Semantic XPath: Structured Agentic Memory Access for Conversational AI

Yifan Simon Liu[*1], **Ruifan Wu**[*1], **Liam Gallagher**[*1], **Jiazhou Liang**[*1]
Armin Toroghi[1], **Scott Sanner**[1,2]
[1]University of Toronto, Canada
[2]Vector Institute of Artificial Intelligence, Toronto, Canada
yifanliu.liu@mail.utoronto.ca

Abstract

Conversational AI (ConvAI) agents increasingly maintain structured memory to support long-term, task-oriented interactions. In-context memory approaches append the growing history to the model input, which scales poorly under context-window limits. RAG-based methods retrieve request-relevant information, but most assume flat memory collections and ignore structure. We propose SEMANTIC XPATH, a tree-structured memory module to access and update structured conversational memory. SEMANTIC XPATH improves performance over flat-RAG baselines by 176.7% while using only 9.1% of the tokens required by in-context memory. We also introduce SEMANTICXPATH CHAT, an end-to-end ConvAI demo system that visualizes the structured memory and query execution details. Overall, this paper demonstrates a candidate for the next generation of long-term, task-oriented ConvAI systems built on structured memory.[1]

1 Introduction

Conversational AI (ConvAI) agents aim to support long-term, task-oriented interactions in which users manage multiple tasks (e.g., travel itinerary planning, to do list management, meal kit recipe recommendation) and repeatedly inspect and revise their evolving artifacts across many versions. To enable such interactions, ConvAI agents increasingly maintain *structured memory* that preserves the conversation history, revision history, and hierarchical organization of each task artifact (Rezazadeh et al. 2024; Sun and Zeng 2025).

Early ConvAI agents rely on an in-context approach that appends the growing conversation history to the model input (Figure 1, left; Zhang et al. 2020). However, this approach scales poorly under context-window limits and incurs higher token costs and latency. Long contexts also lead to poor reasoning and hallucinated outputs (Liu et al., 2024; Zhang et al., 2024). Our empirical results show that the in-context memory approach fails to satisfy half of user requests after only five interaction turns.

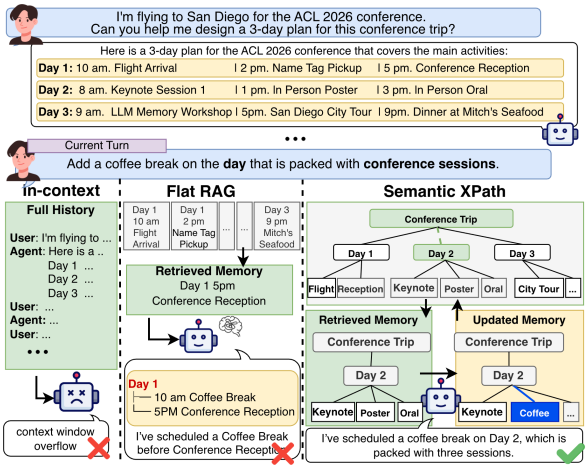

Figure 1: A travel-planning ConvAI agent for a 3-day ACL conference trip in San Diego. **Left:** in-context memory appends the full conversation. **Middle:** flat RAG retrieves from a flattened collection of memory items. **Right:** SEMANTIC XPATH retrieves and updates the relevant substructure from structured memory.

To address these limitations, RAG-based systems have been proposed to retrieve information relevant to the current user request (Lewis et al., 2020; Packer et al., 2023). Many of these approaches treat memory as a flat collection of items during retrieval, which ignores natural hierarchical structure often present in memory (Rezazadeh et al., 2024; Sarthi et al., 2024). For example, in a 3-day ACL trip plan, when a user asks to *"add a coffee break on the day that is packed with conference sessions,"* flat RAG approaches may retrieve conference-related activities from the wrong day due to a lack of structured memory organization (Figure 1, middle).

[*]Equal contribution
[1]Live demo available at https://semanticxpathchat.com. Code available at https://github.com/D3Mlab/SemanticXpath-Chat.git.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations), pages 286–296
July 2-7, 2026

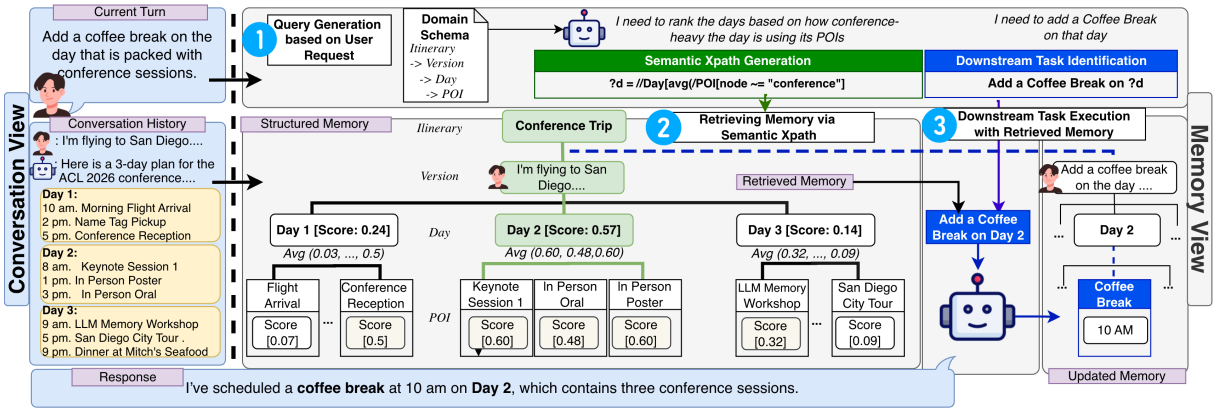

Figure 2: End-to-end SEMANTIC XPATH pipeline on an ACL 2026 trip plan illustrated in both the conversation view (blue) and the memory view (grey). **Step 1:** The user requests to *"add a coffee break on the day that is packed with conference sessions"* is translated into a SEMANTIC XPATH query based on the memory schema. **Step 2:** The query is executed with structural matching and semantic relevance scoring, selecting *Day 2*. **Step 3:** The retrieved substructure is passed to downstream generation to insert a coffee-break POI and respond to the user.

