MEMQUILT: CONTEXT-BUDGET-AWARE SUBMODULAR MEMORY SELECTION FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-term memory enables large language model agents to reuse past information, but its effectiveness depends on which memories enter their limited working context. Existing memory-selection methods estimate individual values from relevance, utility, or task-specific signals. However, pointwise ranking may select memories that repeatedly support the same requirements, while similarity-based diversification does not ensure task complementarity. Moreover, under a context budget, lengthy memories may displace complementary evidence, and selecting a fixed number of memories may add low-value context after the required evidence has been recovered. To address these limitations, we propose MemQuilt, a set-aware memory selection framework that integrates individual value, task-requirement coverage, and memory-context cost through set-conditioned marginal utility. Specifically, MemQuilt proceeds in three stages: (i) missing-requirement analysis identifies missing task requirements, (ii) requirement-conditioned memory assessment estimates a requirement-support vector for each candidate, and (iii) context-budget-aware submodular selection selects a memory set using an objective that combines individual value with concave requirement coverage. If no missing requirement is identified, MemQuilt returns an empty memory context. Otherwise, cost-aware greedy selection and threshold stopping determine when further context is no longer worthwhile. Extensive experiments on MS-TOD, Mem2Act, and LoCoMo demonstrate that MemQuilt consistently achieves empirical Pareto improvements in task performance and memory-context cost across different memory units and evidence structures. Notably, relative to Top-K, MemQuilt improves SlotAcc by **11.17** percentage points while reducing memory-context tokens by **38.8%** on MS-TOD.

1 INTRODUCTION

Long-term memory enables large language model agents to reuse facts, preferences, and experiences across interactions, supporting sustained interaction and task execution (Shinn et al., 2023; Park et al., 2023; Zhong et al., 2024; Wang et al., 2024; He et al., 2026). As histories accumulate, however, memory banks outgrow the context that can be usefully devoted to the current task. Supplying all available history increases computation and exposes the agent to irrelevant, redundant, or conflicting information (Wu et al., 2024; Pan et al., 2025; Yang et al., 2026). Memory systems therefore commonly adopt a retrieve-then-select pipeline. They first construct a candidate pool and then select a small memory context using individual values derived from relevance, utility, or task-specific signals (Tan et al., 2025; Yan et al., 2026; Zhang et al., 2026). This pipeline makes growing memory banks tractable by reducing the available history to a small set of promising memories.

However, individual usefulness does not ensure collective usefulness. Several high-scoring memories may repeat the same evidence while leaving another information need unsupported. Selecting a fixed number of memories may also continue adding low-value context after the required evidence has been recovered. Recent agent-memory systems incorporate set-aware rerankers such as maximal marginal relevance (MMR), which reduce redundancy by favoring semantic diversity (Carbonell & Goldstein, 1998; Rasmussen et al., 2025). Yet semantic difference does not necessarily imply task complementarity. A semantically distinct memory may provide no missing evidence, whereas a similar memory may contain a critical detail. As illustrated in Fig. 1, both pointwise ranking and

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

similarity-based diversification can leave task requirements unsupported. The central question is therefore how to define the collective value of memories for the current task.

To address this gap, we formulate memory selection through set-conditioned marginal utility. The value of a candidate should depend on the task-relevant information it adds to the memories already selected. Once a requirement is well supported, further evidence for it should provide diminishing additional value. Memories that address unmet requirements should instead retain their complementary value. This requirement-level view allows evidence to accumulate without repeatedly rewarding the same contribution. Set construction must also account for memory-context cost. A lengthy memory may strongly support one requirement while displacing several shorter memories that jointly support different needs. The selector should therefore compare conditional task gain against token cost and stop before exhausting the budget when no remaining candidate provides sufficient additional value. Memory selection must jointly determine **which information to select for, how much context to allocate, and when to stop**.

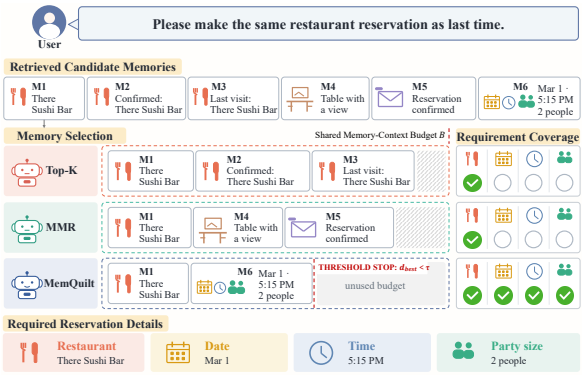

Figure 1: Motivating example under a shared memory-context budget. Top-K selects redundant evidence, while MMR favors diversity without covering the required details. MemQuilt selects complementary memories that cover all active requirements and adaptively stops with unused budget.

In this work, we propose **MemQuilt**, a **set-aware memory selection framework** that jointly models individual value, task-requirement coverage, and memory-context cost. MemQuilt proceeds in three stages. First, missing-requirement analysis identifies the information that the current task still needs from historical memory. Second, requirement-conditioned memory assessment produces a **requirement-support vector** for each candidate. Third, context-budget-aware submodular selection combines individual value with concave requirement coverage. When the first stage identifies no active requirement, MemQuilt returns an empty memory context. Otherwise, cost-aware greedy construction selects budget-feasible candidates according to their conditional marginal gain relative to context cost, and threshold stopping terminates selection when further context is no longer justified. This design rewards complementary evidence across requirements while retaining useful supplementary support. We evaluate MemQuilt on MS-TOD, Mem2Act, and LoCoMo, covering task-oriented dialogue, tool invocation, and long-conversation memory question answering. Across all three benchmarks, MemQuilt achieves empirical Pareto improvements over score-based baselines, delivering stronger task performance with lower memory-context cost. In particular, relative to Top-K, MemQuilt reduces memory-context tokens by 38.8%, 66.2%, and 18.2%, while improving SlotAcc on MS-TOD and F1 on Mem2Act and LoCoMo by 11.17, 5.29, and 12.44 percentage points, respectively. These gains persist across context budgets and alternative selection models. Component ablations further show that MemQuilt's performance–cost balance benefits from individual value, diminishing returns, and threshold stopping. To summarize, our contributions are:

1. **A task-conditioned formulation of set-aware memory selection.** We extend pointwise ranking to set-conditioned marginal utility under a hard token budget, explicitly distinguishing semantic diversity from complementarity with respect to task requirements.
2. **An explicit requirement-driven and context-budget-aware selection framework.** MemQuilt identifies missing requirements, estimates fixed requirement-support vectors, and combines individual value with diminishing-return requirement coverage through a submodular objective. Its cost-aware greedy selection treats the context budget as an active signal, while threshold stopping determines when further context is no longer worthwhile.
3. **A systematic evaluation of task performance and memory-context cost across diverse memory settings.** We evaluate MemQuilt in task-oriented dialogue, tool invocation, and long-conversation memory question answering, supported by fixed-budget analyses, controlled variants, evaluations with alternative selection models, and case studies.

2 Related Work

Memory Retrieval and Selection in LLM Agents. Long-term memory systems preserve conversational facts, reflections, trajectories, and reusable workflows to support agents across interactions (Wang et al., 2023b; Lee et al., 2024; Wang et al., 2023a; Zhong et al., 2024; Shinn et al., 2023; Zhao et al., 2024; Wang et al., 2024). Recent systems further organize accumulated experience through multi-granularity associations or compact knowledge-centric representations to improve subsequent retrieval and reuse (Xu et al., 2026b;a; Yang et al., 2026). Stored memories become useful only when relevant information is brought into the agent's current working context. This process commonly involves two stages. Candidate retrieval first constructs a small pool from the memory bank, after which memory selection determines which candidates enter the memory context. For our purposes, existing selection methods can be broadly organized into *score-based selection* and *model-directed selection*. Score-based methods assign individual values derived from relevance, utility, or task-specific signals to explicit candidates and select memories using predefined rules (Tan et al., 2025; Kang et al., 2025; Du et al., 2026). Model-directed methods instead delegate inclusion decisions to an LLM or learned policy, either over a retrieved pool or during iterative retrieval (Yan et al., 2026; Banerjee et al., 2026). Our work follows the *score-based selection* route since it preserves an inspectable selection criterion and direct control over the context budget.

Set-Aware Memory Selection. Recent score-based methods enrich per-memory assessment beyond relevance alone. RMM learns retrieval scores from response citations, while MemGuide identifies missing dialogue slots and evaluates whether each memory can supply them (Tan et al., 2025; Du et al., 2026). These methods provide more informative individual values, but pointwise ranking does not account for evidence already contained in the selected set. MMR introduces set awareness by balancing query relevance against semantic redundancy and has been adopted as a reranker in agent-memory systems such as Zep (Carbonell & Goldstein, 1998; Rasmussen et al., 2025). More generally, submodular optimization provides a principled framework for informative subset selection under diminishing returns (Krause & Golovin, 2014). PACMS applies this principle to agent-context selection through query-weighted facility location, favoring token-bounded subsets that represent relevant candidates in embedding space (Ghulyani et al., 2026). These objectives make the final choice set-aware, but characterize complementarity primarily through semantic similarity. CompassMem takes a different, model-directed route by organizing memories as an event graph and progressively gathering evidence through LLM navigation (Hu et al., 2026). In our work, rather than making memory inclusion an implicit outcome of model inference, MemQuilt uses model judgments to estimate fixed *requirement-support vectors* and applies an *explicit set objective* based on marginal support for unresolved task requirements.

