i\}_{i=1}^{n_v}$ at version v. In each textual-evolution attempt, ForeDreamer selects the currently best validated MemGuide, samples an example from $\mathcal{D}_{\text{evo}}$, runs a rollout with the current bank, and asks an editor to propose $o \in \{\text{ADD}, \text{MODIFY}, \text{REMOVE}\}$, yielding $\widetilde{\mathcal{B}} = o(\mathcal{B}^{(v)})$. ADD appends an entry, MODIFY revises one, and REMOVE deletes an ineffective entry. The candidate bank is evaluated on $\mathcal{D}_{\text{evo}}$ under the selected MemGuide and accepted only when validation improves:

$$\mathcal{B}^{(v+1)} = \begin{cases} \widetilde{\mathcal{B}}, & \widetilde{\mathcal{B}} \succ \mathcal{B}^{(v)}, \\ \mathcal{B}^{(v)}, & \text{otherwise}, \end{cases} \tag{4}$$

where $\widetilde{\mathcal{B}} \succ \mathcal{B}^{(v)}$ means better validation score.

3.2 Procedural Evidence-Processing Evolution

Procedural evidence-processing experience consists of MemGuides, which specify subagent workflows, and MemTools, which provide executable evidence-processing operations. Unlike textual experience, procedural experience must evolve both instructions and tools, and generated assets are accepted only after interface checks and validation. Figure 3 shows vanilla rollout-based expansion and two optimizations.

3.2.1 Vanilla Rollout-Based Expansion

The vanilla mechanism represents procedural experience as a MemGuide tree $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each node $g_i \in \mathcal{V}$ is a MemGuide and each edge records provenance. In one expansion attempt, ForeDreamer selects a parent MemGuide, samples an example from $\mathcal{D}_{\text{evo}}$, runs a rollout, and uses a critic to derive a design requirement d when improvement is needed. With validation results, parent selection follows validation-rank Zipf sampling: if g_i has rank r_i among N valid nodes, it is selected with probability

$$p(g_i) = \frac{1/r_i}{\sum_{j=1}^{N} 1/r_j}. \tag{5}$$

This favors stronger MemGuides while preserving nonzero probability for lower-ranked nodes. ForeDreamer then generates $(\widetilde{g}, \widetilde{\mathcal{T}})$ from d. If $\text{Valid}(\widetilde{g}; \mathcal{D}_{\text{evo}}) = 1$, the candidate is admitted as a child and the tree is updated as

$$\mathcal{V} \leftarrow \mathcal{V} \cup \{\widetilde{g}\}, \quad \mathcal{E} \leftarrow \mathcal{E} \cup \{(g_i, \widetilde{g})\}. \tag{6}$$

Validity requires successful validation runs on all examples in $\mathcal{D}_{\text{evo}}$.

The tree provides provenance but can cause redundant MemTool generation and over-concentration around locally successful guide families. ForeDreamer addresses these limitations through compositional tool reuse and diversity-guided exploration.

3.2.2 Compositional Tool Reuse

Observation. Vanilla rollout-based procedural evolution often regenerates MemTools with overlapping logic. In the vanilla FutureX setting, we cluster generated MemTools by pairwise source-code similarity. As shown in Figure 4, at token-Jaccard threshold $\tau = 0.5$, 201 generated MemTools collapse into 98 clusters, yielding a cluster ratio of 48.8%. This suggests recurring implementation patterns and wasted evolution budget on near-duplicate operations. Additional similarity views are provided in Appendix E.1.

Method. ForeDreamer therefore introduces compositional tool reuse, illustrated in Figure 3(b). Given design requirement d, ForeDreamer summarizes existing valid MemTools and selects a compatible subset from the active set, $\mathcal{T}_{\text{reuse}} = R(d, \mathcal{T}_{\text{active}})$ with $\mathcal{T}_{\text{reuse}} \subseteq \mathcal{T}_{\text{active}}$. The candidate MemGuide is newly generated, but it reuses selected tools; new MemTools $\mathcal{T}_{\text{new}}$ are generated only for uncovered operations, yielding $\widetilde{\mathcal{T}} = \mathcal{T}_{\text{reuse}} \cup \mathcal{T}_{\text{new}}$. The candidate returns to the same validation gate, reducing duplicate tool creation while retaining the ability to introduce new tools when needed.

3.2.3 Diversity-Guided Exploration

Observation. The vanilla MemGuide tree can also constrain exploration. Since each expansion

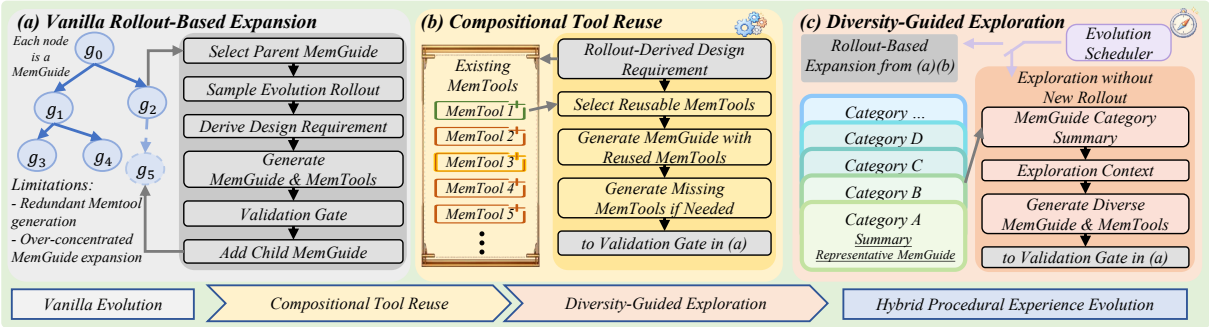


Figure 3: Evolution of procedural evidence-processing experience. (a) Vanilla rollout-based expansion grows a MemGuide tree by selecting a parent MemGuide, sampling an evolution rollout, deriving a design requirement, generating a candidate MemGuide and MemTools, and admitting the candidate through a validation gate. (b) Compositional tool reuse optimizes the candidate-generation step by selecting compatible existing MemTools before generating the MemGuide and any missing MemTools. (c) Diversity-guided exploration uses an evolution scheduler to retain the rollout-based path while adding an exploration path that generates category-diverse candidates from MemGuide-category summaries and representative MemGuides without requiring a new rollout result.

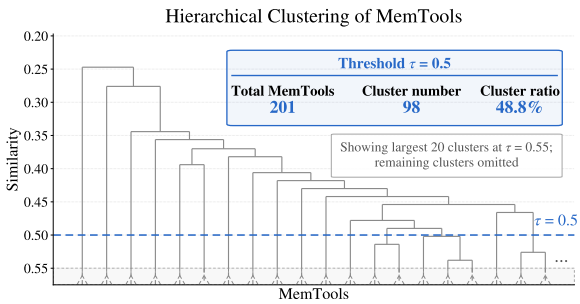


Figure 4: Hierarchical clustering of MemTools generated by vanilla procedural evolution on FutureX, using token-Jaccard source-code similarity. The reduced cluster count indicates that many generated MemTools share substantial token-level overlap, suggesting that vanilla evolution spends many attempts on near-duplicate operations.

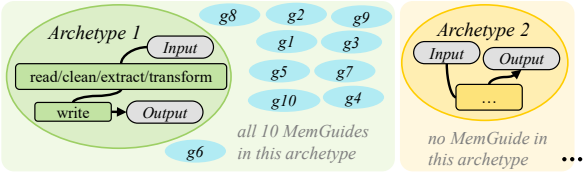


Figure 5: Pipeline archetype distribution for vanilla rollout-based expansion on FutureX. All 10 evolved MemGuides fall into the same tool-orchestrated processing archetype, indicating limited exploration of alternative evidence-processing pipelines.

conditions on a parent guide and a local rollout-derived requirement, successful branches may attract nearby variants while other strategies receive fewer trials. On 10 FutureX MemGuides evolved with vanilla expansion, all fall into the same tool-orchestrated pipeline archetype (Figure 5), indicating refinement of one guide family rather than exploration of alternative evidence-processing pipelines.

Method. Figure 3(c) illustrates diversity-guided exploration. ForeDreamer augments rollout-based expansion with an exploration path that does not require a new rollout result, and a scheduler alternates between exploration and rollout-based expansion. For exploration, ForeDreamer summarizes current MemGuide categories, identifies representative MemGuides, and generates candidates that are not merely local variants of existing branches. Accepted candidates are stored in the same tree, with provenance pointing to the representative guides used as context, and pass through the same validation gate as rollout-derived candidates.

Together, compositional tool reuse and diversity-guided exploration form a hybrid procedural state with MemGuide-tree provenance, validated reusable MemTools, MemGuide-category summaries, representative MemGuides, and validation records. This preserves tree interpretability while moving procedural evolution beyond local parent-child expansion. Additional details on MemGuide and MemTool validation, procedural summary records, and validation-based candidate selection are provided in Appendix B.4, Appendix B.5, and Appendix B.6, respectively.

4 Experiments

4.1 Experimental Setup

We evaluate ForeDreamer on open-ended future event prediction using Qwen3.5-Flash (Qwen

Model	Method	Climate/Weather	Companies	Economics	Entertainment	Mentions	Other	Politics	Sports	Avg. (↓)
Qwen3.5-Flash	Full Text	0.3342	0.2657	0.0720	0.2005	0.2106	0.1306	0.2214	0.2122	0.2059
	RAG	0.2511	0.2319	0.0752	0.1739	0.2178	0.1381	0.2094	0.2030	0.1876
	HippoRAG 2	0.2798	0.2553	0.0524	0.1318	0.2020	0.1423	0.2069	0.1945	0.1831
	Mem0	0.2792	0.2902	0.0695	0.1936	0.2308	0.1778	0.2423	0.2045	0.2110
	MemoryOS	0.2385	0.2348	0.0542	0.1155	0.1917	0.1676	0.2216	0.1925	0.1771
	A-MEM	0.2306	0.2591	0.0602	0.1516	0.2218	0.1542	0.2122	0.1996	0.1862
	LightMem	0.2562	0.2382	0.0709	0.1253	0.1783	0.1455	0.1971	0.1971	0.1761
	LangMem	0.2611	0.2516	0.0658	0.1500	0.2190	0.1427	0.2017	0.2024	0.1868
	ForeDreamer	**0.1770**	**0.2162**	**0.0414**	**0.0962**	**0.1573**	**0.1274**	**0.1742**	**0.1869**	**0.1471**
GPT-5.4-Nano	Full Text	0.2275	0.3719	0.0631	0.1424	0.2878	0.1410	0.2257	0.2075	0.2084
	RAG	0.1856	0.3698	0.0596	0.1377	0.2768	0.1449	0.2216	0.2014	0.1997
	HippoRAG 2	0.2338	0.4052	0.1216	0.1376	0.2806	0.1515	0.2339	0.2031	0.2209
	Mem0	0.2404	0.3941	0.0582	0.2138	0.3021	0.2009	0.2582	**0.1782**	0.2307
	MemoryOS	0.2039	0.3578	0.1191	0.1455	0.2810	0.1551	0.2026	0.1934	0.2073
	A-MEM	0.1980	0.4155	0.1248	0.1782	0.2817	0.1291	0.2273	0.1939	0.2186
	LightMem	**0.1584**	0.4226	0.1172	0.1609	0.2799	0.1439	0.2117	0.2041	0.2123
	LangMem	0.1968	0.3654	0.0754	0.1805	0.2844	0.1656	0.2410	0.2076	0.2146
	ForeDreamer	0.1811	**0.3456**	**0.0506**	**0.1117**	**0.2704**	**0.1272**	**0.1927**	0.1919	**0.1839**

Table 1: Main comparison on Prophet Arena. The table reports Brier score for each category and the average score; lower values indicate better calibrated forecasts.