These limitations motivate a structure-aware RAG method that retrieves relevant structured data from memory. A well-known solution for structure-aware retrieval in hierarchical tree-structured data is the XPath query language (Clark and DeRose, 1999), but it lacks semantic awareness and relies on exact matches. Therefore, in this paper:

1. We propose SEMANTIC XPATH (Figure 1, right), a tree-structured memory module that uses an XPath-style query language to retrieve only the relevant memory substructure for efficient memory access and updates.

2. We empirically show that SEMANTIC XPATH improves performance over flat-RAG baselines by 176.7% while using only 9.1% of the tokens required by in-context memory.

3. We introduce SEMANTICXPATH CHAT, an end-to-end ConvAI system for long-term, task-oriented interaction that demonstrates SEMANTIC XPATH with visualizations of the structured memory and query execution details.

2 System Description

Overview. To support long-term, task-oriented interaction between users and agents, we introduce SEMANTICXPATH CHAT, an end-to-end ConvAI system that maintains task-specific, tree-structured memory. This system uses a tree-structured memory module supported by SEMANTIC XPATH to retrieve only the relevant structured data at the appropriate level of granularity, which in turn enables efficient memory access and updates.

Figure 2 shows an example workflow. After creating a 3-day ACL conference trip, the user asks to "*add a coffee break on the day packed with conference sessions*." The system first translates the request into a SEMANTIC XPATH query grounded in the memory schema, then executes it to rank candidates. The top-ranked result is *Day 2*, which contains three conference-related activities. The system then takes the retrieved node and the corresponding subtree for downstream generation, inserts a coffee-break activity into *Day 2*, and records the update by creating a new version branch in structured memory.

Features. Figure 3 shows a screenshot from the SEMANTICXPATH CHAT system to illustrate key features on the same running example. We refer the reader to the demonstration video[2] for a detailed walkthrough of the system.

Conversation View (left) shows the conversation history. Each turn includes action buttons bringing the user to the structured memory view and the execution view in the right panels.

Memory View (middle) visualizes the structured memory. The highlighted path indicates the relevant path for the current user request. (e.g., *Day 2* and added coffee break are highlighted.)

Execution View (right) visualizes the step-by-step execution and scoring. Users can select a query step to inspect its execution details and click candidate nodes to view semantic relevance scores. We present SEMANTIC XPATH query language and execution details in Section 3.

[2] https://www.youtube.com/watch?v=1-DrEr4P5JA

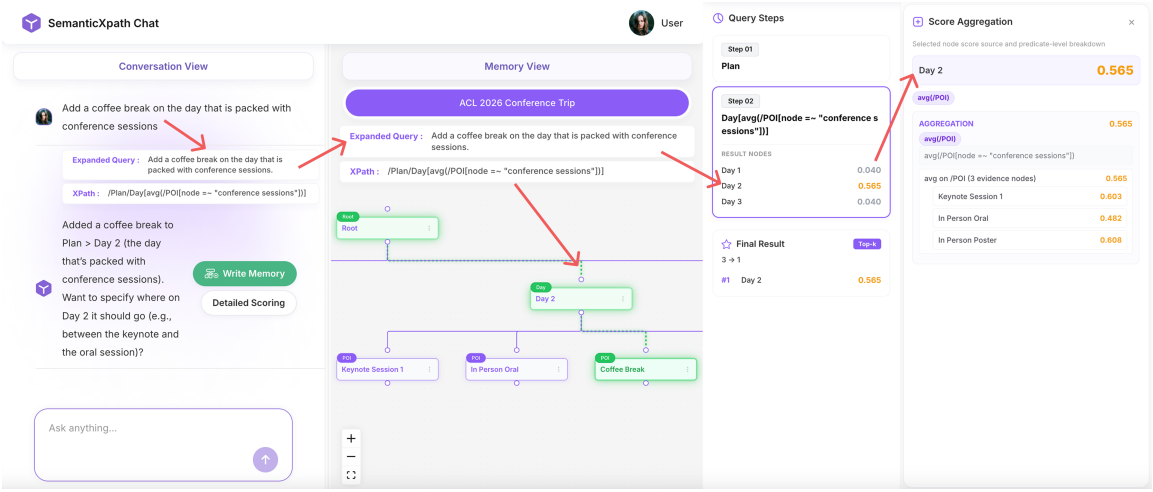

Figure 3: SEMANTICXPATH CHAT demonstration[??]. **Left:** Conversation view, where the user asks to *add a coffee break on the day packed with conference sessions*. **Middle:** Memory view, highlighting the retrieved *Day 2* and the newly inserted coffee break. **Right:** Execution view, showing step-by-step query execution and scoring details.

3 Semantic XPath

SEMANTIC XPATH is a tree-structured memory module that uses an XPath-style query language to retrieve only the relevant structured data for efficient memory access and updates. Figure 2 illustrates the overall pipeline.

3.1 Data Model

We represent structured conversation memory as a rooted tree $T = (V, E, r)$. The root r represents the whole memory. Each parent memory unit contains its child memory units, such as a project containing tasks. Every node except the root has exactly one parent and zero or more children. Each node $v \in V$ stores a node type κ and textual attributes.

The schema Σ specifies the possible node types, textual attributes, and valid parent-child relations. We do not assume a manually designed schema in advance. Instead, Σ is directly determined by the structure already present in the memory data.

ACL Trip Example: The schema for *"3-day ACL conference trip"* example (Figure 2) is:

$$\textit{Itinerary} \rightarrow \textit{Version} \rightarrow \textit{Day} \rightarrow \textit{POI}.$$

- *Itinerary* is the task-instance root.
- *Version* captures revision history (e.g., the initial plan and an updated version after adding a coffee break).
- *Day* and *POI* are task-specific. Each day contains conference activities.