Memory Selection under Context Budgets. Existing memory-selection methods commonly control memory-context size through a fixed number of returned items, a token limit, hierarchical filtering, or query-dependent retrieval depth (Kang et al., 2025; Du et al., 2026; Zhang et al., 2026; Liu et al., 2026). Prompt-compression methods instead shorten retrieved content through selective augmentation or token-level compression (Xu et al., 2024; Jiang et al., 2023). Although both directions reduce model input, they operate at different levels. Memory selection determines which complete memory units enter the context, whereas prompt compression rewrites or prunes content after retrieval. Rather than treating the context budget only as an output limit, MemQuilt uses it as an *active selection signal*, selecting memories according to cost-adjusted conditional task gain and stopping when further context is no longer worthwhile.

3 Preliminaries

3.1 Memory Context Construction in LLM Agents

Long-Term Memory and Working Context. Large language model agents retain information from past interactions in an external long-term memory bank, allowing historical information to be reused across tasks (Packer et al., 2023; Tan et al., 2025). The memory bank serves as persistent storage, whereas an agent typically needs only a task-specific portion of this history during execution. We refer to the historical information selected and organized for the current task as the *memory context*. The agent receives this memory context together with *non-memory content*, including sys-

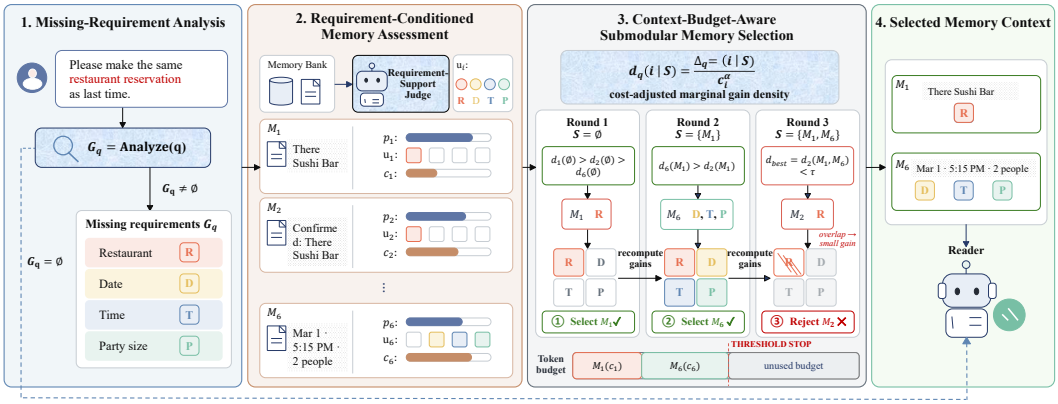

Figure 2: Overview of MemQuilt. The framework identifies missing task requirements, estimates how each candidate memory supports them, and performs budget-feasible submodular selection using cost-adjusted conditional gains and adaptive threshold stopping.

tem instructions, the current query, the dialogue state, and tool descriptions. Together, the memory context and non-memory content form the agent's *working context* for the current task.

Retrieve-then-Select Pipeline. Many memory systems construct the memory context through two successive stages: *candidate retrieval* and *memory selection* (Tan et al., 2025; Du et al., 2026). Candidate retrieval identifies potentially relevant memory units from the long-term memory bank and forms a finite candidate pool, commonly using signals such as embedding similarity. Memory selection then determines which units from this fixed candidate pool should be retained for the current task. The selected memories are subsequently ordered and formatted to form the memory context. The complete process can be summarized as *long-term memory bank* $\rightarrow$ *candidate pool* $\rightarrow$ *selected memory set* $\rightarrow$ *memory context* $\rightarrow$ *working context*. In this work, upstream candidate retrieval is fixed, and we study the subsequent memory-selection stage.

3.2 Submodularity and Budgeted Maximization

Submodularity characterizes diminishing returns in set-valued objectives and provides a principled foundation for informative subset selection (Krause & Golovin, 2014). Let V be a finite ground set and $f : 2^V \rightarrow \mathbb{R}$ a set function.

Definition 3.1 (Submodular Function). A set function f is submodular if, for any $A \subseteq B \subseteq V$ and $e \in V \setminus B$,

$$f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B). \tag{1}$$

This inequality formalizes diminishing returns: adding an element to a smaller set provides at least as much additional value as adding it to a larger set.

Definition 3.2 (Marginal Gain). For $S \subseteq V$ and $e \in V \setminus S$, the marginal gain of e with respect to S is

$$\Delta_f(e \mid S) = f(S \cup \{e\}) - f(S). \tag{2}$$

Under submodularity, $\Delta_f(e \mid S)$ cannot increase as S grows, allowing an objective to assign less additional value to information already represented by the selected set.

Budgeted Submodular Maximization. Maximizing a monotone submodular function under a budget constraint is NP-hard. For heterogeneous costs $c_e > 0$, marginal gain density $\Delta_f(e \mid S)/c_e$ compares incremental value with resource consumption (Lin & Bilmes, 2010). Appendix B.2 discusses classical guarantees and their relation to MemQuilt.

4 Method

MemQuilt operates at the memory-selection stage of the retrieve-then-select pipeline introduced in Section 3.1. Given the candidate pool produced by fixed upstream retrieval, it proceeds in

three stages, as illustrated in Fig. 2: *missing-requirement analysis* identifies the information still needed from memory, *requirement-conditioned memory assessment* estimates how each candidate supports these requirements, and *context-budget-aware submodular selection* constructs the final set by jointly considering individual value, requirement coverage, and memory-context cost.

Let q denote the current task state and V_q the fixed candidate pool returned by upstream retrieval. MemQuilt selects a set $S \subseteq V_q$, where each candidate i has memory-context cost $c_i > 0$ and $c(S) = \sum_{i \in S} c_i$. Given a memory-context budget $B_q > 0$, we formulate the selection problem for a task requiring memory as

$$S_q^\star \in \arg\max_{S \subseteq V_q} F_q(S) \qquad \text{s.t.} \qquad c(S) \leq B_q. \tag{3}$$

Here, $F_q(S)$ is a task-dependent surrogate for the collective value of the selected memory set, and the hard budget applies only to the selected memory content.

4.1 Missing-Requirement Analysis

MemQuilt begins by identifying the information that the current task still needs from historical memory. Following MemGuide's use of missing-slot reasoning to guide memory selection (Du et al., 2026), we generalize such information needs as *task requirements*. Depending on the task, these requirements may correspond to unfilled dialogue slots, missing tool arguments, or atomic information needed to answer a question. An LLM-based requirement analyzer takes the current task state q as input and produces the active requirement set

$$G_q = \text{Analyze}(q) = \{g_1, \ldots, g_m\}. \tag{4}$$

Each g_j describes one piece of information that must be recovered from memory. For tasks with an explicit specification, the analyzer compares the required fields against the information already available and retains those requiring historical evidence. For question answering, it decomposes the question into a minimal set of independently checkable information requirements while preserving the entities, relations, and temporal constraints expressed in the question. For example, the request "Please make the same restaurant reservation as last time" may require the restaurant, date, time, and party size to be recovered from previous interactions. If $G_q = \varnothing$, MemQuilt returns $S = \varnothing$ before candidate assessment, implementing a no-memory decision distinct from threshold stopping during set construction. Otherwise, G_q provides a shared task-conditioned reference for assessing all candidate memories.

4.2 Requirement-Conditioned Memory Assessment

Given G_q, MemQuilt assesses what evidence each candidate provides for the active requirements. For each candidate $i \in V_q$, an LLM-based *requirement-conditioned memory judge* produces a *requirement-support vector*

$$\mathbf{u}_i = \text{Judge}_q(i, G_q) = \left(u_{ig_1}, \ldots, u_{ig_m}\right)^\top \in [0,1]^m. \tag{5}$$

Each component u_{ig} estimates how strongly memory i supports requirement g. Higher scores indicate concrete and usable evidence, while lower scores reflect weaker or less directly applicable information. In the reservation example, a memory identifying the restaurant and another recording the date, time, and party size support different coordinates. These shared requirement coordinates make candidates directly comparable and allow the selector to track the evidence accumulated for each requirement. Alongside this requirement-level representation, MemQuilt retains a fixed *individual value*

$$p_i = \eta \widetilde{r}_i + (1 - \eta) a_i, \tag{6}$$

where $\widetilde{r}_i \in [0,1]$ is normalized retrieval relevance, $a_i \in [0,1]$ is an optional scalar task-utility score, and $\eta \in [0,1]$ controls their relative contribution. Setting $\eta = 1$ uses retrieval relevance alone, while $\eta < 1$ allows an available task-utility signal to contribute to the individual value. The individual value summarizes a candidate's overall usefulness, whereas its requirement-support vector specifies the evidence it provides for each active requirement. Both assessments are computed before selection and remain fixed. The selector subsequently combines them through set-conditioned marginal utility, which changes as the selected set grows.