Model	Method	Accuracy (↑)
Qwen3.5-Flash	Full Text	0.3298
	RAG	0.3382
	HippoRAG 2	0.3285
	Mem0	0.3264
	MemoryOS	0.3269
	A-MEM	0.3495
	LightMem	0.2885
	LangMem	0.3333
	ForeDreamer	**0.4108**
GPT-5.4-Nano	Full Text	0.2766
	RAG	0.3221
	HippoRAG 2	0.3173
	Mem0	0.3567
	MemoryOS	0.3262
	A-MEM	0.3480
	LightMem	0.2788
	LangMem	0.3333
	ForeDreamer	**0.3883**

Table 2: Main comparison on FutureX. The table reports prediction accuracy; higher values are better.

Team, 2026) and GPT-5.4-Nano (OpenAI, 2026) as backbone models. Evaluations are conducted on Prophet Arena (Yang et al., 2025b) and FutureX (Zeng et al., 2025), with temporally constrained web search used for evidence collection. Prophet Arena is evaluated with Brier score, while FutureX is evaluated with accuracy. Baselines include the Full Text setting, a generic RAG baseline, and previously proposed agent-memory systems, including HippoRAG 2 (Gutierrez et al., 2025), Mem0 (Chhikara et al., 2025), MemoryOS (Kang et al., 2025), A-MEM (Xu et al., 2026), LightMem (Fang et al., 2025), and LangMem (The LangChain Team, 2025). More implementation details and benchmark descriptions are provided in Appendix C, and the prompt templates used in our implementation are provided in Appendix F.

Method	Prophet Arena Avg. (↓)	FutureX (↑)
Full Text	0.2059	0.3298
w/o evolving MemGuide&MemTool	0.1663	0.3892
w/o evolving Experience Bank	0.1769	0.3351
ForeDreamer	**0.1471**	**0.4108**

Table 3: Dual-track experience ablation on Qwen3.5-Flash. Prophet Arena reports average Brier score, while FutureX reports accuracy.

4.2 Evaluation Results

Main performance comparison. Table 1 and Table 2 present the main benchmark comparison on Prophet Arena and FutureX. On Prophet Arena, ForeDreamer achieves the best average Brier performance among the compared methods for both backbone models, and improves over Full Text across event categories. On FutureX, ForeDreamer also obtains the strongest accuracy among the evaluated methods. These results indicate that evolving task-level forecasting experience together with procedural evidence-processing experience is more effective than directly using retrieved context or applying existing memory baselines to the retrieved evidence. Additional main benchmark results are provided in Appendix D.1.

Ablation of dual-track experience. Table 3 evaluates the contribution of the two evolution tracks in ForeDreamer. Removing either textual forecasting experience evolution or procedural evidence-processing evolution degrades performance. The full system performs best on both benchmarks, showing that the Experience Bank and the evolved

Method	Prophet Arena Avg. (↓)	FutureX (↑)
Full Text	0.2059	0.3298
w/o Both Optimizations	0.1592	0.3850
w/o Compositional Tool Reuse	0.1541	0.4032
w/o Diversity-Guided Exploration	0.1554	0.3564
ForeDreamer	**0.1471**	**0.4108**

Table 4: Procedural-evolution optimization ablation on Qwen3.5-Flash. The variants remove Compositional Tool Reuse, Diversity-Guided Exploration, or both; Prophet Arena reports average Brier score, and FutureX reports accuracy.

MemGuide–MemTool procedures provide complementary benefits for future event prediction. Additional dual-track experience ablations are provided in Appendix D.2.

Effect of procedural-evolution optimizations. Table 4 isolates the two optimizations used in procedural evidence-processing evolution: Compositional Tool Reuse and Diversity-Guided Exploration. The full design outperforms variants that remove one or both optimizations, indicating that both optimizations contribute to the final performance. Additional procedural-evolution optimization ablations are provided in Appendix D.3.

Robustness across search settings. Table 5 evaluates ForeDreamer under alternative search and interaction settings. ForeDreamer consistently improves across these settings on both benchmarks. The results suggest that the observed gains are not tied to a single search provider, retrieval budget, or interaction configuration. Per-category search-setting results are provided in Appendix D.4.

Check for potential data leakage. Table 6 addresses the concern that the backbone models may have encountered similar forecasting data during pretraining. The No Information setting removes web search and asks the model to forecast from the question alone, while the Full Text setting and ForeDreamer use temporally valid retrieved evidence. The performance gap between No Information and the search-based settings indicates that external web evidence is needed for strong forecasting performance, supporting that the gains are not explained by the backbone model alone. Appendix D.1 further reports results with GPT-5-Nano (OpenAI, 2025), whose training-data cutoff precedes the evaluated benchmark periods, and Appendix D.5 provides additional no-information checks.

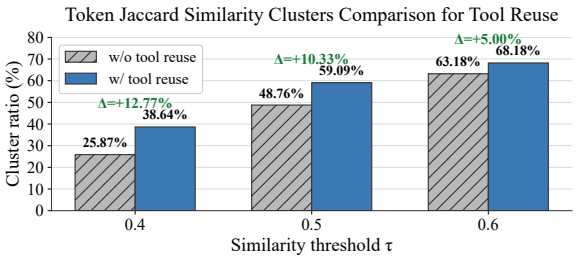


Figure 6: MemTool similarity analysis on FutureX using token-Jaccard source-code similarity. Compositional Tool Reuse produces a less redundant tool set under the same clustering criterion.

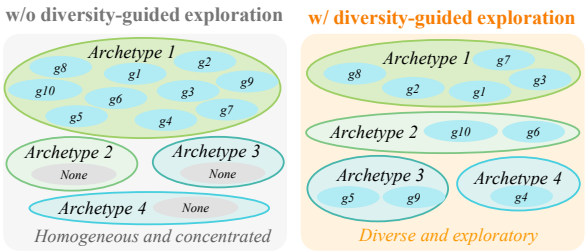


Figure 7: MemGuide pipeline-archetype analysis on FutureX. Diversity-Guided Exploration encourages evolved guides to cover more pipeline archetypes, while vanilla rollout-based expansion remains concentrated in one archetype.

Analysis of Compositional Tool Reuse. Beyond aggregate benchmark scores, we analyze how the evolved procedural memory changes under the proposed optimizations. Section 3.2.2 reports the token-Jaccard clustering observation that motivates Compositional Tool Reuse. Figure 6 compares the generated MemTools on FutureX under the same token-Jaccard setting. The comparison shows that Compositional Tool Reuse produces a less redundant MemTool set, supporting its role in reducing repeated tool construction and improving the efficiency of procedural evolution. Appendix E.1 provides the metric definitions, additional normalized-sequence and AST-based clustering views, and the tool-reuse cluster-ratio table.

Analysis of Diversity-Guided Exploration. Figure 7 analyzes the effect of Diversity-Guided Exploration on evolved MemGuide pipelines. The comparison shows that the proposed exploration strategy encourages MemGuides to cover more pipeline archetypes rather than repeatedly refining a single guide family. Appendix E.2 provides detailed archetype definitions and guide assignments.

Setting	Prophet Arena Full Text (↓)	Prophet Arena ForeDreamer (↓)	FutureX Full Text (↑)	FutureX ForeDreamer (↑)
Base Setting	0.2059	**0.1471**	0.3298	**0.4108**
Max Interaction Turns = 3	0.1871	**0.1570**	0.3894	**0.4423**
Max Interaction Turns = 4	0.1908	**0.1629**	0.3942	**0.4279**
Firecrawl Search	0.1995	**0.1514**	0.4394	**0.4596**
Top-2 Retrieved Results	0.1831	**0.1514**	0.3544	**0.3606**
Top-6 Retrieved Results	0.1845	**0.1426**	0.3398	**0.4183**
Top-8 Retrieved Results	0.1811	**0.1430**	0.3503	**0.3671**
60K Search Context	0.1833	**0.1504**	0.3930	**0.4010**
90K Search Context	0.1855	**0.1504**	0.3831	**0.3990**

Table 5: Robustness across search settings on Qwen3.5-Flash. Each row changes one search or interaction configuration from the base setup and reports the corresponding comparison; Prophet Arena reports average Brier score, and FutureX reports accuracy.

Model	No Information	Full Text	ForeDreamer
Prophet Arena Avg. (↓)			
Qwen3.5-Flash	0.2545	0.2059	**0.1471**
GPT-5.4-Nano	0.2492	0.2084	**0.1839**
FutureX (↑)			
Qwen3.5-Flash	0.3077	0.3298	**0.4108**
GPT-5.4-Nano	0.2260	0.2766	**0.3883**

Table 6: Check for potential data leakage through no-information prompting. The table compares forecasting without web search against settings that use retrieved evidence.

5 Related Work

5.1 Agent Memory

Long-horizon agents require memory beyond prompt-local context, especially when preferences, temporal facts, and reusable task experience must persist across sessions (Zhang et al., 2025b; Hu et al., 2025; Du, 2026; Wu et al., 2025; Lin et al., 2026). Early memory-augmented language systems largely relied on retrieval-augmented generation and dense or non-parametric memory (Lewis et al., 2020; Guu et al., 2020; Borgeaud et al., 2022; Packer et al., 2023), while recent agent-memory work develops more structured storage, updating, and retrieval policies. HippoRAG and HippoRAG 2 organize external knowledge through graph-based associative retrieval for long-term factual and relational recall (Gutierrez et al., 2024, 2025); Mem0, MemoryOS, A-MEM, and LightMem explore persistent profiles, hierarchical memory management, dynamic memory graphs, and efficient consolidation (Chhikara et al., 2025; Kang et al., 2025; Xu et al., 2026; Fang et al., 2025). LangMem represents a complementary engineering direction that integrates semantic, episodic, and procedural memories into agent frameworks (The LangChain Team, 2025).

5.2 Agent Self-Evolving

Self-evolving agents (Yang et al., 2026; Yue et al., 2026; He et al., 2026; Hou et al., 2026; Weng et al., 2026; Guo et al., 2026) improve after deployment through reflection, experience reuse, skill discovery, or procedural modification. Prior work studies failed-trajectory reflection, iterative refinement, tool-use learning, and open-ended skill acquisition (Shinn et al., 2023; Madaan et al., 2023; Schick et al., 2023; Wang et al., 2023). For memory-centric agents, evolution can target reusable memory skills (Zhang et al., 2026), self-reflective meta-memory (Xin et al., 2026), memory-system architectures (Zhang et al., 2025a), or streaming test-time adaptation (Wei et al., 2025). Self-improving coding agents further show that agents can revise their own implementation rather than only their natural-language memory (Robeyns et al., 2025).

5.3 Future Prediction

Future prediction evaluates LLM agents on information gathering, temporal reasoning, calibration, and decisions about unresolved events. Earlier work examines whether language models can approach human forecasting and introduces agentic forecasting benchmarks such as MIRAI and ForecastBench (Halawi et al., 2024; Ye et al., 2024; Karger et al., 2024). Live benchmarks reduce contamination by resolving questions after predictions are made: Prophet Arena evaluates probabilistic predictive intelligence on real-market forecasting tasks (Yang et al., 2025b), while FutureX studies dynamic future prediction with tool use, temporal validity, and robustness to misleading web evidence (Zeng et al., 2025).

6 Conclusion

We presented ForeDreamer, a self-evolving dual-agent framework for open-web future event pre-

diction. ForeDreamer separates question-specific factual memory from experiential memory accumulated across forecasting episodes, allowing a main forecasting agent to reason over processed evidence while a memory-processing subagent constructs this evidence interface with MemGuides and executable MemTools. It further improves procedural evolution through compositional tool reuse and diversity-guided exploration. Experiments on Prophet Arena and FutureX show that open-web forecasting benefits from treating memory not only as stored information, but also as evolving experience for evidence processing and prediction.

Limitations

ForeDreamer is designed for open-web future event prediction, where agents must process noisy search results into forecast-ready factual memory. Evaluating the framework on traditional agent-memory benchmarks is therefore outside the scope of this work, as these benchmarks mainly test whether an agent can store, retrieve, or reuse clean interaction histories. Such benchmarks emphasize different memory functions from the evidence-processing and forecasting setting studied in this paper. Another limitation is that resolved forecasting feedback is scarce in temporally valid future-event prediction, so ForeDreamer evolves and validates its memory using a small feedback pool. This setting may raise concerns about validation-set overfitting. However, the evolved memories are evaluated on held-out forecasting questions that are not used for evolution or validation, and the observed improvements indicate that ForeDreamer can extract reusable forecasting and evidence-processing patterns from a small number of resolved examples rather than merely fitting individual validation questions.