3.2 Query Grammar

Semantic XPath Grammar

Query
$Q ::= \varnothing \mid S \varnothing \mid S\ Q$

Step
$S ::= A\ N\ [R]\ [P]$

Axis navigator
$A ::= \texttt{/} \mid \texttt{//}$

Node selector
$N ::= \texttt{NodeType} \mid *$

Positional selector
$R ::= [i] \mid [-i] \mid [i{:}j]$

Semantic Relevance Operator
$P ::= \frac{(P+P)}{2} \mid P \cdot P \mid \min(P,P) \mid \max(P,P) \mid 1 - P \mid \texttt{Local} \mid \texttt{Agg}(S)$

Local Semantic Relevance
$\texttt{Local} ::= [\texttt{attr\_name ~= String}] \mid [\texttt{node ~= String}]$

Aggregation Semantic Relevance
$\texttt{Agg} ::= \texttt{avg} \mid \texttt{min} \mid \texttt{max} \mid \texttt{gmean}$

A query Q (symbol Q) is a sequence of steps. Each step (symbol S) contains four components:

- An axis navigator (symbol A) that determines how the structured memory is traversed. `/` selects children and `//` selects descendants.
- A node selector (symbol N), which specifies either a concrete node type or the wildcard.

- An optional positional selector (symbol R).
- An optional semantic relevance operator (symbol P), which assigns a semantic relevance score to matched nodes.

We walk through two concrete examples.

ACL Trip Example (cont.): The query Q_1 for a request *"add a coffee break on the day packed with conference sessions"* can be translated into:

```
//Day[avg(/POI[node~="conference"])]
```

Q_1 first matches all *Day* nodes at any depth ($A = //$, $N = Day$). To reflect the notion of a day being *"packed"* with conference sessions, it scores each day with an *aggregation semantic relevance operator* $Agg(S')$, which aggregates POI-level local evidence S' =`/POI[node~="conference"]`.

Using *avg* yields a density-style measure of how conference-heavy the day is.

ACL Trip Example (cont.). The query Q_2 for *"On the third day, remove all activities except the workshop."* can be translated into:

```
//Day[3]/POI[1-[node~="workshop"]]
```

Q_2 first uses a *positional selector*, `//Day[3]`, to match the third day in the structured memory. It then applies a *local semantic relevance operator* at the *POI* level (*Local* = [node $\approx$ "workshop"]), to score each POI by how closely its description matches *"workshop"*. Finally, the *semantic relevance operator* $1 - P$ inverts this score to target all POIs that do *not* match "workshop", identifying the activities to be removed.

3.3 Query Execution

SEMANTIC XPATH query execution maintains a weighted node set $W \subseteq V \times [0, 1]$, where each pair (u, w) assigns a weight w to a node $u \in V$.

Evaluation function. A query is executed by a single recursive evaluation function $\mathsf{E} : \mathcal{X} \times \mathcal{W} \to \mathcal{W}$, where $\mathcal{X}$ ranges over well-formed syntax objects induced by the grammar (e.g., queries, steps, and step components).

Execution is initialized at the root as $\mathsf{E}(Q, \{(r, 1)\})$. Query-level recursion is then:

$$\mathsf{E}(Q, W) = \begin{cases} W, & \text{if } Q = \varnothing, \\ \mathsf{E}(S, W), & \text{if } Q = S\varnothing, \\ \mathsf{E}\big(Q', \mathsf{E}(S, W)\big), & \text{if } Q = SQ'. \end{cases}$$

The remaining cases for the step S and step components (A, N, R, P) are defined in Appendix A. We next illustrate the execution process with the ACL trip running example.

ACL Trip Example (cont.). Consider the SEMANTIC XPATH query Q_1 for *"add a coffee break on the day packed with conference sessions"* :

```
//Day[avg(/POI[node~="conference"])]
```

Q_1 is parsed as a single-step query with an empty suffix, $Q_1 = S\varnothing$, where the step $S = ANP$ with $A = //$, $N = Day$, $P = avg(S')$, and S' =`/POI[node~="conference"]`. The query retrieves the target day and the actual coffee-break insertion is performed by the downstream memory update step.

Execution then proceeds as follows (Figure 2):

1. **Initialize:** Let $W_0 = \{(r, 1)\}$. Each working set W_i contains candidate nodes with relevance scores.
   Initialize execution as $\mathsf{E}(Q, W_0)$.
2. **Reduce the query:** $\mathsf{E}(Q, W_0) = \mathsf{E}(S, W_0)$
3. **Apply axis navigator:** Let $W_1 = \mathsf{E}(//, W_0)$

$$\begin{aligned} \mathsf{E}(Q, W_0) &= \mathsf{E}\big(NP, \mathsf{E}(//, W_0)\big) \\ &= \mathsf{E}(NP, W_1). \end{aligned}$$

   Here, axis navigator maps the root node to all its descendant nodes.
4. **Apply node selector:** Let $W_2 = \mathsf{E}(Day, W_1)$

$$\begin{aligned} \mathsf{E}(NP, W_1) &= \mathsf{E}\big(P, \mathsf{E}(Day, W_1)\big) \\ &= \mathsf{E}(P, W_2). \end{aligned}$$

   Here, the node selector restricts the root node's descendants to *Day* node only.
5. **Apply semantic relevance:**

$$\mathsf{E}(P, W_2) = \mathsf{E}\big(avg(S'), W_2\big).$$

6. **Evaluate inner recursion:**
   $\mathsf{E}\big(avg(S'), W_2\big)$ yields POI-level relevance scores for each day d (i.e., how conference-related its child POIs are). For *Day 2*, the POIs receive semantic relevance scores of 0.603, 0.482, and 0.608 for `/POI[node~="conference"]`, whose average gives a day-level score of 0.565.
7. **Terminate:** Recursion ends at the base case $\mathsf{E}(\varnothing, \cdot)$, selecting *Day 2* with three conference activities as the top-ranked match.