4.3 CONTEXT-BUDGET-AWARE SUBMODULAR MEMORY SELECTION

From Candidate Assessments to Set Value Candidate assessment describes what each memory provides, while selection must additionally account for what the current set already contains. Evidence for an unsupported requirement should contribute more than repeated support for a requirement already well represented. Let $z_g(S) = \sum_{i \in S} u_{ig}$ denote the support accumulated by the selected set for requirement g. MemQuilt applies a concave transformation to this accumulated support and combines the resulting requirement coverage with individual value

$$F_q(S) = \sum_{i \in S} p_i + \lambda \sum_{g \in G_q} w_g \left(\sum_{i \in S} u_{ig} \right)^{\rho}, \qquad 0 < \rho \leq 1. \tag{7}$$

Here, $\lambda \geq 0$ controls the contribution of requirement coverage, and $w_g \geq 0$ weights individual requirements. We use $w_g = 1$ throughout our experiments. When $0 < \rho < 1$, the same amount of additional support contributes less to a requirement that is already well supported. For example, once the restaurant is represented, another memory naming the same restaurant receives less additional coverage value, while evidence about an unsupported date or time retains a larger contribution. Unlike coverage that saturates after one supporting memory, this formulation still allows supplementary evidence for the same requirement to contribute. When $\rho = 1$, the coverage term becomes additive and the objective is modular. For an unselected memory $i \notin S$, its conditional marginal gain is

$$\begin{aligned} \Delta_q(i \mid S) &= F_q(S \cup \{i\}) - F_q(S) \\ &= p_i + \lambda \sum_{g \in G_q} w_g \left[\left(z_g(S) + u_{ig} \right)^{\rho} - z_g(S)^{\rho} \right]. \end{aligned} \tag{8}$$

The individual-value term remains fixed, while the coverage term depends on the support already accumulated by S. A candidate may therefore lose priority as selected memories provide overlapping requirement support, even though its original assessments remain unchanged. Since x^{ρ} is nondecreasing and concave for $0 < \rho \leq 1$, F_q is normalized, monotone nondecreasing, and submodular under the nonnegative scores and weights defined above. Together with the hard budget in Eq. 3, the selection problem is monotone submodular maximization under a single knapsack constraint. Appendix B.1 and Appendix B.2 provide the proofs and clarify the scope of the classical approximation results.

Cost-Aware Greedy Selection The set objective compares the contributions of candidates, but selection must also account for the context they occupy. Starting from $S = \varnothing$, MemQuilt evaluates each candidate using the cost-adjusted gain density

$$d_q(i \mid S) = \frac{\Delta_q(i \mid S)}{c_i^{\alpha}}, \qquad \alpha \geq 0. \tag{9}$$

The exponent α controls sensitivity to memory-context cost. Setting $\alpha = 1$ yields marginal gain per unit cost, while $\alpha = 0$ leaves cost to the hard budget alone. At each step, MemQuilt selects the budget-feasible candidate $i^{\star}$ with the highest gain density and accepts it only if $\Delta_q(i^{\star} \mid S) > 0$ and $d_q(i^{\star} \mid S) \geq \tau$. It then updates the accumulated support and repeats until no candidate fits or the best feasible candidate fails the acceptance test. Since candidate assessments remain fixed, the greedy loop requires no additional model calls. Algorithm 1 provides the complete procedure, including deterministic tie-breaking and output construction.

Adaptive Stopping and Theoretical Properties The hard budget caps the memory context but need not be exhausted. Since F_q is submodular and candidate costs are fixed, each remaining candidate's gain density cannot increase as the selected set grows. The threshold τ therefore stops selection when the best feasible candidate no longer justifies additional context. For $\alpha = 1$, this test corresponds to a nonnegative marginal contribution to the cost-penalized objective $F_q(S) - \tau c(S)$. More generally, threshold stopping provides a residual-value certificate for unused budget. Appendix B.3–B.4 provide the derivations and scope, while Appendix B.5 establishes budget feasibility and computational complexity.

Table 1: Cross-benchmark task performance and memory-context cost. All methods use relevance-based individual values and share a per-query budget matched to Retriever Top-K, where $K = 5$.

(a) Task performance

	MS-TOD			Mem2Act		LoCoMo
Method	LLMScore ↑	SlotAcc ↑	TaskSucc ↑	F1 ↑	BLEU-1 ↑	F1 ↑
No Memory	6.912	0.490	0.332	–	–	–
Top-K	6.975	0.640	0.451	0.362	0.522	0.507
MMR	6.716	0.607	0.388	0.378	0.534	0.509
Dedup	6.780	0.625	0.415	0.363	0.524	0.488
MemQuilt (Ours)	**7.548**	**0.752**	**0.520**	**0.414**	**0.570**	**0.632**

(b) Memory-context cost

	MS-TOD		Mem2Act		LoCoMo	
Method	MemTok ↓	#Mem ↓	MemTok ↓	#Mem ↓	MemTok ↓	#Mem ↓
No Memory	0.00	0.00	–	–	–	–
Top-K	45.52	5.00	98.51	5.00	268.88	5.00
MMR	44.77	5.40	97.07	4.99	264.88	4.90
Dedup	44.57	5.37	97.20	5.02	262.22	4.95
MemQuilt (Ours)	**27.86**	**4.36**	**33.31**	**1.88**	**220.07**	**4.26**

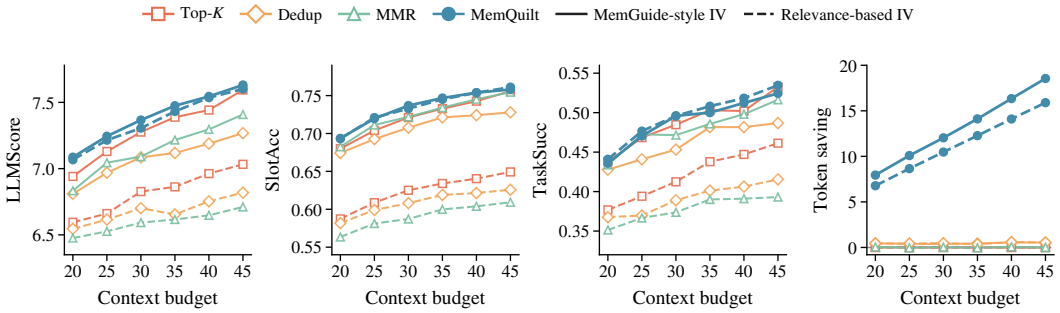

Figure 3: Task performance and memory-context use across budgets on MS-TOD. Solid curves use MemGuide-style individual values, whereas dashed curves use relevance-based individual values. Token saving is the context budget minus MemTok. All metrics are higher-is-better.

5 Experiments

5.1 Experimental Setup

Benchmarks. We evaluate MemQuilt on three benchmarks with different memory units and evidence structures: MS-TOD (Du et al., 2026), Mem2Act (Shen et al., 2026), and LoCoMo (Maharana et al., 2024). MS-TOD serves as our primary benchmark and evaluates whether a selector can compose historical memories to cover the slot values required by a task-oriented dialogue. Mem2Act and LoCoMo extend the evaluation to memory-grounded tool invocation and question answering over long conversations, respectively. Dataset construction, retrieval protocols, and prompt templates are detailed in Appendix C.1–C.3.

Baselines. We compare MemQuilt with representative score-based memory-selection methods. **Top-K** performs pointwise ranking by individual value. **MMR** performs similarity-based set-aware selection by balancing relevance against redundancy, whereas **Dedup** applies similarity-based redundancy filtering. To isolate MemQuilt's design components, we evaluate three single-change variants: (1) **MemQuilt w/o IV** sets $p_i = 0$, (2) **MemQuilt w/o DR** sets $\rho = 1$, and (3) **MemQuilt w/o TS** sets $\tau = 0$. In the cross-benchmark comparison, retrieval relevance serves as the individual value for all applicable methods. On MS-TOD, we additionally instantiate a_i using the task-specific utility defined by MemGuide and report this setting separately as the **MemGuide-style individual**

Table 2: Results on MS-TOD with MemGuide-style individual values. All methods share the candidate pool, context budget, Reader, and evaluator used in the relevance-based comparison.