Ethical Considerations

This work focuses on open-web future event prediction using public benchmark datasets. It does not involve collecting private user data, conducting human-subject experiments, or annotating sensitive personal information. All experiments are conducted for research evaluation under temporally constrained forecasting settings. We do not identify additional ethical concerns specific to this work.

References

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George B. M. Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, and 1 others. 2022. Improving language models by retrieving from trillions of tokens. In *Proceedings of the 39th International Conference on Machine Learning*, pages 2206–2240. PMLR.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*.

Pengfei Du. 2026. Memory for autonomous llm agents: Mechanisms, evaluation, and emerging frontiers. *arXiv preprint arXiv:2603.07670*.

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. 2025. Lightmem: Lightweight and efficient memory-augmented generation. *arXiv preprint arXiv:2510.18866*.

Xinshuai Guo, Jiayi Kuang, Linyue Pan, Yinghui Li, Yangning Li, Hai-Tao Zheng, Ying Shen, Di Yin, and Xing Sun. 2026. Evoconfig: Self-evolving multi-agent systems for efficient autonomous environment configuration. *arXiv preprint arXiv:2601.16489*.

Bernal Jimenez Gutierrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. Hipporag: Neurobiologically inspired long-term memory for large language models. *arXiv preprint arXiv:2405.14831*.

Bernal Jimenez Gutierrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. 2025. From rag to memory: Non-parametric continual learning for large language models. *arXiv preprint arXiv:2502.14802*.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR.

Danny Halawi, Fred Zhang, Chen Yueh-Han, and Jacob Steinhardt. 2024. Approaching human-level forecasting with language models. *arXiv preprint arXiv:2402.18563*.

Jinghan He, Junfeng Fang, Feng Xiong, Zijun Yao, Fei Shen, Haiyun Guo, Jinqiao Wang, and Tat-Seng Chua. 2026. Active zero: Self-evolving vision-language models through active environment exploration. *arXiv preprint arXiv:2602.11241*.

Xinmeng Hou, Peiliang Gong, Bohao Qu, Wuqi Wang, Qing Guo, and Yang Liu. 2026. Learn like humans: Use meta-cognitive reflection for efficient self-improvement. *arXiv preprint arXiv:2601.11974*.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, and 1 others. 2025. Memory in the age of ai agents. *arXiv preprint arXiv:2512.13564*.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. 2025. Memory os of ai agent. *arXiv preprint arXiv:2506.06326*.

Ezra Karger, Hamsa Bastani, Matt Jacobs, Kian Ng, Kevin Zhu, Danny Halawi, Jacob Steinhardt, and Philip E. Tetlock. 2024. Forecastbench: A dynamic benchmark of ai forecasting capabilities. *arXiv preprint arXiv:2409.19839*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen-tau Yih, Tim Rocktaschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474.

Zehao Lin, Chunyu Li, and Kai Chen. 2026. A survey on the security of long-term memory in llm agents: Toward mnemonic sovereignty. *arXiv preprint arXiv:2604.16548*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. 2024. Evaluating very long-term conversational memory of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*.

OpenAI. 2024. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

OpenAI. 2025. Introducing GPT-5. https://openai.com/index/introducing-gpt-5/. Accessed: 2026-05-22.

OpenAI. 2026. Introducing GPT-5.4. https://openai.com/index/introducing-gpt-5-4/. Accessed: 2026-05-22.

Charles Packer, Vivian Fang, Shishir G. Patil, Kevin Lin, Sarah Wooders, and Joseph E. Gonzalez. 2023. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*.

Qwen Team. 2026. Qwen3.5: Accelerating productivity with native multimodal agents. https://qwen.ai/blog?id=qwen3.5. Accessed: 2026-05-22.

Maxime Robeyns, Martin Szummer, and Laurence Aitchison. 2025. A self-improving coding agent. *arXiv preprint arXiv:2504.15228*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pages 68539–68551.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*.

The LangChain Team. 2025. LangMem SDK for agent long-term memory. https://www.langchain.com/blog/langmem-sdk-launch. Accessed: 2026-05-22.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H. Chi, Chi Wang, Shuo Chen, Fernando Pereira, Wang-Cheng Kang, and Derek Zhiyuan Cheng. 2025. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory. *arXiv preprint arXiv:2511.20857*.

Zhaotian Weng, Antonis Antoniades, Deepak Nathani, Zhen Zhang, Xiao Pu, and Xin Eric Wang. 2026. Group-evolving agents: Open-ended self-improvement via experience sharing. *arXiv preprint arXiv:2602.04837*.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. 2024. Longmemeval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*.

Yaxiong Wu, Sheng Liang, Chen Zhang, Yichao Wang, Yongyue Zhang, Huifeng Guo, Ruiming Tang, and Yong Liu. 2025. From human memory to ai memory: A survey on memory mechanisms in the era of llms. *arXiv preprint arXiv:2504.15965*.

Haidong Xin, Xinze Li, Zhenghao Liu, Yukun Yan, Shuo Wang, Cheng Yang, Yu Gu, Ge Yu, and Maosong Sun. 2026. Metamem: Evolving meta-memory for knowledge utilization through self-reflective symbolic optimization. *arXiv preprint arXiv:2602.11182*.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2026. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Chengyi Yang, Zhishang Xiang, Yunbo Tang, Zongpei Teng, Chengsong Huang, Fei Long, Yuhan Liu, and Jinsong Su. 2026. Ttcs: Test-time curriculum synthesis for self-evolving. *arXiv preprint arXiv:2601.22628*.

Qingchuan Yang, Simon Mahns, Sida Li, Anri Gu, Jibang Wu, and Haifeng Xu. 2025b. Llm-as-a-prophet: Understanding predictive intelligence with prophet arena. *arXiv preprint arXiv:2510.17638*.

Chenchen Ye, Ziniu Hu, Yihe Deng, Zijie Huang, Mingyu Derek Ma, Yanqiao Zhu, and Wei Wang. 2024. Mirai: Evaluating llm agents for event forecasting. *arXiv preprint arXiv:2407.01231*.

Zhenrui Yue, Kartikeya Upasani, Xianjun Yang, Suyu Ge, Shaoliang Nie, Yuning Mao, Zhe Liu, and Dong Wang. 2026. Dr. zero: Self-evolving search agents without training data. *arXiv preprint arXiv:2601.07055*.

Zhiyuan Zeng, Jiashuo Liu, Siyuan Chen, Tianci He, Yali Liao, Yixiao Tian, Jinpeng Wang, Zaiyuan Wang, Yang Yang, Lingyue Yin, and 1 others. 2025. Futurex: An advanced live benchmark for llm agents in future prediction. *arXiv preprint arXiv:2508.11987*.

Guibin Zhang, Haotian Ren, Chong Zhan, Zhenhong Zhou, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. 2025a. Memevolve: Meta-evolution of agent memory systems. *arXiv preprint arXiv:2512.18746*.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. 2026. Memskill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*.

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. 2025b. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43(6):1–47.

Appendix

A Appendix Overview

This appendix provides additional methodological details, implementation details, experimental results, and analyses to supplement the main paper. It is organized as follows:

- **Appendix B: More Methodological Details**
  - Appendix B.1: Search-and-Process Decomposition
  - Appendix B.2: Factual-Memory Representation
  - Appendix B.3: MemGuide and MemTool Interface
  - Appendix B.4: MemGuide and MemTool Validation
  - Appendix B.5: Procedural Summary Records
  - Appendix B.6: Validation Records and Candidate Selection
  - Appendix B.7: Controlled Tool Execution
- **Appendix C: More Implementation Details**
  - Appendix C.1: Detailed Descriptions of Evaluation Benchmarks
  - Appendix C.2: Detailed Descriptions of Baseline Methods
  - Appendix C.3: Dataset Splits and Evolution Protocol
  - Appendix C.4: Inference and Evaluation Setup
- **Appendix D: More Experimental Results**
  - Appendix D.1: Additional Results on Main Benchmarks
  - Appendix D.2: Additional Dual-Track Experience Ablations
  - Appendix D.3: Additional Procedural-Evolution Optimization Ablations
  - Appendix D.4: Additional Results with Different Search Settings
  - Appendix D.5: Additional Data-Leakage Checks
- **Appendix E: More Analyses**
  - Appendix E.1: Tool Similarity Analysis
  - Appendix E.2: MemGuide Pipeline Archetype Analysis
- **Appendix F: Key Prompt Templates**
  - Appendix F.1: Inference Prompts
  - Appendix F.2: Evolution Prompts

B More Methodological Details

This section provides additional methodological details for ForeDreamer. We describe how search and evidence processing are decomposed, how factual memory is represented, how MemGuides and MemTools are specified and validated, how procedural summaries and validation records support evolution, and how tool execution is controlled.

B.1 Search-and-Process Decomposition

ForeDreamer exposes a single search-and-process interface to the main agent, but internally decomposes factual-memory construction into retrieval and processing stages. The retrieval stage issues the query, applies the task-specific temporal cutoff when supported by the search provider, normalizes the returned evidence, and bounds the amount of evidence passed forward. The processing stage then invokes the memory-processing subagent to transform each retrieved item into concise, task-relevant factual memory. This decomposition keeps the main-agent interface simple while preserving a clear methodological separation between evidence acquisition and evidence transformation. The processed evidence is returned to the main agent as factual support for forecasting, rather than as instructions that can alter the agent's control flow.

B.2 Factual-Memory Representation

ForeDreamer represents factual memory at the granularity of individual retrieved items. Each retrieved item is converted into a structured record that preserves the forecasting question, the search query, the temporal constraint, source metadata, and the retrieved content. The memory-processing subagent transforms this raw record into a plain-text factual-memory output that is consumed by the main agent. This representation makes the evidence path auditable: each piece of processed factual memory can be traced back to the retrieved item from which it was derived. When a MemGuide decomposes evidence processing into multiple stages, intermediate artifacts are kept within the same item-level workspace, so multi-step processing remains localized to the corresponding evidence item.

B.3 MemGuide and MemTool Interface

A MemGuide defines the processing objective and constrains the MemTools that the memory-processing subagent may use during a workflow. It does not directly execute operations; instead, it specifies how the subagent should coordinate available tools to transform raw evidence into factual memory. A MemTool implements a bounded evidence-processing operation with a declared callable interface, such as reading source evidence, extracting salient information, creating intermediate representations, validating processed content, or writing the final factual memory. This separation keeps workflow-level instructions and executable operations distinct while allowing procedural experience to combine LLM-based reasoning with tool-based evidence processing.

B.4 MemGuide and MemTool Validation

Generated procedural assets are checked before they are admitted into active procedural memory. ForeDreamer first verifies that the generated MemGuide has the required workflow fields and that every referenced tool is available either as an existing MemTool or as part of the same candidate. It then validates the generated MemTools for syntactic correctness, interface consistency, and compatibility with the tool-calling schema used by the subagent. Candidate assets are staged and loaded together before publication, so a guide and its tools are accepted only as a mutually consistent unit. This prevents partially valid procedural updates from entering the evolving memory state.

B.5 Procedural Summary Records

ForeDreamer maintains lightweight summaries of the procedural memory state. The MemTool summary describes the capabilities and interfaces of valid tools, and Compositional Tool Reuse uses this summary to select compatible existing tools before generating missing operations for a new design requirement. The MemGuide summary records high-level workflow patterns and representative guides. For Diversity-Guided Exploration, candidate guides are compared with representatives according to workflow intent, stage structure, state transitions, intermediate-artifact usage, tool composition, and responsibility allocation across tools. A candidate that matches an existing workflow pattern is assigned to that category; otherwise, it becomes a new representative. These summaries expose procedural evolution history to the generator in a compact form, without requiring all previous tools, guides, or rollouts to be included in the prompt.