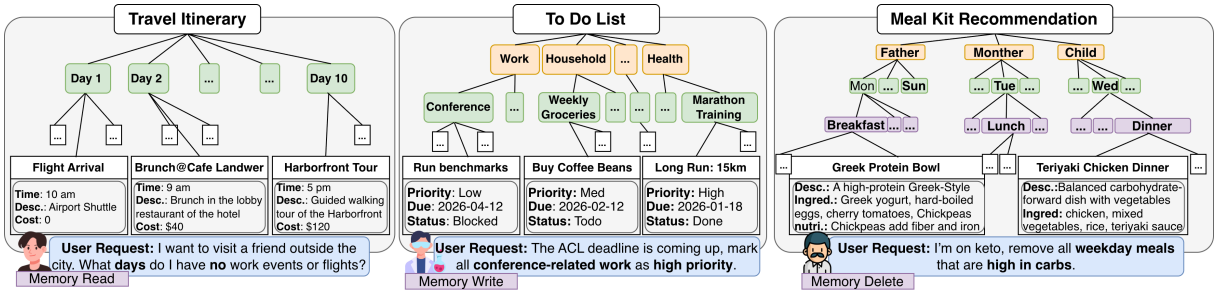

Figure 4: Example of structured memory for three ConvAI scenarios: Travel Itinerary, To-do List, and Meal Kit Recommendation, with representative user requests covering Memory Read, Memory Write, and Memory Delete.

4 Evaluation

4.1 Setup

Evaluation Domains. We evaluate SEMANTIC XPATH on three curated structured-memory ConvAI domains and one standard long-term conversational memory benchmark. The curated domains involve task states with natural hierarchical structure, with simplified schemas shown in Figure 4.

- **Travel itinerary**: Seven-day travel plan, with each day consisting of restaurant and point-of-interest options annotated with descriptions, estimated costs, and preferences.
- **To-do list**: Structured to-do list with task organization spanning 5 categories such as work, household, and personal categories. Each category contains related projects consisting of tasks annotated with a description, progress status, priority level, and deadline.
- **Meal kit recommendation**: 7-day meal plans for a family, where each member is offered three options per meal time, each of which is annotated with ingredients and nutrition.

Each domain has a domain-specific schema and a curated interaction set consisting of 20 *single-turn* conversations and 5 *multi-turn* conversations.

LOCOMO. Beyond these curated task-state domains, we also evaluate on LOCOMO (Maharana et al., 2024), a standard long-term conversational memory benchmark. For each LOCOMO conversation, we directly use its dataset-native hierarchy to form a lightweight tree, `Memory` → `Conversation` → `Session` → `Turn`. This setting tests whether SEMANTIC XPATH can exploit naturally available conversational structure.

Baselines. We compare SEMANTIC XPATH with two memory-access baselines. *In-context* provides the full available memory. *Flat RAG* performs dense retrieval from a flat list of memory units.

Configurations. We use `GPT-5 mini` (OpenAI, 2026) and `Gemini 3 Flash` (Google, 2026) as answer-model backbones. For SEMANTIC XPATH, we compare two relevance scorers: an embedding-based semantic-similarity scorer using `Qwen3-Embedding-8B`, and an entailment scorer using `facebook/bart-large-mnli`. Flat RAG also uses `Qwen3-Embedding-8B` for dense retrieval.

Evaluation metrics. We use two types of metrics. For performance, we report pass rate on the three curated domains, defined as the fraction of requests for which the system retrieves the correct structured data and produces an output satisfying the ground-truth constraints. For LOCOMO, we report LLM-judge accuracy, defined as the fraction of answers judged correct against the reference answer. For cost, we report average token usage per request.

Research Questions. We organize our evaluation around three research questions:

- **RQ1:** In a *single-turn* setting with a static structured memory snapshot, how does SEMANTIC XPATH compare against existing methods on structured ConvAI tasks?
- **RQ2:** In a *multi-turn* setting with dynamic memory maintenance, how does each method perform on interactions that require retrieving revision history across versions, and how does token usage change over turns?
- **RQ3:** Can SEMANTIC XPATH generalize beyond curated task-state domains to a standard conversational memory benchmark?

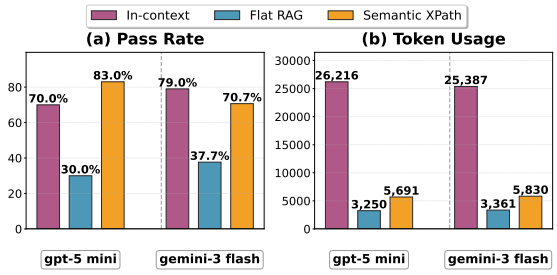

Figure 5: Single-turn evaluation across three methods. We report average pass rate (**left**) and token usage (**right**) averaged over three domains.

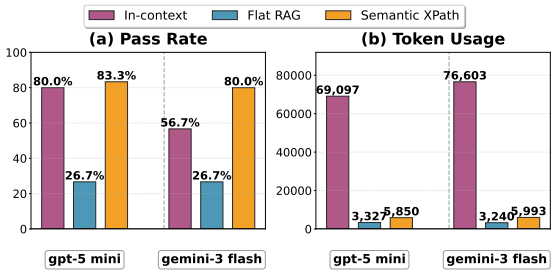

Figure 6: Multi-turn evaluation across three methods. We report average pass rate (**left**) and token usage (**right**) averaged over three domains.