Method	LLMScore ↑	SlotAcc ↑	TaskSucc ↑	MemTok ↓	#Mem ↓
No Memory	6.912	0.490	0.332	0.00	0.00
Top-K	7.607	0.749	0.526	44.26	5.00
MMR	7.403	0.751	0.516	43.60	5.45
Dedup	7.200	0.724	0.482	43.25	5.34
MemQuilt (Ours)	**7.615**	**0.761**	**0.528**	**24.77**	**3.15**

value. We also include **No Memory** on MS-TOD to measure the contribution of long-term memory. Formal definitions and hyperparameters are provided in Appendix C.4. Since model-directed set selection tends to soften the hard budget constraint, we analyze this route separately in Appendix C.6.

Evaluation metrics and protocol. Following the corresponding benchmark protocols, we report **LLMScore**, slot accuracy (**SlotAcc**), and task success (**TaskSucc**) on MS-TOD, **F1** and **BLEU-1** on Mem2Act, and F1 on LoCoMo. We additionally report selected memory-context tokens (**MemTok**) and the number of selected memories (**#Mem**) to measure memory-context cost. For every query, all methods select from the same fixed candidate pool under the same budget. Selected memories are restored to chronological order before being provided to the Reader. In the main comparison, the per-query budget is matched to the actual memory-context cost of Retriever Top-K, where $K = 5$. Unless otherwise specified, all model-based Analyzers, Memory Judges, Readers, and the MS-TOD LLMScore Evaluator use Gemini-3.8-Flash with the default reasoning configuration. Throughout the experiments, displayed values are rounded, while reported differences are computed from unrounded results. Metric definitions and computation details are provided in Appendix C.5.

5.2 Overall Effectiveness

MemQuilt consistently improves task performance while reducing memory-context cost across benchmarks. As shown in Table 1 and Fig. 4, MemQuilt consistently occupies the favorable upper-left region across all three benchmarks. Relative to Top-K, MemQuilt reduces MemTok by 38.8%, 66.2%, and 18.2% on MS-TOD, Mem2Act, and LoCoMo, respectively. Meanwhile, it improves SlotAcc and TaskSucc on MS-TOD by 11.17 and 6.92 percentage points, F1 and BLEU-1 on Mem2Act by 5.29 and 4.79 percentage points, and F1 on LoCoMo by 12.44 percentage points. In contrast, MMR and Dedup retain more than 97% of the Top-K context while yielding only limited and non-uniform improvements across benchmarks. These results demonstrate a stronger task performance–memory-context trade-off across different memory units and evidence structures. Further analyses are provided in Appendix D.

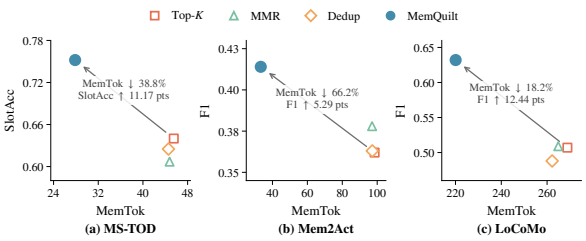

Figure 4: Task performance versus memory-context cost across benchmarks. All points use relevance-based individual values, and arrows show the change from Top-K to MemQuilt using unrounded results.

Set-level modeling remains beneficial even with MemGuide-style individual values. Comparing Table 1 with Table 2 shows that MemGuide-style individual values improve all compared methods over their relevance-based counterparts. Under the same MemGuide-style individual values, MemQuilt further improves SlotAcc by 1.21 percentage points and reduces MemTok by 44.0% relative to Top-K. More notably, as shown in Fig. 3, relevance-based MemQuilt achieves higher LLMScore and SlotAcc than MemGuide-style Top-K at every budget while obtaining comparable TaskSucc and reducing MemTok by 6.86–15.97. Mem2Act and LoCoMo show the same pattern, as detailed in Appendix D. These results indicate that stronger individual values are useful but cannot replace requirement-aware set-level optimization.

MemQuilt preserves its performance–cost advantage across both tight and ample context budgets. Since MS-TOD involves multiple active slot requirements, we evaluate budgets from 20 to 45,

Table 3: Component ablations of MemQuilt on MS-TOD.

Method	LLMScore ↑	SlotAcc ↑	TaskSucc ↑	MemTok ↓	#Mem ↓
MemQuilt w/o IV	7.549	0.748	0.513	**23.86**	**3.16**
MemQuilt w/o DR	7.393	0.732	0.490	27.84	4.37
MemQuilt w/o TS	**7.550**	**0.752**	0.519	28.71	4.50
MemQuilt	7.548	**0.752**	**0.520**	27.86	4.36

with supporting statistics provided in Appendix D.3. As shown in Fig. 3, relative to the corresponding Top-K baselines, relevance-based MemQuilt improves SlotAcc by 10.58–11.33 percentage points and TaskSucc by 6.42–8.35 percentage points while reducing context use by approximately 35%. Relative to MemGuide-style Top-K, MemQuilt with MemGuide-style individual values saves approximately 41% of the context while maintaining comparable task performance. MMR and Dedup nearly exhaust the available budget without providing consistent gains. This contrast supports selection based on conditional marginal gains and adaptive threshold stopping rather than filling a predetermined context budget.

5.3 Ablation Studies and Theory-Guided Analysis

MemQuilt's ability to balance task performance and memory-context cost simultaneously benefits from individual value, diminishing returns, and threshold stopping. As shown in Table 3, removing DR reduces SlotAcc and TaskSucc by 2.03 and 3.05 percentage points with nearly unchanged MemTok, demonstrating the contribution of diminishing-return requirement coverage. Removing IV saves 14.4% of the memory context but lowers both SlotAcc and TaskSucc. Removing TS increases MemTok by 3.0% without improving task performance. Together, the three components allow MemQuilt to preserve task quality while avoiding unnecessary context. Results on Mem2Act and LoCoMo show the same performance–cost pattern, as detailed in Appendix E.

MemQuilt exhibits the selection behavior implied by its objective. As shown in Fig. 5, the median normalized conditional gain density decreases sharply along the diagnostic greedy trajectory and falls below the stopping threshold after the fifth step. Meanwhile, MemQuilt's budget utilization increases from 0.67 for one active requirement to 0.94 for five or more, whereas Top-K, MMR, and Dedup consistently consume nearly the full budget. These results are empirically consistent with diminishing returns and show that MemQuilt adapts context allocation to task demand rather than treating the budget as a target to exhaust. Case studies further illustrate how MemQuilt adapts memory selection to distinct evidence structures across the three benchmarks, as detailed in Appendix G.

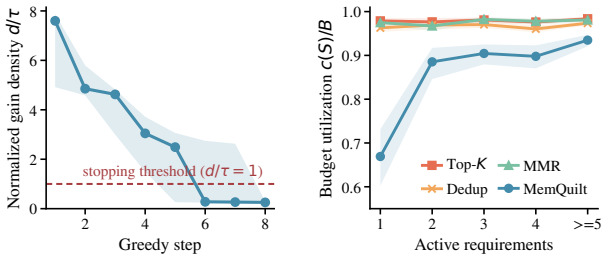

Figure 5: Selection dynamics on MS-TOD. Shading denotes the interquartile range for normalized gain density and 95% confidence intervals for budget utilization. Values below 1 indicate rejected diagnostic additions. Details are provided in Appendix F.

6 Conclusion

In this paper, we investigate how LLM agents can construct task-relevant memory context under constrained context budgets. We observe that pointwise selection can retain overlapping evidence while overlooking information still needed for the current task. Building on this observation, we propose MemQuilt, a context-budget-aware submodular framework that identifies missing requirements, assesses how each candidate supports them, and performs cost-aware set selection with adaptive threshold stopping. Experiments on MS-TOD, Mem2Act, and LoCoMo demonstrate that MemQuilt consistently achieves empirical Pareto improvements over score-based baselines, improving task performance while using less memory context. Our theoretical and empirical analyses further validate the roles of diminishing-return requirement coverage and adaptive stopping in efficient memory-context construction across heterogeneous tasks, memory units, and evidence structures.

AI USE STATEMENT

In this work, we used generative AI tools to assist with research ideation, experimental scripting and debugging, literature retrieval, result interpretation, and drafting and polishing portions of the manuscript. All AI-assisted outputs were reviewed and verified by the authors. The authors take full responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

To support reproducibility, we submit anonymized source code implementing MemQuilt and the experimental pipeline, including baseline configurations and evaluation scripts, as supplementary material. The complete selection algorithm and theoretical analysis are presented in Appendices A and B, respectively. Dataset construction, retrieval protocols, prompts, implementation details, hyperparameters, and metric definitions are documented in Appendix C. Additional results and analyses are reported in Appendices D–G.

REFERENCES

Pratyay Banerjee, Masud Moshtaghi, Shivashankar Subramanian, Amita Misra, and Ankit Chadha. Apex-mem: Agentic semi-structured memory with temporal reasoning for long-term conversational ai. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16470–16489, 2026.