B.6 Validation Records and Candidate Selection

ForeDreamer stores validation records for evolved MemGuides and Experience Bank versions. A validation record is considered admissible only when all validation examples are completed successfully and the task metric can be computed. Candidate ranking follows the benchmark metric: FutureX candidates are ranked by accuracy, while Prophet Arena candidates are ranked by Brier score. Textual-experience versions are distinguished by the content of the active Experience Bank, which avoids conflating validation results from different banks. During procedural evolution, only validation-qualified MemGuide nodes are used for validation-based candidate ranking.

B.7 Controlled Tool Execution

ForeDreamer controls tool execution through two design choices. First, the main agent can access only the public search-and-process interface, while internal retrieval and processing tools remain hidden behind the orchestration layer. Second, during factual-memory construction, the memory-processing subagent may call only the MemTools specified by the active MemGuide. This allowlist ensures that a guide defines not only the intended workflow but also the executable action space available to the subagent. Tool execution is further isolated from external network access and unrelated local state, and each call is subject to a bounded execution budget. These constraints keep evolved tools focused on item-level evidence processing and reduce the risk that procedural memory depends on uncontrolled side effects.

C More Implementation Details

This section provides implementation details for the experimental setup. We describe the evaluation benchmarks, the baseline methods, the dataset splits and evolution protocol, and the inference and evaluation settings used in our experiments.

C.1 Detailed Descriptions of Evaluation Benchmarks

We provide additional details on the two forecasting benchmarks used in our evaluation.

- **Prophet Arena** is a live forecasting benchmark designed to evaluate predictive intelligence on real-world events. The original benchmark continuously collects unresolved events from prediction markets, constructs forecasting contexts, asks models to assign probabilities to possible market outcomes, and evaluates the predictions only after the events resolve. Its evaluation emphasizes both statistical forecast quality and decision value, including Brier score and market-return-style metrics. In our experiments, we use a resolved Prophet Arena snapshot containing 1200 forecasting questions from eight categories: Climate/Weather, Companies, Economics, Entertainment, Mentions, Other, Politics, and Sports. The snapshot times range from June 17, 2025 to November 16, 2025. Because the Sports category contains many examples, our actual evaluation uses a 400-example subset for Sports. We report Brier score as the main metric and mean market return across categories as an auxiliary metric.

- **FutureX** is a live benchmark for future prediction by LLM agents. It is built around a dynamic pipeline that gathers future-oriented questions from web sources, records model predictions before resolution, obtains answers after the resolution date, and scores the prior predictions. The benchmark is designed to reduce data contamination by evaluating events that are genuinely unresolved at prediction time and by refreshing the event pool over time. FutureX covers broad real-world domains such as politics, economics, finance, sports, culture, technology, weather, health, and space, and includes multiple event formats, including single-choice, multi-choice, ranking, and numerical prediction. In our experiments, we use a resolved FutureX evaluation split containing 208 questions with end times from January 15, 2026 to March 10, 2026. Each instance provides the forecasting prompt, event title, end time, difficulty level, and ground-truth answer. We report accuracy after matching the model's final answer to the released ground truth.

C.2 Detailed Descriptions of Baseline Methods

Below, we describe the baseline methods included in our comparison, excluding the Full Text setting:

- **RAG** is a generic retrieval-augmented generation method that conditions generation on retrieved external evidence rather than relying only on parametric model knowledge.

- **HippoRAG 2** is a non-parametric continual-learning framework for LLMs that augments RAG with graph-based memory retrieval, deeper passage integration, and online LLM use.

- **Mem0** is a scalable long-term memory architecture that dynamically extracts, consolidates, and retrieves salient information from ongoing conversations, with an optional graph-memory variant for relational structure.

- **MemoryOS** organizes agent memory through hierarchical short-term, mid-term, and long-term storage together with storage, updating, retrieval, and generation modules.

- **A-MEM** constructs agent memory as an adaptive, interconnected note network inspired by the Zettelkasten method, using dynamic indexing, linking, and memory evolution when new memories are added.

- **LightMem** is a lightweight memory-augmented generation framework that separates sensory, short-term, and long-term memory stages to reduce memory-management overhead while preserving historical interaction information.

- **LangMem** is a LangChain toolkit that provides primitives and tools for extracting, storing, searching, and updating long-term agent memories across interactions.

In our experiments, these baseline methods are applied to process the evidence retrieved by the search module before producing the final forecast.

C.3 Dataset Splits and Evolution Protocol

For Prophet Arena, ForeDreamer is evolved independently for each category. In each category, we randomly sample five examples and use them for both evolution and validation, while the remaining

examples are reserved for evaluation. For FutureX, we randomly sample 20 examples from the 208 resolved examples for evolution and validation, and use the remaining examples for evaluation.

Each evolution run is allocated 60 update iterations. The update process alternates between procedural evolution, which revises MemGuides and MemTools, and declarative evolution, which revises the Experience Bank. During procedural evolution, Diversity-Guided Exploration is interleaved with rollout-based expansion. After evolution, we rank the evolved configurations by validation performance, retain the top five candidates, evaluate them on the held-out test set, and report the best test performance among these candidates.

C.4 Inference and Evaluation Setup

All reported forecasting runs enforce temporal validity during retrieval: search results are restricted to content published before the question-specific cutoff time. Unless otherwise specified, each forecasting run allows at most two interaction turns. The default retrieval setup uses Tavily search, retrieves up to four results per query, and allocates a 30K-character context budget to each search turn. In Table 5 and Table S8, "Max Interaction Turns" varies the interaction budget, "Firecrawl Search" changes the search provider, "Top-k Retrieved Results" varies the retrieval breadth per query, and "NK Search Context" varies the per-turn search-context budget.

D More Experimental Results

This section provides additional experimental results that complement the main evaluation in Section 4. We include results with an additional backbone model, an auxiliary Prophet Arena market-return metric, per-category ablations, per-category search-setting results, and additional checks for potential data leakage.

D.1 Additional Results on Main Benchmarks

Table S1 and Table S2 provide results with GPT-5-Nano. ForeDreamer achieves the strongest performance among the compared methods on both Prophet Arena and FutureX, consistent with the main benchmark comparison. Because the GPT-5-Nano training-data cutoff precedes the evaluated benchmark periods, these results also provide an additional model-side check against potential data leakage.

Table S3 reports Prophet Arena mean market return across categories for Qwen3.5-Flash. ForeDreamer obtains the strongest mean market return among the compared methods, which is consistent with the Brier-score conclusion in the main text.

D.2 Additional Dual-Track Experience Ablations

Table S4, Table S5, and Table S6 provide additional dual-track experience ablations. The Prophet Arena tables give per-category Brier scores for Qwen3.5-Flash and GPT-5.4-Nano, while the FutureX table reports GPT-5.4-Nano accuracy. ForeDreamer obtains the best average performance in these comparisons, while removing either the MemGuide–MemTool evolution or the Experience Bank evolution degrades the overall result. These results are consistent with the main ablation and further support the complementarity of the two evolution tracks.

D.3 Additional Procedural-Evolution Optimization Ablations

Table S7 expands the procedural-evolution optimization ablation on Prophet Arena into per-category results. ForeDreamer achieves the best average Brier score among the compared variants, while removing both optimizations or either individual optimization leads to weaker average performance. This per-category table supports the main conclusion that both optimizations contribute to the final system.

D.4 Additional Results with Different Search Settings

Table S8 reports per-category Prophet Arena results under additional search and interaction settings. ForeDreamer lowers the average Brier score in every setting, indicating that its gains are robust to changes in interaction budget, search provider, retrieval breadth, and search-context budget.

D.5 Additional Data-Leakage Checks

Table S9 and Table S10 give per-category Prophet Arena results for the no-information comparison. The No Information setting removes web search, while the other settings use retrieved evidence. The average results show that access to external evidence improves forecasting performance across the evaluated backbones, supporting that the models rely on web evidence rather than only on memorized parametric knowledge.

Model	Method	Climate/Weather	Companies	Economics	Entertainment	Mentions	Other	Politics	Sports	Avg. (↓)
GPT-5-Nano	Full Text	0.1994	0.2870	0.1347	0.1989	0.2225	0.1897	0.2537	0.2066	0.2116
	RAG	0.2438	0.2489	0.0988	0.1884	0.1997	0.1759	0.2658	0.2049	0.2033
	HippoRAG 2	0.2250	0.2617	0.0987	0.2089	0.2437	0.1656	0.2607	0.2024	0.2083
	Mem0	0.2124	0.3497	0.1373	0.2242	0.2024	0.2378	0.2751	0.2025	0.2302
	MemoryOS	0.1366	0.2603	0.1354	0.2021	0.2412	0.2036	0.2598	0.2063	0.2057
	A-MEM	0.2189	0.2786	0.1075	0.1935	0.2142	0.2021	0.2688	0.2032	0.2109
	LightMem	0.1779	0.2842	0.0764	**0.1760**	0.1989	**0.1029**	0.2583	0.2053	0.1850
	LangMem	0.2753	0.2438	0.0962	0.1906	0.1954	0.1322	0.2610	0.2019	0.1996
	ForeDreamer	**0.0946**	**0.1559**	**0.0476**	0.1970	**0.1798**	0.1804	**0.2517**	**0.1974**	**0.1631**

Table S1: Additional Prophet Arena results for GPT-5-Nano. The table reports Brier score for each category and the average score; lower values indicate better calibrated forecasts.

Model	Method	Accuracy (↑)
GPT-5-Nano	Full Text	0.3723
	RAG	0.3942
	HippoRAG 2	0.3894
	Mem0	0.3776
	MemoryOS	0.3667
	A-MEM	0.3510
	LightMem	0.3077
	LangMem	0.3654
	ForeDreamer	**0.4149**

Table S2: Additional FutureX results for GPT-5-Nano. The table reports prediction accuracy; higher values are better.

Method	Market Return (↑)
Full Text	0.5790
RAG	0.5798
HippoRAG 2	0.6038
Mem0	0.4763
MemoryOS	0.4901
A-MEM	0.5679
LightMem	0.5505
LangMem	0.7324
ForeDreamer	**0.7521**

Table S3: Additional Prophet Arena average market-return results across categories on Qwen3.5-Flash; higher values are better.

E More Analyses

This section provides additional analyses that support the optimization design in ForeDreamer. We include a source-code similarity analysis for MemTools and a pipeline-archetype analysis for evolved MemGuides.

E.1 Tool Similarity Analysis

This appendix details the source-code similarity analysis used to diagnose redundant MemTool generation in the vanilla FutureX setting. We analyze all generated tools in the vanilla configuration without compositional tool reuse or diversity-guided exploration. The analysis contains 201 tools and 20,100 pairwise comparisons. For clustering, we use average-linkage agglomerative clustering over pairwise similarities and report the cluster count at similarity threshold $\tau = 0.5$.

Similarity metrics. We compute three complementary source-code similarities. *Token Jaccard* tokenizes each Python tool with the standard Python tokenizer, removes comments and layout-only tokens, and computes the Jaccard overlap between the two token sets, i.e., $|T_i \cap T_j|/|T_i \cup T_j|$. This is the metric used in the main text because it captures shared implementation vocabulary while being robust to local reordering. *Normalized sequence similarity* uses the same normalized token sequence, but compares two full token sequences with a sequence-matching ratio; it is more sensitive to ordering and near-copy structure. *AST sequence similarity* parses each tool into a Python AST, records the sequence of AST node types visited by `ast.walk`, and compares these node-type sequences with the same sequence-matching ratio; it focuses on structural similarity rather than exact lexical overlap.

Additional clustering results. The main text reports token-Jaccard clustering because it offers the most interpretable evidence of repeated tool-level implementation vocabulary. The additional metrics lead to the same qualitative conclusion that vanilla procedural evolution produces many reusable or near-reusable tools. Under normalized sequence similarity at $\tau = 0.5$, the 201 tools form 84 clusters, corresponding to a cluster ratio of 41.8%. Under AST sequence similarity at $\tau = 0.5$, the tools form 136 clusters, corresponding to a cluster ratio of 67.7%. Figure S1 and Figure S2 show the corresponding clustering views.