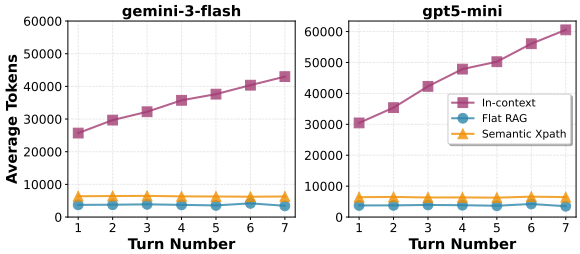

Figure 7: Token usage vs. number of turns in a multi-turn ConvAI setting. Our proposed SEMANTIC XPATH maintains stable token consumption across turns, whereas the in-context baseline exhibits steadily increasing token usage as the conversation history grows.

4.2 Evaluation Results

RQ1: single-turn evaluation. We report single-turn conversation results averaged across three domains (Figure 5, see Appendix B for domain-specific results). As shown on the left of Figure 5, SEMANTIC XPATH achieves a comparable pass rate to the in-context method, with both significantly outperforming the flat RAG baseline.

Flat RAG tends to return surface-level semantic matches that lack hierarchical reasoning. For example, for the request "*The weather forecast shows heavy rain on Day 7. Which activities are outdoors?*", it retrieves outdoor POI across the entire trip rather than scoping results to Day 7.

The in-context method requires roughly 5× higher token usage than the other two methods (Figure 5, right). Therefore, we can conclude that SEMANTIC XPATH achieves comparable effectiveness to the in-context baseline while requiring substantially fewer tokens.

RQ2: Multi-turn evaluation. In Figure 6, the in-context baseline shows a drop in pass rate on requests that require accessing prior conversation history, as longer and more complex contexts increase the chance of misidentifying relevant information. In contrast, SEMANTIC XPATH retrieves only the relevant structured data, maintaining pass rates close to the single-turn setting.

In Figure 7, we evaluate token usage across turns. The in-context method shows a steady increase as it appends each user request and agent response to the input at every turn, whereas SEMANTIC XPATH maintains stable token usage by filtering out information irrelevant to the current request.

RQ3: Broader evaluation on LOCOMO. Table 1 shows that SEMANTIC XPATH generalizes to LOCOMO. The trend is consistent with the curated domains: SEMANTIC XPATH substantially outperforms Flat RAG while approaching the In-context upper bound with much lower token cost. Specifically, it improves the overall answer score from 35.71 to 65.19 with `GPT-5 mini`, and from 41.49 to 73.25 with `Gemini 3 Flash`. Including query generation, it uses 3,032 and 3,870 input tokens on average, which are only 12.2% and 14.1% of the corresponding In-context inputs.

4.3 Failure Analysis

We further analyze final judge-failed cases across both the three curated domains and LOCOMO. We use a three-stage taxonomy: *query generation* errors, where the generated query is syntactically incorrect; *retrieval* errors, where the system fails to retrieve sufficient target evidence or memory nodes; and *generation* errors, where the required evidence or memory scope is retrieved but the final answer is still incorrect.

As shown in Figure 8, retrieval is the main source of remaining failures in both the curated domains and LOCOMO. In the curated domains, these failures usually mean that the system either returns no usable candidate, retrieves only part of the required target set, or selects a related but incorrect memory node. In LOCOMO, the same pattern appears at

Method	Overall	Multi -hop	Temporal	Open -domain	Single -hop	Input Tokens
GPT-5 mini						
In-context	74.55	58.87	59.81	42.71	89.06	24,857
Flat RAG	35.71	14.54	33.96	16.67	45.66	1,341
SEMANTIC XPATH	65.19	43.26	62.31	32.29	77.41	3,032
Gemini 3 Flash						
In-context	87.01	83.33	84.74	52.08	93.10	27,499
Flat RAG	41.49	20.21	47.35	26.04	48.16	1,996
SEMANTIC XPATH	73.25	56.38	77.26	44.79	80.62	3,870

Table 1: Answer score and average input token cost on LOCOMO. For SEMANTIC XPATH, input tokens include query generation. Token counts are approximate across providers because the two answer models use different tokenizers.

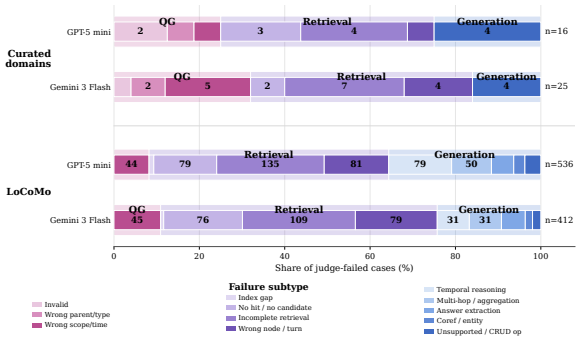

Figure 8: Failure breakdown the three curated domains and LOCOMO. Bars are normalized within each model and setting. We group errors into query generation, retrieval, and generation stages, with lighter shades indicating fine-grained subtypes.

the evidence level: many failed cases retrieve no answer-bearing turn, retrieve only part of the required evidence, or retrieve a topically related but incorrect turn. Query-generation errors are comparatively rare, suggesting that the system usually produces executable queries. Generation errors also occur, but they are less frequent than retrieval errors across both settings.

5 Related Work

External corpus vs. conversational memory. Prior work on memory for LLM agents often treats memory as an *external corpus*, namely a static collection of documents or passages that exists independently of the conversation (Sarthi et al., 2024; Chen et al., 2023; Li et al., 2025; Gupta et al., 2025; Wen et al., 2024, 2025; Liu et al., 2025b,c,a; Liang et al., 2026; Kim et al., 2026). This setting differs from *conversational memory*, where memory represents the agent's current working state for a task and is updated through interaction, making it inherently dynamic and mutable (Sun and Zeng, 2025; Rezazadeh et al., 2024). Our focus is the conversational memory.