Jaime Carbonell and Jade Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, pp. 335–336, New York, NY, USA, 1998. Association for Computing Machinery. ISBN 1581130155. doi: 10.1145/290941.291025. URL https://doi.org/10.1145/290941.291025.

Yiming Du, Bingbing Wang, Yang He, Bin Liang, Baojun Wang, Zhongyang Li, Lin Gui, Jeff Z Pan, Ruifeng Xu, and Kam-Fai Wong. Memguide: Intent-driven memory selection for goal-oriented multi-session llm agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 30584–30592, 2026.

Manu Ghulyani, Arunabh Singh, Karan Bharadwaj, Ankit Nath, and Suranjan Goswami. Pacms: Submodular context selection as a pluggable engine for llm agents. *arXiv preprint arXiv:2606.20047*, 2026.

Zexue He, Yu Wang, Churan Zhi, Yuanzhe Hu, Tzu-Ping Chen, Lang Yin, Ze Chen, Tong Arthur Wu, Siru Ouyang, Zihan Wang, et al. Memoryarena: Benchmarking agent memory in interdependent multi-session agentic tasks. *arXiv preprint arXiv:2602.16313*, 2026.

Yuyang Hu, Jiongnan Liu, Jiejun Tan, Yutao Zhu, and Zhicheng Dou. Memory matters more: Event-centric memory as a logic map for agent searching and reasoning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 22389–22407, 2026.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 13358–13376, 2023.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory os of ai agent. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 25972–25981, 2025.

Andreas Krause and Daniel Golovin. Submodular function maximization. *Tractability*, 3(71-104): 3, 2014.

Kuang-Huei Lee, Xinyun Chen, Hiroki Furuta, John Canny, and Ian Fischer. A human-inspired reading agent with gist memory of very long contexts. *arXiv preprint arXiv:2402.09727*, 2024.

Hui Lin and Jeff Bilmes. Multi-document summarization via budgeted maximization of submodular functions. In *Human Language Technologies: The 2010 Annual conference of the North American chapter of the association for computational linguistics*, pp. 912–920, 2010.

Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. Simplemem: Efficient lifelong memory for llm agents. *arXiv preprint arXiv:2601.02553*, 2026.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13851–13870, 2024.

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

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Xufang Luo, Hao Cheng, Dongsheng Li, Yuqing Yang, Chin-Yew Lin, H Vicky Zhao, Lili Qiu, et al. Secom: On memory construction and retrieval for personalized conversational agents. In *International Conference on Learning Representations*, volume 2025, pp. 91851–91885, 2025.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: a temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025.

Yiting Shen, Kun Li, Wei Zhou, and Songlin Hu. Mem2actbench: A benchmark for evaluating long-term memory utilization in task-oriented autonomous agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8173–8190, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32(1):41–43, 2004.

Zhen Tan, Jun Yan, I-Hung Hsu, Rujun Han, Zifeng Wang, Long Le, Yiwen Song, Yanfei Chen, Hamid Palangi, George Lee, et al. In prospect and retrospect: Reflective memory management for long-term personalized dialogue agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8416–8439, 2025.

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

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. Augmenting language models with long-term memory. *Advances in neural information processing systems*, 36:74530–74543, 2023b.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

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

Derong Xu, Yi Wen, Pengyue Jia, Yingyi Zhang, Wenlin Zhang, Yichao Wang, Huifeng Guo, Ruiming Tang, Xiangyu Zhao, Enhong Chen, et al. From single to multi-granularity: Toward long-term memory association and selection of conversational agents. In *International Conference on Learning Representations*, volume 2026, pp. 91581–91617, 2026a.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. Recomp: Improving retrieval-augmented lms with context compression and selective augmentation. In *International Conference on Learning Representations*, volume 2024, pp. 43478–43502, 2024.

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

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z Pan, et al. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12805–12825, 2026.

Ke Yang, Zixi Chen, Xuan He, Jize Jiang, Michel Galley, Chenglong Wang, Jianfeng Gao, Jiawei Han, and ChengXiang Zhai. Plugmem: A task-agnostic plugin memory module for llm agents. *arXiv preprint arXiv:2603.03296*, 2026.

Jiaquan Zhang, Chaoning Zhang, Shuxu Chen, Zhenzhen Huang, Pengcheng Zheng, Zhicheng Wang, Ping Guo, Fan Mo, Sung-Ho Bae, Jie Zou, et al. Lightweight llm agent memory with small language models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12914–12929, 2026.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

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

CONTENTS OF SUPPLEMENTARY MATERIALS FOR MEMQUILT

A Complete Selection Algorithm **14**

B Analysis of Budgeted Memory Selection **14**

B.1 Monotonicity and Submodularity 14

B.2 Budgeted Maximization and Approximation Scope 15

B.3 A Cost-Penalty Interpretation 15

B.4 Residual Value after Threshold Stopping 16

B.5 Budget Feasibility and Complexity 16

C Experimental Details **17**

C.1 MS-TOD Details 17

C.2 Mem2Act Details 20

C.3 LoCoMo Details 22

C.4 Baseline and Ablation Implementations and Hyperparameters 25

C.5 Evaluation Metrics 27

C.6 Controlled Comparison with Model-Directed Selection 29

D Additional Effectiveness Analysis **31**

D.1 Cross-Benchmark Results and Evidence Structures 31

D.2 Results by Memory Requirement 33

D.3 Context-Budget Analysis 34

D.4 Complete Results Across Context Budgets 34

E Component Ablations Across Benchmarks **35**

E.1 Mem2Act Component Ablation 35

E.2 LoCoMo Component Ablation 36

F Theory-Guided Analysis Details **37**

F.1 Data and Shared Protocol 37

F.2 Conditional-Gain Trajectories 37

F.3 Budget Utilization by Task Demand 37

G Case Studies **38**

G.1 MS-TOD: Composing Distributed Slot Evidence 38

G.2 Mem2Act: Isolating an Argument-Bearing Fact 40

G.3 LoCoMo: Aggregating Complementary Conversation Turns 41

Algorithm 1 MemQuilt: Context-Budget-Aware Submodular Memory Selection

Input: Current task state q; retrieved candidate pool V_q; memory-context costs $\{c_i\}_{i\in V_q}$ with $c_i > 0$; normalized retrieval relevance scores $\{\widetilde{r}_i\}_{i\in V_q}$; optional scalar task-utility scores $\{a_i\}_{i\in V_q}$; memory-context budget $B_q > 0$; individual-value mixing weight η; requirement-coverage weight λ; coverage exponent ρ; cost exponent α; gain-density stopping threshold τ.
When scalar task utility is unused, set $\eta = 1$ and $a_i = 0$ for all $i \in V_q$.

Output: Selected memory set $S \subseteq V_q$ for constructing the memory context, with $c(S) = \sum_{i\in S} c_i \le B_q$.

Identify active requirements $G_q \leftarrow \mathrm{Analyze}(q)$ from the current task state.
if $G_q = \varnothing$ **then**
  return $\varnothing$ ▷ No active memory requirements
end if
for all candidate memory $i \in V_q$ **do**
  Compute the requirement-support vector $\mathbf{u}_i \leftarrow \mathrm{Judge}_q(i, G_q)$.
  Compute individual value $p_i \leftarrow \eta\widetilde{r}_i + (1-\eta)a_i$.
end for
Initialize selected memories $S \leftarrow \varnothing$ and used memory-context budget $b \leftarrow 0$.
Initialize cumulative support $z_g \leftarrow 0$ and requirement weights $w_g \leftarrow 1$ for all $g \in G_q$.
while true do
  Form the budget-feasible candidate set $\mathcal{A} \leftarrow \{i \in V_q \setminus S : b + c_i \le B_q\}$.
  if $\mathcal{A} = \varnothing$ **then**
    break ▷ No budget-feasible candidate
  end if
  Evaluate conditional marginal gains $\Delta_q(i \mid S)$ using Eq. 8 for all $i \in \mathcal{A}$.
  Compute cost-adjusted gain densities $d_q(i \mid S) \leftarrow \Delta_q(i \mid S)/c_i^{\alpha}$ for all $i \in \mathcal{A}$.
  Select $i^\star \leftarrow \arg\max_{i\in\mathcal{A}} d_q(i \mid S)$ using deterministic tie-breaking.
  if $\Delta_q(i^\star \mid S) \le 0$ **or** $d_q(i^\star \mid S) < \tau$ **then**
    break ▷ Context-budget-aware stopping
  end if
  Add $i^\star$ to the selected memory set: $S \leftarrow S \cup \{i^\star\}$.
  Update used memory-context budget $b \leftarrow b + c_{i^\star}$.
  Update cumulative requirement support $z_g \leftarrow z_g + u_{i^\star g}$ for all $g \in G_q$.
end while
return selected memory set S.

A Complete Selection Algorithm

This appendix gives the complete procedure that transforms the candidate pool into a selected memory set. The requirement analyzer and memory judge are called before the greedy loop, and their outputs remain fixed throughout selection. The selected memories are subsequently ordered and formatted to construct the memory context supplied to the downstream agent.