Effect of compositional tool reuse. Table S11 compares the cluster ratios before and after enabling compositional tool reuse on Qwen3.5-Flash.

Method	Climate/Weather	Companies	Economics	Entertainment	Mentions	Other	Politics	Sports	Avg. (↓)
Full Text	0.3342	0.2657	0.0720	0.2005	0.2106	0.1306	0.2214	0.2122	0.2059
w/o evolving MemGuide&MemTool	0.2328	0.2199	0.0800	0.1125	0.1678	**0.1274**	0.1843	0.2056	0.1663
w/o evolving Experience Bank	0.2440	0.2492	0.0666	0.1351	0.1869	0.1306	0.2035	0.1996	0.1769
ForeDreamer	**0.1770**	**0.2162**	**0.0414**	**0.0962**	**0.1573**	**0.1274**	**0.1742**	**0.1869**	**0.1471**

Table S4: Additional dual-track experience ablation on Prophet Arena for Qwen3.5-Flash. The table reports Brier score; lower values are better.

Method	Climate/Weather	Companies	Economics	Entertainment	Mentions	Other	Politics	Sports	Avg. (↓)
Full Text	0.2275	0.3719	0.0631	0.1424	0.2878	0.1410	0.2257	0.2075	0.2084
w/o evolving MemGuide&MemTool	0.2275	0.3719	0.0963	0.1349	0.2878	0.1397	0.2222	0.2003	0.2101
w/o evolving Experience Bank	0.2251	**0.3456**	0.0547	0.1551	0.2915	0.1546	0.2230	0.2075	0.2071
ForeDreamer	**0.1811**	**0.3456**	**0.0506**	**0.1117**	**0.2704**	**0.1272**	**0.1927**	**0.1919**	**0.1839**

Table S5: Additional dual-track experience ablation on Prophet Arena for GPT-5.4-Nano. The table reports Brier score; lower values are better.

Method	Accuracy (↑)
Full Text	0.2766
w/o evolving MemGuide&MemTool	0.2766
w/o evolving Experience Bank	0.3404
ForeDreamer	**0.3883**

Table S6: Additional dual-track experience ablation on FutureX for GPT-5.4-Nano. The table reports prediction accuracy; higher values are better.

Across both FutureX and Prophet Arena, tool reuse consistently increases the cluster ratio, meaning that the evolved tool set is less aggressively collapsed into similarity clusters. This supports the intended effect of reducing redundant tool generation.

E.2 MemGuide Pipeline Archetype Analysis

This appendix provides the detailed pipeline-archetype analysis used in Section 4. We analyze 10 MemGuides evolved on FutureX and categorize each guide by its overall evidence-processing pipeline rather than by the surface wording of the guide. Figure S3 shows the resulting archetype assignments before and after enabling diversity-guided exploration.

Pipeline archetypes. We identify four recurring pipeline archetypes:

- **Tool-Orchestrated Processing** uses MemTools for the main evidence-processing path. The pipeline reads the input, applies MemTool-based reading, cleaning, extraction, and transformation, and then writes the final output through a MemTool.
- **Evidence-first LLM synthesis** first extracts evidence with a MemTool and stores it in an intermediate cache or file. The LLM then synthesizes the cached evidence, and a writer MemTool produces the final output.
- **Specification-content decomposition** separates task specification processing from content extraction. A specification extractor writes task requirements into a specification file, a content extractor writes evidence content into a data file, and the LLM jointly synthesizes the two sources before a save-output MemTool writes the result.
- **Planning and validation** decomposes the pipeline into explicit planning and checking stages. The guide reads the input, performs planning, query construction, or option parsing, aggregates or extracts evidence, validates, scores, or ranks candidate evidence, and then writes the final output.

Archetype distribution. Without diversity-guided exploration, all 10 MemGuides are categorized as Tool-Orchestrated Processing. This confirms that vanilla rollout-based expansion concentrates on one pipeline family. After enabling diversity-guided exploration, the 10 MemGuides are distributed across four archetypes: five guides use Tool-Orchestrated Processing, two use Evidence-first LLM synthesis, two use Specification-content decomposition, and one uses Planning and validation. This broader distribution indicates that the exploration path successfully expands procedural evolution beyond local variants of the initially dominant guide pattern.

Method	Climate/Weather	Companies	Economics	Entertainment	Mentions	Other	Politics	Sports	Avg. (↓)
Full Text	0.3342	0.2657	0.0720	0.2005	0.2106	0.1306	0.2214	0.2122	0.2059
w/o Both Optimizations	0.2319	0.1948	0.0471	0.1096	0.1917	**0.1169**	0.1881	0.1935	0.1592
w/o Compositional Tool Reuse	**0.1531**	0.2149	0.0621	0.1038	0.1890	0.1226	0.1965	0.1907	0.1541
w/o Diversity-Guided Exploration	0.1747	**0.1935**	0.0602	0.1023	0.1823	0.1439	0.1921	0.1942	0.1554
ForeDreamer	0.1770	0.2162	**0.0414**	**0.0962**	**0.1573**	0.1274	**0.1742**	**0.1869**	**0.1471**

Table S7: Additional procedural-evolution optimization ablation on Prophet Arena for Qwen3.5-Flash. The table reports Brier score; lower values are better.

Setting	Method	Climate/Weather	Companies	Economics	Entertainment	Mentions	Other	Politics	Sports	Avg. (↓)
Base Setting	Full Text	0.3342	0.2657	0.0720	0.2005	0.2106	0.1306	0.2214	0.2122	0.2059
	ForeDreamer	**0.1770**	**0.2162**	**0.0414**	**0.0962**	**0.1573**	**0.1274**	**0.1742**	**0.1869**	**0.1471**
Max Interaction Turns = 3	Full Text	0.2577	0.2437	0.0955	0.1354	0.2051	0.1639	0.2016	0.1941	0.1871
	ForeDreamer	**0.2165**	**0.2227**	**0.0655**	**0.0930**	**0.1741**	**0.1286**	**0.1710**	**0.1844**	**0.1570**
Max Interaction Turns = 4	Full Text	0.2838	**0.2196**	0.1283	0.1830	0.1999	0.1408	**0.1814**	0.1897	0.1908
	ForeDreamer	**0.2480**	0.2276	**0.0538**	**0.0992**	**0.1672**	**0.1263**	0.1974	**0.1834**	**0.1629**
Firecrawl Search	Full Text	0.2901	0.2377	0.1354	0.1370	0.2056	0.1885	0.2016	0.2001	0.1995
	ForeDreamer	**0.1818**	**0.2019**	**0.0515**	**0.0927**	**0.1782**	**0.1098**	**0.1979**	**0.1970**	**0.1514**
Top-2 Retrieved Results	Full Text	0.2254	0.2514	0.1025	0.1173	0.2083	0.1451	0.2171	**0.1975**	0.1831
	ForeDreamer	**0.2195**	**0.2183**	**0.0456**	**0.0842**	**0.1505**	**0.1064**	**0.1794**	0.2075	**0.1514**
Top-6 Retrieved Results	Full Text	0.2290	**0.2158**	0.1081	0.1372	0.2202	0.1639	0.1905	0.2114	0.1845
	ForeDreamer	**0.1158**	0.2286	**0.0516**	**0.0930**	**0.1580**	**0.1153**	**0.1804**	**0.1980**	**0.1426**
Top-8 Retrieved Results	Full Text	0.2359	0.2300	0.0466	0.1620	0.1770	0.1791	0.2269	**0.1913**	0.1811
	ForeDreamer	**0.1847**	**0.1955**	**0.0335**	**0.0944**	**0.1657**	**0.0919**	**0.1783**	0.2003	**0.1430**
60K Search Context	Full Text	0.2417	**0.2488**	0.0641	0.1438	0.2063	0.1618	0.1957	0.2041	0.1833
	ForeDreamer	**0.1782**	**0.2488**	**0.0524**	**0.1066**	**0.1481**	**0.0995**	**0.1769**	**0.1923**	**0.1504**
90K Search Context	Full Text	0.2711	0.2312	0.0682	0.1251	0.1983	0.1792	0.2242	**0.1868**	0.1855
	ForeDreamer	**0.1908**	**0.2041**	**0.0488**	**0.0904**	**0.1608**	**0.1182**	**0.1869**	0.2035	**0.1504**

Table S8: Per-category results for additional search settings on Prophet Arena with Qwen3.5-Flash (Brier score; lower is better).

Method	Climate/Weather	Companies	Economics	Entertainment	Mentions	Other	Politics	Sports	Avg. (↓)
No Information	0.2576	0.3837	0.1225	0.2634	0.2576	0.2390	0.2945	0.2177	0.2545
Full Text	0.3342	0.2657	0.0720	0.2005	0.2106	0.1306	0.2214	0.2122	0.2059
ForeDreamer	**0.1770**	**0.2162**	**0.0414**	**0.0962**	**0.1573**	**0.1274**	**0.1742**	**0.1869**	**0.1471**

Table S9: Per-category data-leakage check on Prophet Arena for Qwen3.5-Flash. The table reports Brier score; lower values are better.

Method	Climate/Weather	Companies	Economics	Entertainment	Mentions	Other	Politics	Sports	Avg. (↓)
No Information	0.2043	0.4272	0.1441	0.1973	0.2911	0.2310	0.2774	0.2210	0.2492
Full Text	0.2275	0.3719	0.0631	0.1424	0.2878	0.1410	0.2257	0.2075	0.2084
ForeDreamer	**0.1811**	**0.3456**	**0.0506**	**0.1117**	**0.2704**	**0.1272**	**0.1927**	**0.1919**	**0.1839**

Table S10: Per-category data-leakage check on Prophet Arena for GPT-5.4-Nano. The table reports Brier score; lower values are better.

Dataset	Similarity metric	τ	w/o tool reuse	w/ tool reuse	Δ
FutureX	AST sequence	0.4	54.73%	68.18%	+13.46%
FutureX	AST sequence	0.5	67.66%	79.55%	+11.88%
FutureX	AST sequence	0.6	83.58%	84.09%	+0.51%
FutureX	Normalized sequence	0.4	23.88%	36.36%	+12.48%
FutureX	Normalized sequence	0.5	41.79%	45.45%	+3.66%
FutureX	Normalized sequence	0.6	57.21%	63.64%	+6.42%
FutureX	Token Jaccard	0.4	25.87%	38.64%	+12.77%
FutureX	Token Jaccard	0.5	48.76%	59.09%	+10.33%
FutureX	Token Jaccard	0.6	63.18%	68.18%	+5.00%
Prophet Arena	AST sequence	0.4	54.73%	64.21%	+9.48%
Prophet Arena	AST sequence	0.5	70.14%	73.95%	+3.81%
Prophet Arena	AST sequence	0.6	80.52%	81.63%	+1.11%
Prophet Arena	Normalized sequence	0.4	20.66%	33.26%	+12.60%
Prophet Arena	Normalized sequence	0.5	38.92%	49.98%	+11.06%
Prophet Arena	Normalized sequence	0.6	57.67%	62.55%	+4.88%
Prophet Arena	Token Jaccard	0.4	22.06%	36.37%	+14.31%
Prophet Arena	Token Jaccard	0.5	46.54%	55.66%	+9.12%
Prophet Arena	Token Jaccard	0.6	63.98%	68.09%	+4.11%

Table S11: Effect of compositional tool reuse on MemTool similarity clusters. Values are cluster ratios after converting to percentages; higher ratios indicate that fewer tools are merged into similarity clusters and therefore suggest less redundant tool generation. Δ is computed as w/ tool reuse minus w/o tool reuse.