Flat vs. structured conversation memory. Within conversational memory, early approaches primarily store interaction history as *flat* textual records with lightweight formatting or compressed summaries of past interactions (Zhong et al., 2024; Wang et al., 2025; Packer et al., 2023; Lu et al., 2023). More recent work introduces *structured* memory representations that organize information into hierarchies to support interaction across multiple abstraction levels (Sarthi et al., 2024; Chen et al., 2023; Rezazadeh et al., 2024; Sun and Zeng, 2025; Li et al., 2025; Gupta et al., 2025). Our work focuses on structured conversational memory.

Abstractive vs. compositional hierarchies in structured memory. Structured conversational memory can be organized as an *abstractive* hierarchy, where higher-level units are lossy summaries of lower-level content, or as a *compositional* hierarchy that structurally organizes task-relevant information (e.g., itinerary → day → activity). While most prior methods focus on abstractive hierarchies (Sarthi et al., 2024; Chen et al., 2023; Rezazadeh et al., 2024; Sun and Zeng, 2025; Li et al., 2025; Gupta et al., 2025), compositional hierarchies are often more natural in task-oriented ConvAI because they reflect the hierarchical representation of interaction and tasks, with each level storing distinct information (e.g., an itinerary vs. individual activities). We thus focus on **compositional hierarchies**.

6 Conclusion

This paper proposes SEMANTIC XPATH, a tree-structured memory module for ConvAI agents. Experiments show that SEMANTIC XPATH outperforms flat RAG baselines while using substantially fewer tokens than in-context memory. We also introduce SEMANTICXPATH CHAT, a live demo for SEMANTIC XPATH that visualizes structured memory and query execution details. Beyond providing an engaging long-term user experience for task-oriented settings such as itinerary planning, SEMANTIC XPATH serves as a practical framework for ConvAI developers to build more efficient structure-aware memory systems.

Ethical Considerations

Our proposed method, SEMANTIC XPATH, is a tree-structured memory module for ConvAI systems. As with any LLM-based approach, the system may inherit biases, factual inaccuracies, or reasoning errors from the underlying LLM. Furthermore, while the structured retrieval approach adopted in our work improves efficiency and relevance, it does not guarantee factual correctness. Users should therefore avoid over-reliance on generated outputs in high-stakes or safety-critical contexts.

References

Howard Chen, Ramakanth Pasunuru, Jason Weston, and Asli Celikyilmaz. 2023. Walking down the memory maze: Beyond context limit through interactive reading. *arXiv preprint arXiv:2310.05029*.

James Clark and Steve DeRose. 1999. XML Path Language (XPath) Version 1.0. W3C Recommendation.

Google. 2026. Gemini 3 Flash. Google AI for Developers Documentation. Accessed: 2026-05-12.

Nilesh Gupta, Wei-Cheng Chang, Ngot Bui, Cho-Jui Hsieh, and Inderjit S Dhillon. 2025. Llm-guided hierarchical retrieval. *arXiv preprint arXiv:2510.13217*.

Junyoung Kim, Anton Korikov, Jiazhou Liang, Justin Cui, Yifan Simon Liu, Qianfeng Wen, Mark Zhao, and Scott Sanner. 2026. Bayesian active learning with gaussian processes guided by llm relevance scoring for dense passage retrieval. *arXiv preprint arXiv:2604.17906*.

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

Rui Li, Zeyu Zhang, Xiaohe Bo, Zihang Tian, Xu Chen, Quanyu Dai, Zhenhua Dong, and Ruiming Tang. 2025. Cam: A constructivist view of agentic memory for llm-based reading comprehension. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Jiazhou Liang, Yifan Simon Liu, David Guo, Minqi Sun, Yilun Jiang, and Scott Sanner. 2026. Evaluating scene-based in-situ item labeling for immersive conversational recommendation. *arXiv preprint arXiv:2604.09698*.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173.

Yifan Liu, Gelila Tilahun, Xinxiang Gao, Qianfeng Wen, and Michael Gervers. 2025a. A comparative study of static and contextual embeddings for analyzing semantic changes in medieval latin charters. In *Proceedings of the First Workshop on Language Models for Low-Resource Languages*, pages 182–192.

Yifan Liu, Qianfeng Wen, Jiazhou Liang, Mark Zhao, Justin Cui, Anton Korikov, Armin Toroghi, Junyoung Kim, and Scott Sanner. 2025b. Multimodal item scoring for natural language recommendation via gaussian process regression with llm relevance judgments. *Preprint*, arXiv:2510.22023.

Yifan Liu, Qianfeng Wen, Mark Zhao, Jiazhou Liang, and Scott Sanner. 2025c. Ma-dpr: Manifold-aware distance metrics for dense passage retrieval. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 31073–31091.

Junru Lu, Siyu An, Mingbao Lin, Gabriele Pergola, Yulan He, Di Yin, Xing Sun, and Yunsheng Wu. 2023. Memochat: Tuning llms to use memos for consistent long-range open-domain conversation. *Preprint*, arXiv:2308.08239.

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. 2026. GPT-5 mini. OpenAI API Documentation. Accessed: 2026-05-12.

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

Alireza Rezazadeh, Zichao Li, Wei Wei, and Yujia Bao. 2024. From isolated conversations to hierarchical schemas: Dynamic tree memory representation for llms. *arXiv preprint arXiv:2410.14052*.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D Manning. 2024. Raptor: Recursive abstractive processing for tree-organized retrieval. In *The Twelfth International Conference on Learning Representations*.

Haoran Sun and Shaoning Zeng. 2025. Hierarchical memory for high-efficiency long-term reasoning in llm agents. *arXiv preprint arXiv:2507.22925*.

Qingyue Wang, Yanhe Fu, Yanan Cao, Shuai Wang, Zhiliang Tian, and Liang Ding. 2025. Recursively summarizing enables long-term dialogue memory in large language models. *Neurocomputing*, 639:130193.

Qianfeng Wen, Yifan Liu, Justin Cui, Joshua Zhang, Anton Korikov, George-Kirollos Saad, and Scott Sanner. 2025. A simple but effective elaborative query

reformulation approach for natural language recommendation. *Preprint*, arXiv:2510.02656.