B Analysis of Budgeted Memory Selection

The following analysis conditions on a fixed query q, candidate pool V_q, and requirement set G_q. All individual values, support estimates, requirement weights, and memory costs are fixed before selection. We assume $p_i, u_{ig}, w_g, \lambda \ge 0$, $c_i > 0$, $0 < \rho \le 1$, and $\alpha, \tau \ge 0$. The results concern the surrogate objective in Eq. 7. They do not require the downstream agent's task performance to be monotone or submodular.

B.1 Monotonicity and Submodularity

Proposition B.1. The objective F_q in Eq. 7 is normalized, nonnegative, monotone nondecreasing, and submodular.

Proof. An empty set has zero individual value and zero accumulated support, so $F_q(\varnothing) = 0$. Nonnegativity follows from the nonnegative coefficients and scores. If $A \subseteq D$, then $z_g(A) \le z_g(D)$

for every requirement. Since x^ρ is nondecreasing on $[0, \infty)$, both the individual-value term and each coverage term are nondecreasing under set inclusion. It remains to establish diminishing returns. For any fixed $u \geq 0$, concavity of x^ρ implies that $h_u(x) = (x + u)^\rho - x^\rho$ is nonincreasing for $x \geq 0$. Consequently, for $A \subseteq D \subseteq V_q$ and $i \in V_q \setminus D$,

$$\begin{aligned} \Delta_q(i \mid A) &= p_i + \lambda \sum_g w_g h_{u_{ig}}\big(z_g(A)\big) \\ &\geq p_i + \lambda \sum_g w_g h_{u_{ig}}\big(z_g(D)\big) \\ &= \Delta_q(i \mid D). \end{aligned} \tag{10}$$

This is the diminishing-returns definition of submodularity. The argument also covers $\rho = 1$, when $h_u(x) = u$ and the objective is modular. Dividing Eq. 10 by the fixed positive quantity c_i^α gives $d_q(i \mid A) \geq d_q(i \mid D)$. Thus, the gain density of each remaining candidate is nonincreasing. Along the greedy trajectory, the feasible candidate sets also shrink, so their maximum density cannot increase while they remain nonempty. This property does not imply that the threshold must be reached before the budget becomes binding. In particular, the fixed contribution p_i/c_i^α can keep a candidate's density above the threshold.

B.2 Budgeted Maximization and Approximation Scope

Proposition B.2. Under the assumptions of Proposition B.1, the fixed-budget problem in Eq. 3 is monotone submodular maximization under a single knapsack constraint.

Proof. Proposition B.1 establishes that F_q is normalized, nonnegative, monotone nondecreasing, and submodular. Since every cost $c_i > 0$ is fixed before selection and $c(S) = \sum_{i \in S} c_i$, the feasible family in Eq. 3 is defined by one additive knapsack constraint. The stated classification follows.

For this problem class, classical algorithms that combine partial enumeration with greedy completion attain a $(1 - 1/e)$ approximation ((Sviridenko, 2004)). This result motivates cost-aware greedy construction, but it applies to the complete classical algorithm rather than to any procedure that merely ranks candidates by gain density. To see the distinction, consider a modular instance with budget $L > 2$, two memories with costs 1 and L, and values $2/L$ and 1, respectively. With $\alpha = 1$ and $\tau = 0$, density greedy selects the first memory, after which the second no longer fits. Its value is $2/L$, while the optimal feasible value is 1, so the ratio approaches zero as L grows. MemQuilt additionally permits a general cost exponent and threshold stopping. We therefore claim the objective classification in Proposition B.2, budget feasibility, and the stopping properties established below, but do not claim that Algorithm 1 inherits the classical approximation ratio.

B.3 A Cost-Penalty Interpretation

The stopping threshold admits a useful interpretation in terms of a penalized objective. Define the adjusted additive cost $C_\alpha(S) = \sum_{i \in S} c_i^\alpha$ and let

$$H_{q,\tau,\alpha}(S) = F_q(S) - \tau C_\alpha(S). \tag{11}$$

The marginal contribution of candidate i to this objective is

$$\Delta H_{q,\tau,\alpha}(i \mid S) = \Delta_q(i \mid S) - \tau c_i^\alpha = c_i^\alpha \big(d_q(i \mid S) - \tau\big). \tag{12}$$

Thus, the density test $d_q(i \mid S) \geq \tau$ is equivalent to requiring a nonnegative penalized marginal gain. Ranking by d_q also gives the same ordering as ranking by $\Delta H_{q,\tau,\alpha}(i \mid S)/c_i^\alpha$, because the scores differ by the constant τ. It generally differs from ranking by the unnormalized penalized gain $\Delta H_{q,\tau,\alpha}$. Although $H_{q,\tau,\alpha}$ remains submodular after subtracting a modular cost, it need not remain monotone.

When $\alpha = 1$, the adjusted cost equals the original memory cost, $C_1(S) = c(S)$. The Lagrangian associated with the budget constraint is then

$$\mathcal{L}_q(S,\tau) = F_q(S) - \tau\big(c(S) - B_q\big) = H_{q,\tau,1}(S) + \tau B_q. \tag{13}$$

For a fixed multiplier τ, the constant τB_q does not affect marginal gains. This establishes a Lagrangian interpretation of the acceptance test when $\alpha = 1$. MemQuilt nevertheless retains the hard budget and follows forward greedy steps, so the identity does not mean that it globally optimizes the relaxed problem. For $\alpha \neq 1$, Eq. 11 penalizes the adjusted cost C_α, while feasibility is still determined by $c(S)$. The interpretation therefore concerns adjusted cost in this case, rather than the Lagrangian of the original budget constraint.

B.4 Residual Value after Threshold Stopping

Proposition B.3. Let S be a feasible selected set for which every candidate in $\mathcal{A}(S)$ satisfies $d_q(i \mid S) \leq \tau$. For any $T \subseteq V_q \setminus S$ satisfying $c(T) \leq B_q - c(S)$, the residual gain satisfies $F_q(S \cup T) - F_q(S) \leq \tau \sum_{i \in T} c_i^\alpha$.

Proof. Write $T = \{t_1, \ldots, t_r\}$ and define $T_{<j} = \{t_1, \ldots, t_{j-1}\}$. Since costs are positive and T is jointly feasible as an addition to S, every t_j is individually feasible at S. The assumed density bound therefore applies to every element of T. Telescoping the set increments and applying submodularity gives

$$\begin{aligned} F_q(S \cup T) - F_q(S) &= \sum_{j=1}^{r} \Delta_q\big(t_j \mid S \cup T_{<j}\big) \\ &\leq \sum_{j=1}^{r} \Delta_q(t_j \mid S) \\ &= \sum_{j=1}^{r} c_{t_j}^\alpha \, d_q(t_j \mid S) \\ &\leq \tau \sum_{j=1}^{r} c_{t_j}^\alpha. \end{aligned} \tag{14}$$

The bound holds trivially for $T = \varnothing$, completing the proof. For $\alpha = 1$, feasibility of T further gives

$$F_q(S \cup T) - F_q(S) \leq \tau c(T) \leq \tau\big(B_q - c(S)\big). \tag{15}$$

Strict threshold stopping in Algorithm 1 satisfies the proposition's weak inequality. The result also covers zero-gain termination: if the maximum-density feasible candidate has zero marginal gain, nonnegativity and positive costs imply zero marginal gain for every feasible candidate. If no candidate fits within the remaining budget, the only feasible addition is the empty set. These statements describe additions that retain the selected set. They do not bound improvements obtained by removing or replacing selected memories, and they do not provide a gap to the global optimum of Eq. 3. Equivalently, Proposition B.3 implies that no feasible addition can improve the adjusted penalized objective in Eq. 11 after threshold stopping. This is an optimality statement with respect to additions only, not a guarantee of a globally Pareto-optimal memory set.

B.5 Budget Feasibility and Complexity

Algorithm 1 in Appendix A specifies the complete selection procedure. The requirement analyzer and memory judge are used before the greedy loop. Their outputs remain fixed, and only accumulated support and marginal gains change during selection. Returning an empty set when $G_q = \varnothing$ is an activation rule based on the analyzer's output. It is separate from the submodular optimization argument. Without this rule, Eq. 7 would reduce to its individual-value term for an empty requirement set and could still favor a nonempty selection.

Every addition is drawn from the feasible set, so the algorithm always respects the budget. At most $|V_q|$ memories can be selected. Let $n = |V_q|$, $m = |G_q| > 0$, and $k = |S|$. With accumu-

lated support stored explicitly, one marginal-gain evaluation takes $O(m)$ time. Direct recomputation therefore costs $O((k+1)nm)$, including a possible final unsuccessful scan, with $O(nm)$ storage for the support matrix. This complexity excludes candidate retrieval, model-based assessment, and downstream inference. The density monotonicity established in Appendix B.1 also allows cached densities to serve as upper bounds in a lazy implementation, provided that exact reevaluation and tie-breaking are preserved. In our evaluation, the selected memories are presented in their original chronological order under the same formatting convention across methods. The downstream agent receives the memory content, rather than the support vectors or greedy selection trace.