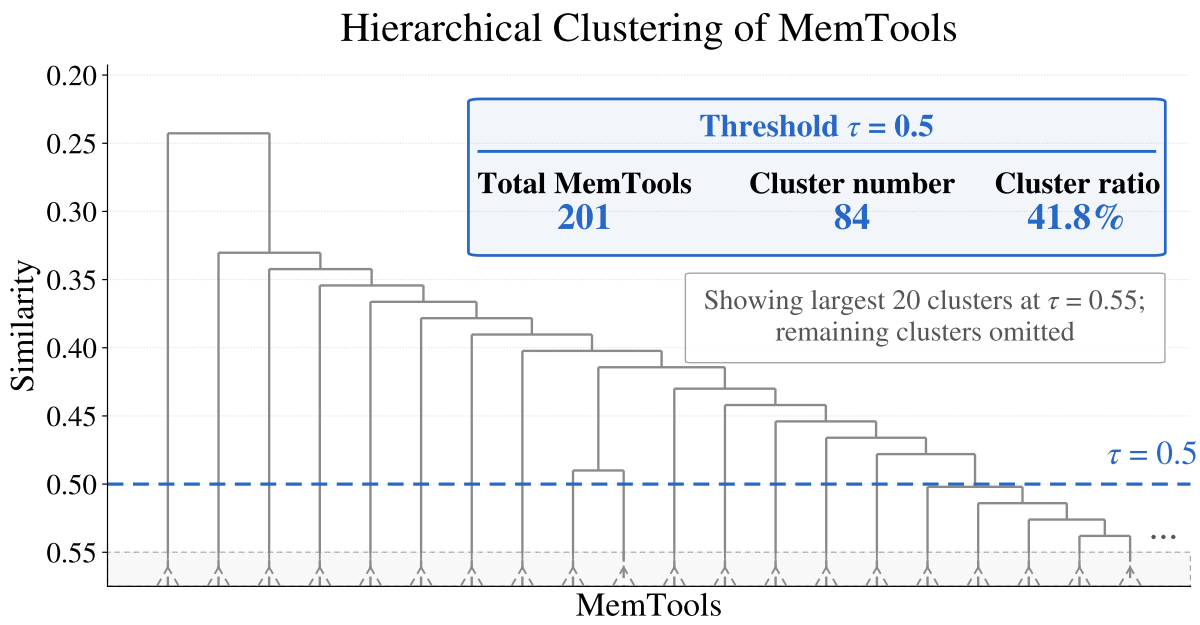


Figure S1: Hierarchical clustering of vanilla FutureX MemTools using normalized token-sequence similarity. The result shows substantial sequence-level overlap among generated tools, complementing the token-Jaccard analysis in the main text.

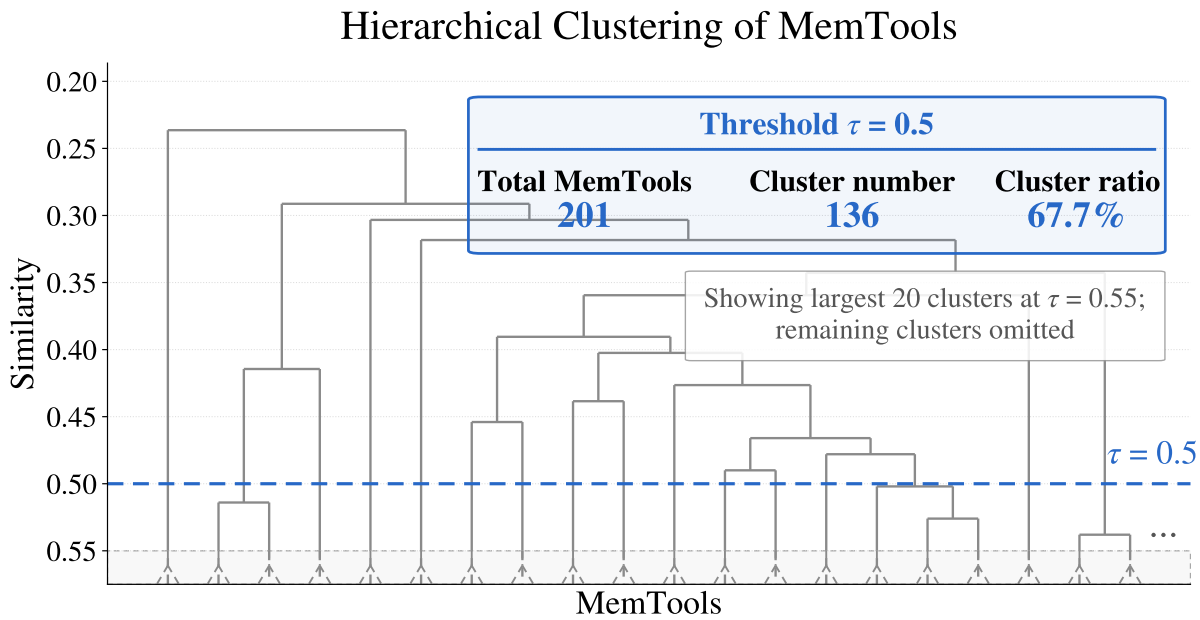
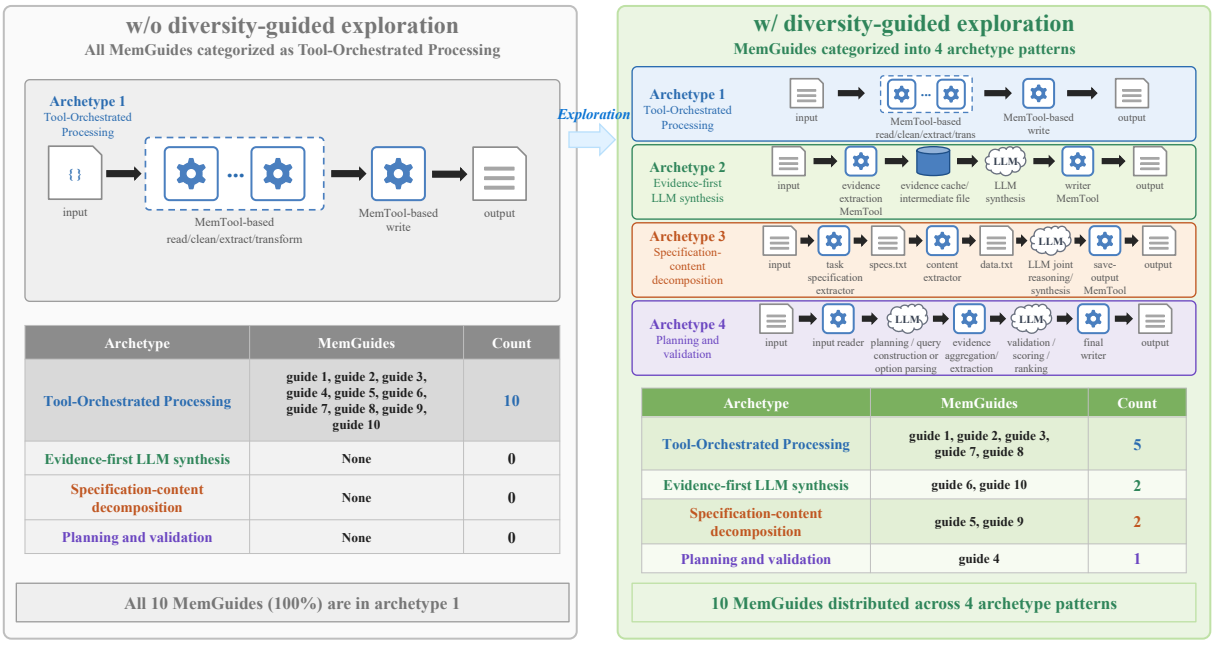


Figure S2: Hierarchical clustering of vanilla FutureX MemTools using AST node-sequence similarity. Even when lexical details are abstracted away, non-singleton clusters remain, indicating repeated structural patterns in generated tools.



Archetype	MemGuides	Count
Tool-Orchestrated Processing	guide 1, guide 2, guide 3, guide 4, guide 5, guide 6, guide 7, guide 8, guide 9, guide 10	10
Evidence-first LLM synthesis	None	0
Specification-content decomposition	None	0
Planning and validation	None	0

All 10 MemGuides (100%) are in archetype 1

Archetype	MemGuides	Count
Tool-Orchestrated Processing	guide 1, guide 2, guide 3, guide 7, guide 8	5
Evidence-first LLM synthesis	guide 6, guide 10	2
Specification-content decomposition	guide 5, guide 9	2
Planning and validation	guide 4	1

10 MemGuides distributed across 4 archetype patterns

Figure S3: Detailed pipeline-archetype analysis for FutureX MemGuides. Without diversity-guided exploration, all 10 MemGuides fall into Tool-Orchestrated Processing. With diversity-guided exploration, the guides cover four archetype patterns, showing broader procedural exploration.

F Key Prompt Templates

F.1 Inference Prompts

```
System message:
You may solve the task over multiple turns and can use the {tool_name} tool when helpful. {cutoff_text} The {tool_name} tool
      searches the web and returns processed evidence rather than raw results. That processed evidence may still be truncated, so
      extract only the most relevant information. You may issue at most one {tool_name} tool call in a single assistant turn. Do
      not batch multiple search requests in one turn. If you need more evidence, use a later turn instead of issuing multiple tool
       calls at once. Only your final answer should follow the task-specific output format.

Final-turn user message:
This is your final allowed assistant turn. Do not call any tool. Produce the final answer now using only the required task-
      specific answer format.
```

Prompt 1: Main-agent multi-turn search prompt.

```
Experience Bank:
These are reusable prior experiences for the main agent. Use them when relevant, but do not let them override direct evidence from
       the current task.
- [{experience_id}] {experience_text}
```

Prompt 2: Experience Bank prompt section.

```
You are an agent that predicts future events. You may reason over multiple turns and use web_search_and_process to gather
      supporting evidence when helpful. The web_search_and_process tool returns processed evidence, not raw search results. {
      search_note} In your final response, provide a detailed rationale section that evaluates and cites the most relevant
      processed evidence, explains why that evidence is relevant or limited, discusses uncertainties or conflicting signals, and
      then end with the boxed final answer.

{experience_bank_section}

Event to be predicted:
{question_text}

Final answer requirements:
{answer_requirements}

Response format for this run:
Rationale: <a detailed explanation that cites and evaluates the most relevant evidence, explains its limitations, and justifies
      the final answer>
\boxed{<final answer>}
Do not put any text after the boxed answer.

Constraint for the final boxed answer line only:
{final_constraints}
```

Prompt 3: FutureX forecasting prompt.

```
You are an AI forecasting assistant. You may reason over multiple turns and use web_search_and_process to gather additional
      evidence when useful. The web_search_and_process tool returns processed evidence, not raw search results. {search_note} In
      your final rationale, evaluate and cite the most relevant processed evidence instead of giving only a brief generic
      explanation. Only your final response should be the required JSON object.

{experience_bank_section}

Question:
{question_context}

Market context:
{market_context}

Evidence:
{sources_context}

Final answer requirements:
Return exactly one JSON object with this schema and no markdown fences:
{
  "rationale": "{rationale_placeholder}",
  "probabilities": {
    "{market_name_1}": <probability_value_from_0_to_1>,
    "{market_name_2}": <probability_value_from_0_to_1>
  }
}

Requirements:
- Use exactly the listed outcome names, case-sensitive.
- Do not invent additional outcomes.
- Every listed outcome must appear exactly once.
- Every probability must be a number between 0 and 1.
- {rationale_requirement}
```

Prompt 4: Prophet Arena forecasting prompt.

```
System message:
You are a data-processing subagent. For the current item_dir, use the available MemTool tools to read the required input file,
      process it, and ensure the required output file is written successfully. Do not invent file contents. Stop once the output
      file has been created.

User message:
Current item_dir: {item_dir}
Input filename: {input_filename}
Output filename: {output_filename}
Create {output_filename} for this item_dir.
```

Prompt 5: Memory-processing subagent prompt.