Qianfeng Wen, Yifan Liu, Joshua Zhang, George Saad, Anton Korikov, Yury Sambale, and Scott Sanner. 2024. Elaborative subtopic query reformulation for broad and indirect queries in travel destination recommendation. *Preprint*, arXiv:2410.01598.

Xuechen Zhang, Xiangyu Chang, Mingchen Li, Amit Roy-Chowdhury, Jiasi Chen, and Samet Oymak. 2024. Selective attention: Enhancing transformer through principled context control. *Advances in Neural Information Processing Systems*, 37:11061–11086.

Yizhe Zhang, Siqi Sun, Michel Galley, Yen-Chun Chen, Chris Brockett, Xiang Gao, Jianfeng Gao, Jingjing Liu, and William B Dolan. 2020. Dialogpt: Large-scale generative pre-training for conversational response generation. In *Proceedings of the 58th annual meeting of the association for computational linguistics: system demonstrations*, pages 270–278.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19724–19731.

A Query Execution

A.1 Step Evaluation

$$\mathsf{E}(S, W) = \begin{cases} \mathsf{E}\big(NRP,\, \mathsf{E}(A, W)\big), & \text{if } {}^{S=}_{ANRP}, \\ \mathsf{E}\big(NP,\, \mathsf{E}(A, W)\big), & \text{if } {}^{S=}_{ANP}, \\ \mathsf{E}\big(NR,\, \mathsf{E}(A, W)\big), & \text{if } {}^{S=}_{ANR}, \\ \mathsf{E}\big(N,\, \mathsf{E}(A, W)\big), & \text{if } {}^{S=}_{AN}. \end{cases}$$

A.2 Axis Selector Evaluation

Define the axis expansion operators:

$$\mathsf{E}(\texttt{/}, W) = \Big\{(u, w) \;\Big|\; {}^{\exists (v,w)\in W,}_{u\in \mathsf{C}(v)}\Big\},$$
$$\mathsf{E}(\texttt{//}, W) = \Big\{(u, w) \;\Big|\; {}^{\exists (v,w)\in W,}_{u\in \mathsf{D}(v)}\Big\}.$$

where $\mathsf{C}(v)$ and $\mathsf{D}(v)$ denote the child and descendant sets of node v, respectively.

The evaluation function for the axis selector is then defined as:

$$\mathsf{E}(A, W) = \begin{cases} \mathsf{E}(\texttt{/}, W), & \text{if } A = \texttt{/}, \\ \mathsf{E}(\texttt{//}, W), & \text{if } A = \texttt{//}. \end{cases}$$

A.3 Node Selector Evaluation

Define the node selector operators:

$$\mathsf{E}(\mathsf{NodeType}, W) = \Big\{(u, w) \;\Big|\; {}^{(u,w)\in W,}_{\kappa(u)=\mathsf{NodeType}}\Big\},$$
$$\mathsf{E}(\star, W) = W.$$

where $\kappa(u)$ returns the node type of u.

The evaluation function for the node selector is then defined as:

$$\mathsf{E}(N, W) = \begin{cases} \mathsf{E}(\mathsf{NodeType}, W), & \text{if } {}^{N=}_{\mathsf{NodeType}}, \\ \mathsf{E}(\star, W), & \text{if } N = \star. \end{cases}$$

A.4 Positional Selector Evaluation

Define the positional selector operators:

$$\mathsf{E}([i], W) = [i](W),$$
$$\mathsf{E}([-i], W) = [-i](W),$$
$$\mathsf{E}([i{:}j], W) = [i{:}j](W),$$

where $[i](\cdot)$, $[-i](\cdot)$, and $[i{:}j](\cdot)$ apply positional selection to W in the tree order induced by the memory tree..

The evaluation function for the positional selector is then defined as:

$$\mathsf{E}(R, W) = \begin{cases} \mathsf{E}([i], W), & \text{if } R = [i], \\ \mathsf{E}([-i], W), & \text{if } R = [-i], \\ \mathsf{E}([i{:}j], W), & \text{if } R = [i{:}j]. \end{cases}$$

A.5 Semantic Relevance Operator Evaluation

We define a semantic relevance scoring function $\mathsf{Rel} : V \times \Phi \to [0, 1]$, which assigns a graded relevance score to a node u under a semantic relevance condition $P \in \Phi$.

Local semantic relevance. Define the relevance scoring function for local semantic relevance:

$$\mathsf{Rel}(u, \texttt{Local}) = \mathsf{Atom}_{\mathrm{loc}}(u, \texttt{Local}),$$

where $\mathsf{Atom}_{\mathrm{loc}}(u, \texttt{Local}) \in [0, 1]$ is computed from either a specific attribute or the full node representation via a semantic scoring function.

Aggregation semantic relevance. If $P := \mathsf{Agg}(S')$, where the inner step S' has the surface form $S' := ANRP'$, let $\mathcal{E}^{(u)}_{S'} \subseteq V$ denote the evidence node set obtained by structurally executing S' from u. The aggregation relevance scoring function is defined recursively as

$$\mathsf{Rel}(u, \mathsf{Agg}(S')) = \mathsf{Agg}\Big(\{\, \mathsf{Rel}(x, P') \mid x \in \mathcal{E}^{(u)}_{S'} \,\}\Big),$$

where Agg is an aggregation operator such as `min`, `max`, `avg`, or `gmean`.