C Experimental Details

This appendix provides the complete experimental details corresponding to Section 5.1. Runtime values in prompt boxes are replaced by angle-bracket placeholders. The instructions and output schemas are retained.

C.1 MS-TOD Details

Dataset and task. MS-TOD (Du et al., 2026) organizes each user history as a sequence of task-oriented dialogue sessions and provides a separate QA memory bank extracted from those sessions. We evaluate all 982 confirmation queries from 132 user histories. For a query in session (t), the dialogue preceding the confirmation turn forms the current task state (q_t). Only QA memories associated with sessions preceding (t) are eligible for retrieval:

$$\mathcal{M}_t = \bigcup_{\ell=1}^{t-1} \mathcal{QA}_\ell,$$

where $\mathcal{QA}_\ell$ denotes the QA memory units associated with historical session ℓ. Consequently, neither the current session nor any future session can enter the candidate pool.

Retrieval and candidate construction. Following the released MS-TOD retrieval setting, we encode the current intent with the query-side instruction of BGE-large-en-v1.5 and encode historical session intents as documents. We retrieve the Top-N historical sessions and expand their QA memories:

$$\mathcal{H}_t^{(N)} = \mathrm{TopN}_{\ell<t} \cos(e_q(x_t), e_d(h_\ell)), \qquad V_{q_t} = \bigcup_{\ell \in \mathcal{H}_t^{(N)}} \mathcal{QA}_\ell,$$

where x_t is the current intent, h_ℓ is the intent of historical session ℓ, and $e_q(\cdot)$ and $e_d(\cdot)$ are the query and document encoders. Following the released setting, we use $N = 10$. Each candidate is judged using its complete question–answer pair. Only its answer is inserted into the memory context, and the answer length determines its context cost.

Dataset example. In one released user history, the query session asks, "Hey, can you play Black Widow again?" A preceding `PlayMedia` session records that the song was played on the bedroom speaker. The current dialogue before confirmation constitutes the query, whereas the QA pair extracted from the earlier session is an eligible candidate memory. The example illustrates the temporal boundary: the selector may use information from prior sessions but cannot inspect the current confirmation or any future dialogue.

Prompt templates. The following boxes transcribe the frozen runtime prompts. Dynamic fields are denoted by angle-bracket placeholders.

MS-TOD Requirement Analyzer

```
[System]
You are a task-goal gap analyzer for task-oriented dialogue.
Output valid JSON only.
```

```
[User]
Given the dialogue before the confirmation turn and the task
   goal
attributes, determine the dialogue-grounding status of every
task-goal attribute.

[Current Intent]
<CURRENT_INTENT>

[Dialogue Context]
<DIALOGUE_BEFORE_CONFIRMATION>

[Original Task Goal Attributes]
<TASK_GOAL_ATTRIBUTE_NAMES>

Return JSON exactly in this schema:
{
  "slot_status": {
    "<slot_name>": {
      "status": "known|missing|uncertain",
      "value_in_dialogue": "<short exact evidence span or empty
    string>"
    }
  }
}

Rules:
- Use only the current dialogue context; do not use future
   reference
  answers or gold slot values.
- Include exactly one entry for every supplied task-goal
   attribute.
- Mark a slot known only when the dialogue gives or confirms a
   clear value.
- Mark a slot missing when no value is provided.
- Mark a slot uncertain when potentially relevant evidence is
   ambiguous.
- Output JSON only.
```

MS-TOD Requirement-Support Judge

```
[System]
You are a value-aware memory judge for task-oriented dialogue.

For each active slot, evaluate whether the candidate QA memory
   provides
a concrete value that is usable for that exact slot in the
   current task.
A score of 1 means that the answer clearly provides a concrete,
role-compatible value. A score of 0 means that no concrete
   value is
provided, the value serves a different slot role, or it
   conflicts with
the current dialogue. Topical, service, or intent similarity
   alone is
not sufficient. Do not infer an unavailable gold value.
Output valid JSON only.
```

```
[User]
[Current Intent]
<CURRENT_INTENT>

[Dialogue Context]
<DIALOGUE_BEFORE_CONFIRMATION>

[Active Slot Queries]
<ACTIVE_SLOT_QUERIES>

[Candidate QA Memory Unit]
Question q: <CANDIDATE_QUESTION>
Answer a: <CANDIDATE_ANSWER>

Return exactly one utility score from 0 to 1 for every active
   slot:
{
  "slot_utilities": {
    "<active_slot>": 0.0
  },
  "reason": "<short reason>"
}
Use the exact active-slot names and include no additional
   fields.
```

Implementation note. MemGuide trains a specialized Llama-3-8B-based scorer to estimate whether a candidate QA memory supplies information requested by the current dialogue. Since neither its checkpoint nor its complete training recipe is publicly available, we use Gemini-3.8-Flash as a strong inference-time proxy. For the MemGuide-style scalar baseline, our prompt and parser strictly follow the published pointwise interface $P(y = 1 \mid c, L_{\text{miss}}, q_i, a_i)$, where c is the dialogue context, L_{miss} is the missing-slot query list, and (q_i, a_i) is one candidate QA memory. MemQuilt preserves this information boundary and pointwise assessment, while extending the scalar output to a requirement-support vector $\mathbf{u}_i = (u_{ig})_{g \in G_q}$ for set-level coverage. This proxy is not presented as a reproduction of MemGuide's unreleased trained scorer.

MS-TOD Reader

```
[System]
You are a dialogue assistant. Generate a confirmation response
   based on
the user's utterance. Include relevant task goals identified in
    the
dialogue or related memory. If memory is unavailable, construct
    the
response accurately from the conversation. Do not introduce
   unsupported
information.

[User]
[Dialogue Context]
<DIALOGUE_BEFORE_CONFIRMATION>

[Task Goal]
<TASK_GOAL>

[Memory]
<SELECTED_MEMORIES_IN_CHRONOLOGICAL_ORDER>
```

MS-TOD LLMScore Evaluator

```
[System]
You are a strict and objective evaluator. Output valid JSON
  only.

[User]
Assess the predicted response using the conversation context,
  the user's
target goal attributes, and the reference answer.

[Dialogue Context]
<DIALOGUE_BEFORE_CONFIRMATION>

[Task Goal]
<TASK_GOAL>

[Reference Answer]
<REFERENCE_CONFIRMATION>

[Predicted Answer]
<PREDICTED_CONFIRMATION>

Evaluate requirement alignment, content accuracy, language
  quality, and
completeness relative to the reference. Return:
{
  "Rationale": "<score rationale>",
  "Score": <integer from 1 to 10>
}
```

C.2 MEM2ACT DETAILS

Dataset and task. Mem2Act evaluates memory-grounded tool invocation under a fixed-target-tool setting. Each query contains a user request together with the public target tool and its parameter schema. Following the benchmark setting, providing the target tool isolates argument grounding from tool selection. The released evaluation histories contain 3,078 original memory sources, which are decomposed into 8,541 atomic facts. We use the 381 evidence-complete queries among the 400 released queries. We exclude seven records with empty evolution chains and twelve records whose referenced evidence sources are absent from the released memory bank. Gold arguments, evidence chains, and evidence-source identifiers are reserved for evaluation and are unavailable to the Analyzer, Retriever, Judge, Selector, and Reader.

Retrieval and candidate construction. Following the released Mem2Act construction code, the retrieval query concatenates the user request with the target-tool name and description. BM25 and BGE-M3 independently rank the atomic facts, after which their complete rankings are combined by unweighted reciprocal-rank fusion:

$$s_{\mathrm{RRF}}(i \mid q) = \frac{1}{60 + \mathrm{rank}_{\mathrm{BM25}}(i)} + \frac{1}{60 + \mathrm{rank}_{\mathrm{BGE}}(i)}, \qquad V_q = \mathrm{TopN}_{i \in \mathcal{M}}\, s_{\mathrm{RRF}}(i \mid q).$$

Here, $\mathcal{M}$ is the released atomic-fact memory bank. We set $N = 50$. The fused scores are divided by the highest fused score within each query and therefore provide a query-local ranking scale rather than a calibrated probability. The BGE-M3 document representations are also used for the shared MMR and Dedup similarity geometry.

Dataset example. One released query asks, "Could you pull up the atmospheric outlook for the place I'm heading to, covering the full stretch I asked about earlier?" The fixed target tool is

`GetWeatherForecast`, for which `location` and `days` require historical grounding. Candidate facts include "The user is planning a trip to Sydney next week" and "The user wants a forecast for Sydney for the next five days." The two facts provide complementary evidence for the required arguments.