F.2 Evolution Prompts

```
You are working inside the Memory-Evolving-For-Future-Prediction repository to create or update one `MemGuide`, and optionally
      create or update one or more `MemTool` files, for the data-process subagent.

Your goal is to define a complete, executable, unambiguous data-processing workflow for a single `item_dir`. The workflow must:
- start from `item_dir/input_filename`
- use one or more MemTool steps to extract, organize, clean, or transform relevant information
- allow a MemTool to read source information and let the LLM directly analyze, compress, rewrite, or generate the answer in the
      conversation
- optionally create intermediate files under `item_dir`
- optionally let a MemTool directly write the LLM's final answer into a file
- finally write the required result to `item_dir/output_filename`

Exploration and design preferences:
- Prefer workflows with explicit stage boundaries instead of repeatedly collapsing to the shortest possible "one reader tool + LLM
       + one writer tool" pattern.
- Favor decomposing complex processing into 3 or more bounded stages when appropriate, for example: read, extract, reorganize,
      save intermediate state, reread intermediate state, compress, validate, and finally write.
- Favor MemTools with clearly different responsibilities rather than repeatedly generating near-duplicate reader or writer tools
      that differ only in names, field labels, or return wording.
- Favor explicit saving and rereading of intermediate files inside `item_dir` so workflow state transitions are visible,
      traceable, and reusable.
- When new tools are needed, prefer tools with clearly distinct roles such as: raw-input reader, structured-field extractor,
      content splitter, intermediate-summary writer, intermediate-summary reader, fragment merger, output validator, or final
      writer.
- Do not treat small prompt rewrites, slight field-format changes, or renamed variants of existing `read_*` / `write_*` tools as
      meaningful novelty.

Definitions:
- `MemGuide`: a JSON workflow definition for the data-process subagent. It contains the subagent prompt and the list of MemTools
      the subagent is allowed to call.
- `MemTool`: a Python tool callable by the data-process subagent. Each MemTool performs one bounded processing step inside the
      current `item_dir`.
- `item_dir`: the working directory for one data item. The workflow starts from files inside this directory and must also write
      its final result inside this directory.
- `input_filename`: the main input file for the workflow. The current default example is `raw_data.json`. It is a UTF-8 encoded
      JSON file. In the current example structure, it contains at least a top-level `item` object, and `item` contains at least:
  - `title`
  - `url`
  - `content`
  - `published_date`
  - `score`
  The most important, most stable, and default-required field is `item.content`, which should be treated as a string. The current
       example may also contain top-level metadata fields such as:
  - `task_id`
  - `run_label`
  - `dataset_name`
  - `run_dir_name`
  - `task_dir_name`
  - `search_turn`
  - `item_rank`
  - `query`
  - `problem_statement`
  - `task_requirements`
  - `search_before`
  - `content_source`
  - `result_count`
- `output_filename`: the required final output file for the workflow. The current default example is `final_data.txt`. It is a
      UTF-8 encoded plain-text file, not JSON. Its contents should be the final extracted, transformed, cleaned, or organized text
       derived from `input_filename`. The workflow is only complete when this file is written under the current `item_dir`.

Follow these requirements strictly:
- Always keep the workflow unambiguous and executable.
- Do not depend on interactive input.
- Do not depend on network access.
- Use ASCII by default in code.
- Do not assume extra files exist outside the sandbox.

Interpret the input and output formats according to these facts:
- the current default input is `raw_data.json`
```

```
- the current default output is `final_data.txt`
- `raw_data.json` is structured JSON input
- `final_data.txt` is final plain-text output
- if your design depends on `raw_data.json`, it must at minimum handle `item.content` correctly
- if your design uses additional fields, that usage must remain compatible with the JSON structure described above
- when `raw_data.json` contains `problem_statement` and `task_requirements`, treat them as the upstream task problem and
      requirement description for the current item
- for FutureX, these fields correspond to the event question plus answer requirements and final boxed-answer constraints
- for Prophet Arena, these fields correspond to the prediction question plus the final JSON output requirements
- a valid workflow may be: a MemTool reads the input, the LLM directly produces the answer in the conversation, and another
      MemTool writes that answer to `output_filename`

Runtime environment and constraints:
- Every MemTool runs inside a `bwrap` sandbox subprocess.
- The current `item_dir` is readable and writable.
- `/tmp` is a private writable temporary directory inside the sandbox, but it is not part of the business output directory.
- `src/`, the Python runtime, and system directories are read-only.
- Do not depend on writing outside `item_dir`.
- Do not depend on network access.

MemGuide requirements:
- A MemGuide must be a JSON object.
- It must contain at least:
  - `guide_name`
  - `prompt`
  - `tool_names`
- `guide_name` must be non-empty.
- `prompt` must be non-empty.
- `tool_names` must be a non-empty list.
- Every name in `tool_names` must correspond to a MemTool available to the subagent.
- The `prompt` must explicitly require that the subagent:
  - works only for the current `item_dir`
  - uses the provided MemTool set to finish the task
  - does not invent file contents
  - stops once the target output file has been created
- A MemGuide does not perform file I/O itself. It defines how the subagent should use the MemTools to complete the workflow.
- The MemGuide description of the workflow must remain consistent with the input/output formats above. Do not describe `
      output_filename` as JSON, and do not assume `input_filename` has an arbitrary undefined format.
- A MemGuide may use any of the following patterns, or a combination of them:
  - a tool reads input and another tool directly processes and writes output
  - a tool reads input, the LLM directly reasons or generates the answer in the conversation, and a tool then writes that answer
      to the output file
  - a tool reads input and writes intermediate files, and later tools or the LLM use those intermediate results to produce the
      final output
- When the task allows it, prefer multi-stage workflows with explicit intermediate state, such as "read raw input -> extract and
      write intermediate state -> reread intermediate state and compress/reorganize -> validate or finalize -> write final output".


MemTool requirements:
- Every MemTool must be a Python file corresponding to path format `src/MemTool/tool_<name>.py`
- Every MemTool must define these exports:
  - `TOOL_NAME: str`
  - `TOOL_SPEC: dict[str, Any]`
  - `build_runner_kwargs(arguments, config, runtime_context) -> dict[str, Any]`
  - `run_tool(**kwargs) -> str`
  - `__all__ = ["TOOL_NAME", "TOOL_SPEC", "build_runner_kwargs", "run_tool"]`
- Every MemTool `TOOL_SPEC` must follow OpenAI function-tool style:
  - the top level must be `{"type": "function", "function": {...}}`
  - `function.name` must equal `TOOL_NAME`
  - `function.parameters` must use a JSON Schema object
- Every MemTool `run_tool(...)` must:
  - return a string only
  - raise clear `ValueError` messages for missing input files, invalid formats, or invalid arguments
  - perform business file reads and writes only inside the current `item_dir`
- An individual MemTool does not have to read `item_dir`, `input_filename`, and `output_filename` directly.
- However, the overall guide workflow must be compatible with a runtime context that provides:
  - `item_dir`
  - `input_filename`
  - `output_filename`
- If a MemTool directly reads `input_filename`, it should treat it as a UTF-8 JSON file and should prioritize logic around `item.
      content`.
- If `problem_statement` and `task_requirements` exist, the MemTool or the LLM may use them to process the current item more
      accurately.
- If a MemTool directly writes `output_filename`, it should write UTF-8 plain text, not JSON.
- If a MemTool needs specific runtime parameters, read only the fields it actually needs and validate them explicitly.
- A MemTool may implement only one stage of the workflow. It does not need to complete the whole workflow by itself.
- Reader-style MemTools are allowed: they may only read `input_filename` or intermediate files and return the relevant
      information to the LLM.
- Writer-style MemTools are allowed: they may only take the LLM's current final text and write it to `output_filename`.
- Intermediate-state MemTools are also allowed, for example tools that:
  - write extracted structured data into intermediate files inside `item_dir`
  - reread summaries, fragments, candidate answers, or checklists from intermediate files
  - merge, sort, filter, validate, or rewrite intermediate results
- If you create multiple tools, try to give them meaningfully different roles across stages, data shapes, or state transitions
      instead of duplicating the same "read original input" or "write final output" function.

Recommended minimal skeleton:
```

```python
#!/usr/bin/env python3

from pathlib import Path
from typing import Any, Final


TOOL_NAME: Final[str] = "tool_example"
TOOL_SPEC: Final[dict[str, Any]] = {
    "type": "function",
    "function": {
        "name": TOOL_NAME,
        "description": "Describe what this MemTool does.",
        "parameters": {
            "type": "object",
            "properties": {},
            "additionalProperties": False,
        },
    },
}


def build_runner_kwargs(arguments: dict[str, Any], config: Any, runtime_context: Any) -> dict[str, Any]:

    item_dir = str(getattr(runtime_context, "item_dir", "") or "").strip()
    if not item_dir:
        raise ValueError(f"{TOOL_NAME} requires a non-empty runtime item_dir")

    return {
        "item_dir": item_dir,
    }


def run_tool(*, item_dir: str) -> str:
    item_dir_path = Path(item_dir)
    if not item_dir_path.exists():
        raise ValueError(f"{TOOL_NAME} could not find item_dir: {item_dir_path}")

    return f"{TOOL_NAME} completed its step"


__all__ = ["TOOL_NAME", "TOOL_SPEC", "build_runner_kwargs", "run_tool"]
```

If your design needs direct access to the main input or output files, you may also use this style:

```python
def build_runner_kwargs(arguments: dict[str, Any], config: Any, runtime_context: Any) -> dict[str, Any]:
    del arguments
    del config

    item_dir = str(getattr(runtime_context, "item_dir", "") or "").strip()
    input_filename = str(getattr(runtime_context, "input_filename", "") or "").strip()
    output_filename = str(getattr(runtime_context, "output_filename", "") or "").strip()
    if not item_dir:
        raise ValueError(f"{TOOL_NAME} requires a non-empty runtime item_dir")
    if not input_filename:
        raise ValueError(f"{TOOL_NAME} requires a non-empty runtime input_filename")
    if not output_filename:
        raise ValueError(f"{TOOL_NAME} requires a non-empty runtime output_filename")

    return {
        "item_dir": item_dir,
        "input_filename": input_filename,
        "output_filename": output_filename,
    }
```

Special attention:
- `input_filename = raw_data.json`
- `output_filename = final_data.txt`
- in the most basic example, `final_data.txt` may simply equal `raw_data.json["item"]["content"]`
- in richer examples, `raw_data.json` may also include `problem_statement` and `task_requirements`
- your workflow may still perform better extraction, cleaning, compression, summarization, or transformation, as long as the final output remains plain text and is traceable to `input_filename`

Prompt 6: Shared MemGuide/MemTool authoring rules.

```
You are evaluating whether the main agent's Experience Bank should be updated after one forecasting rollout.

Your job is to inspect:
1. the current Experience Bank (a simplified view that only lists the active experience_id/text pairs),
2. the current best MemGuide validation context,
3. the final run result for the sampled training question,
4. the rollout summary, which includes task metadata, main-agent logs, ground truth, and evaluation feedback.
```

```
You must reason about:
- which information in the rollout is directly relevant to the correct answer,
- which parts of the model behavior conflict with the correct answer,
- whether the current information could have been used more effectively to answer the question,
- whether any insight is general and reliable enough to become a reusable experience,
- whether any existing experience is invalid, misleading, redundant, or should be rewritten.

Current Experience Bank:
{{CURRENT_EXPERIENCE_BANK}}

Note: `CURRENT_EXPERIENCE_BANK` only contains the currently active experiences. Each entry only includes `experience_id` and `
      text`.

Current best guide context:
{{SELECTED_GUIDE_CONTEXT}}

Final run result:
{{RUN_RESULT}}

Rollout summary:
{{ROLLOUT_SUMMARY}}

Note: `ROLLOUT_SUMMARY` only contains the main agent's rollout information. It does not include subagent logs.

Return exactly one JSON object with no markdown fences and no extra text.

Rules:
- Return at most {{MAX_SUGGESTIONS}} suggestions.
- Suggestions must already be ordered from highest priority to lowest priority.
- Use integer priority values where a smaller number means a higher priority.
- `priority = 1` is the highest priority, `priority = 2` is lower, and so on.
- The system will try suggestions in priority order and stop as soon as one suggestion is accepted, so put the most promising
      suggestion first.
- Valid operations are only `add`, `remove`, and `modify`.
- `remove` and `modify` must reference an existing `target_experience_id`.
- When you use `target_experience_id`, copy it exactly from an `experience_id` field shown in `CURRENT_EXPERIENCE_BANK`.
- Never invent, rename, abbreviate, or paraphrase an existing `target_experience_id`.
- `add` and `modify` must provide a concrete `new_text`.
- Only suggest changes that are genuinely generalizable and likely to improve validation.
- If there are no worthwhile changes, return an empty suggestions list.

Required JSON schema:
{
  "suggestions": [
    {
      "priority": <integer_priority_starting_from_1_where_1_is_highest>,
      "operation": "<add_or_remove_or_modify>",
      "target_experience_id": "<required_for_remove_or_modify_else_empty_string>",
      "new_text": "<required_for_add_or_modify_else_empty_string>",
      "analysis": "<short but concrete reason grounded in the rollout>",
      "generality_assessment": "<why this is general enough or why the old experience is invalid>",
      "expected_benefit": "<how this should improve future validation>"
    }
  ]
}
```