The evaluation function for the semantic relevance operator is then defined as:

$$\mathsf{E}(P, W) = \{(u,\ w \cdot \mathsf{Rel}(u, P)) \mid (u, w) \in W\}.$$

Compositional semantic relevance. For the unary operator $1 - P$, we define

$$\mathsf{Rel}(u,\ 1 - P) = 1 - \mathsf{Rel}(u, P),$$

and the corresponding semantic relevance operator

$$\mathsf{E}(1{-}P, W) = \Big\{(u,\ w{\cdot}\mathsf{Rel}(u, 1{-}P)) \;\Big|\; {}_{(u,w)\in W}\Big\}.$$

For binary compositional conditions, let $\odot \in \{\mathit{avg}, \mathit{prod}, \mathit{min}, \mathit{max}\}$ be a binary operator on $[0, 1]$, and for any node u and relevance conditions P_1, P_2, define

$$\mathsf{Rel}(u,\ P_1 \odot P_2) = \odot\big(\mathsf{Rel}(u, P_1),\ \mathsf{Rel}(u, P_2)\big).$$

The corresponding compositional semantic relevance operator is

$$\mathsf{E}(P_1 \odot P_2, W) = \Big\{(u,\ w \cdot \mathsf{Rel}(u, P_1 \odot P_2)) \;\Big|\; {}_{(u,w)\in W}\Big\}$$

The evaluation function for the semantic relevance operator is then defined as:

$$\mathsf{E}(P, W) = \begin{cases} \mathsf{E}(\texttt{Local}, W), & \text{if } {}^{P=}_{\texttt{Local}}, \\ \mathsf{E}(\mathsf{Agg}(S'), W), & \text{if } {}^{P=}_{\mathsf{Agg}(S')}, \\ \mathsf{E}(1 - P', W), & \text{if } {}^{P=}_{1-P'}, \\ \mathsf{E}(P_1 \odot P_2, W), & \text{if } {}^{P=}_{P_1 \odot P_2}, \end{cases}$$

Model	Method	Scorer	Itinerary		To-do List		Meal Kit		Avg.	
			Pass	Tokens	Pass	Tokens	Pass	Tokens	Pass	Tokens
GPT-5 mini	In-context	–	55.0	12,136	80.0	5,444	75.0	61,069	70.0	26,216
	Flat RAG	–	25.0	3,137	40.0	2,722	25.0	3,892	30.0	3,250
	Semantic XPath	*Sem*	75.0	5,548	64.0	5,937	60.0	7,013	66.3	6,166
	Semantic XPath	*Entail*	85.0	5,161	84.0	5,184	80.0	6,728	83.0	5,691
Gemini 3 Flash	In-context	–	95.0	12,846	72.0	10,465	70.0	52,849	79.0	25,387
	Flat RAG	–	35.0	3,349	48.0	2,934	30.0	3,800	37.7	3,361
	Semantic XPath	*Sem*	65.0	5,454	68.0	5,710	60.0	6,966	64.3	6,043
	Semantic XPath	*Entail*	75.0	5,240	72.0	5,605	65.0	6,644	70.7	5,830

Table 2: Single-turn evaluation across methods. *Sem* denotes the Semantic Similarity scorer and *Entail* denotes the Entailment scorer for SEMANTIC XPATH. We report average pass rate and token usage with GPT-5 mini and Gemini 3 Flash as backbones across three domains: Travel Itinerary, To-do List, and Meal Kit Recommendation.

Model	Method	Scorer	Itinerary		To-do List		Meal Kit		Avg.	
			Pass	Tokens	Pass	Tokens	Pass	Tokens	Pass	Tokens
GPT-5 mini	In-context	–	90.0	28,342	70.0	20,398	80.0	158,552	80.0	69,097
	Flat RAG	–	20.0	3,250	50.0	3,043	10.0	3,686	26.7	3,327
	Semantic XPath	*Sem*	50.0	5,664	100.0	6,132	60.0	7,087	70.0	6,294
	Semantic XPath	*Entail*	80.0	5,820	100.0	5,649	70.0	6,079	83.3	5,850
Gemini 3 Flash	In-context	–	60.0	31,639	70.0	23,602	40.0	174,567	56.7	76,603
	Flat RAG	–	20.0	3,164	50.0	2,959	10.0	3,596	26.7	3,240
	Semantic XPath	*Sem*	60.0	5,451	100.0	5,872	70.0	6,954	76.7	6,092
	Semantic XPath	*Entail*	60.0	5,671	100.0	5,526	80.0	6,782	80.0	5,993

Table 3: Multi-turn evaluation across methods. *Sem* denotes the Semantic Similarity scorer and *Entail* denotes the Entailment scorer for SEMANTIC XPATH. We report average pass rate and token usage with GPT-5 mini and Gemini 3 Flash as backbones across three domains: Travel Itinerary, To-do List, and Meal Kit Recommendation.

B Additional Results

Across both single-turn and multi-turn results (Table 2 and Table 3), in-context memory achieves strong pass rates but incurs very high token usage, especially on meal kit recommendation queries. SEMANTIC XPATH substantially reduces token cost while maintaining competitive accuracy, and the entailment scorer consistently outperforms the semantic similarity scorer in pass rate for both GPT-5 mini and Gemini 3 Flash. Flat RAG remains the lowest-cost baseline but with noticeably lower pass rates.

C Case Study

Beyond the single-turn requests shown in Figure 2, a real ConvAI system must support long-term, task-oriented interactions. Users may revise a plan over many turns and later return to earlier versions or even different plans. The key challenge is whether the system can reliably retrieve the right information across revisions and plans despite ongoing edits and growing history.

Below, we demonstrate a case with example plan revision requests from users using the same ACL trip example:

ACL Trip Example (cont.) We illustrate two requests that interact with revision history.

Request 1: *"Cancel the poster session visit cause I need to take a client meeting."*
Query:

```
//POI[node~="poster"]
```

10 follow-up requests omitted: · · ·

Request 2: *"Wait, what was the poster session time again? I might be able to make it."*
Query:

```
//Version[node~="deletepostersession"]
        //POI[node~="poster"]
```

Result: SEMANTIC XPATH retrieves the version created by the deletion edit, recovers the removed poster-session entry from that revision context, and returns its scheduled time. In contrast, in-context memory fails under the longer context.