Prompt 7: Textual forecasting experience evolution prompt.

```
You are evaluating whether a MemGuide and its available MemTools process retrieved web evidence well enough for downstream
      forecasting.

Your job is not to write code directly. Your job is to inspect:
1. the current MemGuide,
2. the full source code of the tools available to that guide,
3. a processed rollout summary showing how the guide and tools behaved on one training question.

Decide whether the current guide/tool strategy is already good enough.

If it is good enough:
- return `"should_evolve": false`
- explain why in `"analysis"`
- return an empty string for `"design_requirement"`

If it is not good enough:
- return `"should_evolve": true`
- explain what is insufficient in `"analysis"`
- return a concrete `"design_requirement"` that can be passed into a separate MemGuide/MemTool generator
- the design requirement must describe how to improve the current guide/tool strategy while staying compatible with the same
      overall MemTool/MemGuide framework
- the design requirement may reuse existing tools and may also request new tools when necessary
- do not return guide JSON or tool code directly

Evaluation criteria:
- whether the processed output captures the most relevant information from the retrieved source
- whether the guide/tool workflow loses important evidence, keeps too much noise, or formats the result poorly
```

```
- whether the workflow follows the question-specific requirements well enough
- whether the workflow is robust and general rather than overfitting to one exact example
- whether the current tools are sufficient or need to be extended

Current MemGuide JSON:
{{GUIDE_JSON}}

Current MemTool source code:
{{TOOL_SOURCE_CODE}}

Processed rollout summary:
{{ROLLOUT_SUMMARY}}

Return exactly one JSON object with no markdown fences and no extra text:
{
  "should_evolve": <true_or_false>,
  "analysis": "<short but concrete explanation>",
  "design_requirement": "<empty string if should_evolve is false, otherwise a concrete design requirement for the next guide/tool
      generation step>"
}
```

Prompt 8: Procedural critic prompt.

```
{{AUTHORING_COMMON}}

You are now in an intermediate step before final guide/tool generation.

Do not write guide JSON or tool code yet. Your job is to identify up to 3 existing tools that are the best reuse candidates for
      the current design requirement.

Current design requirement:
{{DESIGN_REQUIREMENT}}

Existing available tools (TOOL_NAME + TOOL_SPEC):
{{EXISTING_TOOL_DEFINITIONS}}

Selection rules:
- choose at most 3 existing tools
- only choose from the provided `TOOL_NAME` values
- prefer tools whose `TOOL_SPEC.function.description` and parameter schema indicate they can directly support the requirement
- prefer a small, focused candidate set rather than a broad noisy list
- Do not reuse tools just for the sake of reuse; if the requirement calls for a meaningfully different workflow, returning an
      empty list is acceptable and may leave more room for new tool design.
- Do not choose multiple near-duplicate tools together, especially multiple similar reader tools or multiple similar writer tools;
       such near-duplicates usually should not appear together in the candidate set.
- Prefer complementary tools that can support a multi-stage workflow rather than a homogeneous bundle of tools with nearly
      identical roles.
- If the requirement suggests intermediate files, staged state transitions, or a new split of tool responsibilities, only reuse
      tools that clearly fit one stage of that design; leave the remaining stages open for new tools.
- In your analysis, pay attention to whether a tool can support steps such as reading raw input, writing intermediate state,
      rereading intermediate state, merging intermediate outputs, validating output, or writing the final result.
- if no existing tool is a good fit, return an empty list
- do not invent tool names
- do not return source code
- do not return guide JSON

Return exactly one JSON object with no markdown fences and no extra text:
{
  "candidate_tool_names": ["<TOOL_NAME_1>", "<TOOL_NAME_2>"],
  "analysis": "<short explanation of why these tools are the best reuse candidates, or why none fit>"
}
```

Prompt 9: Compositional tool-reuse selection prompt.

```
{{AUTHORING_COMMON}}

You are now in the final generation step.

Current design requirement:
{{DESIGN_REQUIREMENT}}

Candidate reusable existing tools (TOOL_NAME + TOOL_SPEC):
{{REUSABLE_TOOL_DEFINITIONS}}

Candidate reusable existing tool source code:
{{REUSABLE_TOOL_SOURCE_CODE}}

Generation instructions:
- Prefer reusing the provided candidate existing tools whenever they can satisfy the design requirement.
- If an existing tool is sufficient, reference its existing `TOOL_NAME` in the returned MemGuide and do not recreate that tool.
- Only create new MemTool code when the provided existing tools are not enough, are incompatible with the requirement, or have
      concrete shortcomings.
- It is valid to return only one MemGuide JSON block if no new MemTool code is needed.
- It is also valid to return one MemGuide JSON block plus one or more new MemTool Python code blocks if new tools are required.
```

```
- If you decide not to reuse a provided existing tool, make that decision implicitly through the returned workflow. Do not output
      explanations outside the required code blocks.

Required output format:
- Return only the required content, and nothing else. Do not include explanations, design notes, filename comments, or extra prose
      .
- You must first return exactly one MemGuide JSON block.
- If the current task requires MemTool code, append one or more complete Python code blocks after the MemGuide JSON block.
- If the current task does not require MemTool code, return only the MemGuide JSON block.
- The first non-whitespace characters of your answer must be exactly ```json
- Do not return raw JSON without a fenced code block.
- Do not wrap the whole answer in any outer markdown section, list item, heading, or explanation.
- If you need to return MemTool code, each MemTool must appear in its own separate ```python fenced code block after the JSON
      block.
- If your answer is not formatted as fenced code blocks exactly as required below, it is invalid.
```

Prompt 10: MemGuide/MemTool authoring prompt.

```
You are classifying one newly created MemGuide into an existing set of guide categories.

Each existing category is represented by one representative guide. For each representative guide, you are given:
- `guide_file`
- `guide_name`
- `prompt`
- `tool_names`
- each referenced tool's `TOOL_NAME` and `TOOL_SPEC`

You are also given one new guide candidate with the same fields.

Existing guide category representatives:
{{GUIDE_CATEGORY_REPRESENTATIVES}}

New guide candidate:
{{NEW_GUIDE_CANDIDATE}}

Classification task:
- Decide whether the new guide candidate is highly similar to one existing representative guide category.
- Focus on workflow intent, processing strategy, guide prompt behavior, and the role of the referenced tools.
- Do not focus on filenames or small wording differences alone.
- If two guides share the same high-level state machine, for example both are still "read raw input -> LLM processes -> write
      final output" two-stage workflows, they should usually be classified into the same category even if tool names, field
      formatting, headings, or prompt wording differ.
- If the new guide only renames existing reader or writer tools, changes return formatting, adds a few extra fields, or lightly
      rewrites the prompt while keeping the same stage structure and tool responsibilities, it should still be treated as the
      existing category.
- Only classify a guide as a new category when it introduces substantial change in stage count, state transitions, intermediate-
      file usage, tool-type composition, or responsibility boundaries across tools.
- Pay special attention to whether it introduces explicit intermediate artifacts, longer tool chains, or a genuinely different
      division of roles among tools.
- If the new guide belongs to an existing category, return that representative guide's `guide_file`.
- If the new guide is meaningfully different from every existing category, return `null` so it can become a new category
      representative.

Required output:
- Return exactly one JSON object and nothing else.
- The JSON object must contain:
  - `matched_representative_guide_file`
  - `analysis`
- `matched_representative_guide_file` must be either one of the provided representative `guide_file` values or `null`.
- `analysis` must be a short explanation.

Valid output example:
```json
{
  "matched_representative_guide_file": "guide_3.json",
  "analysis": "The new guide follows the same extract-then-write workflow pattern and uses tools with the same functional role."
}
```

If no existing category matches, return:
```json
{
  "matched_representative_guide_file": null,
  "analysis": "The new guide introduces a distinct workflow pattern and should become a new category representative."
}
```
```

Prompt 11: MemGuide category-classification prompt.

```
{{AUTHORING_COMMON}}

You are now in the exploration generation step.

Current guide category representatives:
{{GUIDE_CATEGORY_REPRESENTATIVES}}
```

```
Exploration objective:
- Create one new MemGuide, and optionally one or more new MemTools, for the data-process subagent.
- The new guide must define a workflow pattern that is clearly different from every existing guide category representative above.
- Do not make only superficial edits such as renaming tools or lightly rewording the prompt.
- The difference should come from workflow design, decomposition strategy, reasoning structure, or tool responsibilities.
- By default, interpret "clearly different" to mean different stage count, different state transitions, different intermediate
      artifacts, different tool-role decomposition, or a meaningfully different information flow through the workflow.
- If the new design is still just "read raw input -> LLM directly processes -> write final output", it is usually not different
      enough unless it also introduces a genuinely different tool-role structure or intermediate-state mechanism.
- Prefer exploring multi-stage workflows over shortest-path workflows; when appropriate, explicitly design 3 or more stages.
- Prefer exploring combinations of different tool types such as extraction tools, structured-reorganization tools, intermediate-
      file writers, intermediate-file readers, fragment-merging tools, validation tools, and final-output writers.
- Strongly favor saving important intermediate results inside `item_dir` and having later tools or the LLM reread them to create
      clearer stage boundaries.
- Do not fake a new workflow by cloning existing `read_*` / `write_*` tools and changing only small wording details, field
      headers, or return strings.
- If existing tools are reused, reuse should usually be only a small part of the new workflow; the main novelty should come from
      new stage structure, tool responsibilities, and intermediate-state design.
- Reusing existing tool names inside the returned MemGuide is allowed only when that truly supports a clearly different guide
      category.
- If a genuinely new workflow requires new tools, create them.

Examples of encouraged exploration directions:
- "read raw input -> extract into intermediate JSON/TXT -> reread intermediate state and produce a compressed version -> reread
      the compressed version and write final output"
- "split original content -> summarize chunks -> save merged notes -> reread notes to generate the final result"
- "extract evidence and metadata first -> save candidate conclusions or checklists -> reread that checklist for validation/rewrite
       -> write final output"
- "separate read / extract / save-intermediate-state / reread-intermediate-state / finalize-write responsibilities across
      different tools instead of pushing all preprocessing into one reader tool"

Required output format:
- Return only the required content, and nothing else. Do not include explanations, design notes, filename comments, or extra prose
      .
- You must first return exactly one MemGuide JSON block.
- If the current task requires MemTool code, append one or more complete Python code blocks after the MemGuide JSON block.
- If the current task does not require MemTool code, return only the MemGuide JSON block.
- The first non-whitespace characters of your answer must be exactly ```json
- Do not return raw JSON without a fenced code block.
- Do not wrap the whole answer in any outer markdown section, list item, heading, or explanation.
- If you need to return MemTool code, each MemTool must appear in its own separate ```python fenced code block after the JSON
      block.
- If your answer is not formatted as fenced code blocks exactly as required below, it is invalid.
```

Prompt 12: Diversity-guided exploration prompt.