Mem2Act Requirement Analyzer

```
[System]
Classify which public tool parameters still require historical
   memory.
Use only the current query and public schema. Do not infer or
   output gold
values. Return strict JSON and no prose.

[User]
[Current Query]
<CURRENT_QUERY>

[Target Tool]
<TOOL_NAME>: <TOOL_DESCRIPTION>

[Public Parameters]
<PUBLIC_PARAMETER_SCHEMA>

For every parameter return exactly one status:
- directly_grounded: its concrete value is present in the
   current query;
- needs_memory: the value must come from history;
- use_schema_default: the public default suffices;
- not_applicable: the current request does not call for the
   parameter.

Return:
{
  "parameter_status": {"<parameter_name>": "needs_memory"},
  "reason": "<short audit>"
}
Do not output any parameter values.
```

Mem2Act Requirement-Support Judge

```
[System]
You score how strongly one historical memory supports each tool
    parameter.
Use only the query, public tool schema, and candidate memory.
   Do not invent
values. Judge the candidate independently; do not reward or
   penalize its
historical position, and do not compare it with unseen memories
   . Return
strict JSON and no prose.

[User]
[Current Query]
<CURRENT_QUERY>

[Target Tool]
<TOOL_NAME>: <TOOL_DESCRIPTION>
```

```
[Candidate Parameters]
<ACTIVE_PARAMETER_SCHEMA>

[One Candidate Memory]
<CANDIDATE_ATOMIC_FACT>

Return:
{
  "parameter_utilities": {"<parameter_name>": 0.0},
  "reason": "<short audit explanation>"
}

Each utility must be between 0 and 1. A high value requires
   concrete,
parameter-compatible, role-compatible, and scenario-compatible
   evidence.
Return exactly the supplied parameter keys.
```

Mem2Act Reader

```
[System]
You fill arguments for one already-selected target tool. Use
   the current
query, public tool schema, and selected memories. Never invent
   unsupported
personal values. Return a JSON object containing only tool
   arguments.

[User]
[Current Query]
<CURRENT_QUERY>

[Target Tool]
<TOOL_NAME>: <TOOL_DESCRIPTION>

[Parameter Schema]
<PUBLIC_PARAMETER_SCHEMA>

[Selected Memories]
The memories below are ordered from earlier to later in the
   released history.
<SELECTED_MEMORIES>

Return only the JSON arguments object for <TOOL_NAME>. Omit
   parameters that
are not requested and are not required; use a schema default
   only when
applicable.
```

C.3 LoCoMo Details

Dataset and task. LoCoMo (Maharana et al., 2024) consists of ten long conversations paired with memory questions and annotated evidence turns. We evaluate all 841 category-4 questions, which test single-hop factual recall from long conversational histories. Each published dialogue turn with an official dialogue identifier forms one memory unit. For question q, the eligible memory bank contains all published turns from its associated conversation $c(q)$:

$$\mathcal{M}_q = \{m_j : m_j \text{ is a published dialogue turn in } c(q)\}.$$

Reference answers and annotated evidence identifiers are reserved for evaluation and are not available to any runtime component.

Retrieval and candidate construction. Following the original LoCoMo retrieval protocol, we use the asymmetric DRAGON+ query and context encoders. Candidate relevance is the raw dot product between their CLS representations:

$$r_i(q) = e_Q(q)^\top e_C(m_i), \qquad V_q = \mathrm{TopN}_{m_i \in \mathcal{M}_q} r_i(q).$$

Here, $e_Q(\cdot)$ and $e_C(\cdot)$ are the DRAGON+ query and context encoders. We set $N = 50$. Selected turns are restored to publication order before being passed to the Reader. Pairwise similarity for MMR and Dedup is computed from calibrated context-encoder cosine similarities, which avoid treating the compressed cosine range caused by DRAGON+ anisotropy as uniformly high redundancy.

Dataset example. One category-4 question asks, "What did the charity race raise awareness for?" The released conversation contains turn `D2:2`: "Making a difference and raising awareness for mental health is super rewarding." The question constitutes the retrieval query, whereas this timestamped dialogue turn is one candidate memory.

LoCoMo Requirement Analyzer

```
[System]
You decompose a personal-memory question into the minimal set
   of atomic
information requirements needed to answer it.

Rules:
- Use only the question. Do not answer it or guess missing
   facts.
- Do not assume access to memories, a reference answer, or gold
    evidence.
- Each requirement must describe one independently checkable
   piece of
  information.
- Split explicitly distinct entities, relations, events, or
   comparison sides.
- Keep one requirement when the question asks for one fact.
- Preserve names, relations, constraints, and time scope.
- Use at most six requirements.

Return exactly:
{
  "requirements": [
    {"id": "r1", "description": "<atomic information
   requirement>"}
  ]
}

[User]
[Question]
<CURRENT_QUESTION>
```

LoCoMo Requirement-Support Judge

```
[System]
Given one personal-memory unit and a list of information
   requirements,
score independently how strongly the memory provides concrete
   evidence
for each requirement.

Rules:
- Evaluate only the supplied memory; do not assume unstated
   facts.
- Thematic similarity alone is not evidence.
- Use 0 when the memory does not help satisfy the requirement.
- Use 1 when it directly provides a concrete, usable answer-
   bearing fact.
- Use an intermediate value only for genuinely partial or
   indirect evidence.
- A memory may support multiple requirements.
- Return one score for every requirement ID and no additional
   keys.

Return exactly:
{
  "support": {"r1": 0.0, "r2": 0.0}
}

[User]
{
  "question": "<CURRENT_QUESTION>",
  "requirements": <ATOMIC_REQUIREMENTS>,
  "candidate_memory": {
    "id": "<DIALOGUE_TURN_ID>",
    "text": "<CANDIDATE_DIALOGUE_TURN>"
  }
}
```

LoCoMo Reader

```
[System]
You answer a question using only the supplied historical
   conversation
memories. Give a short phrase, not an explanation. Use exact
   words from
the memories whenever possible. If the memories do not contain
   the answer,
reply exactly: No information available.

[User]
[Selected Conversation Memories]
<SELECTED_TURNS_IN_CHRONOLOGICAL_ORDER>

[Question]
<CURRENT_QUESTION>

Short answer:
```

C.4 Baseline and Ablation Implementations and Hyperparameters

Alignment with the original benchmark settings. Unless otherwise stated, we follow the data organization, memory unit, retrieval representation, Reader setting, and evaluation protocol of the corresponding benchmark. Specifically, MS-TOD uses historical-session retrieval followed by QA-memory expansion and confirmation generation. Mem2Act uses its fixed-target-tool argument-grounding setting and BM25–BGE-M3 hybrid retrieval. LoCoMo uses dialogue-turn memories, asymmetric DRAGON+ retrieval, and its official answer normalization. Our controlled comparison changes only the memory-selection rule after candidate construction. Within each benchmark, all methods therefore receive the same candidate pool, individual values, context-cost definition, budget, Reader, and evaluator. MMR, Dedup, and the MemQuilt variants are additional selection baselines implemented on top of these benchmark-aligned pipelines. They are not attributed to the original benchmark papers.

Shared protocol. All methods operate on the same per-query Top-N candidate pool. In the main comparison, the budget is matched to the actual memory-context cost of Retriever Top-K, where $K = 5$:

$$B_q = \sum_{i \in S_q^{\mathrm{RetTopK}}} c_i.$$

Here, $c_i > 0$ is the context cost of memory i. Every method must return a feasible set S_q satisfying $\sum_{i \in S_q} c_i \leq B_q$. Selected memories are restored to chronological order before Reader inference. Selector order is never exposed to the Reader.

Setting	MS-TOD	Mem2Act	LoCoMo
Memory unit	QA memory	Atomic fact	Dialogue turn
Candidate pool	Top-N sessions expanded into QA memories	Top-N atomic facts	Top-N dialogue turns
N	10 sessions	50 facts	50 turns
Retriever	BGE-large-en-v1.5	BM25 + BGE-M3 RRF	DRAGON+
Reference K	5	5	5
Reader order	Chronological	Chronological	Chronological

Top-K. Candidates are ranked once by individual value p_i. The method scans this deterministic ranking and retains a candidate whenever it fits the remaining budget. It does not condition a candidate's value on the memories already selected.

MMR. Maximum Marginal Relevance (Carbonell & Goldstein, 1998) trades individual value against semantic similarity to the selected set:

$$s_i^{\mathrm{MMR}}(S) = \beta p_i - (1 - \beta) \max_{j \in S} \mathrm{sim}(i, j).$$

The first term favors individually valuable memories. The second penalizes a candidate resembling any selected memory. We set $\beta = 0.70$, giving greater weight to relevance while retaining a semantic-redundancy penalty.

Dedup. Two candidates are treated as duplicates when

$$\mathrm{sim}(i, j) \geq \delta.$$

For MS-TOD and Mem2Act, thresholded similarity defines connected duplicate groups, and the highest-value representative of each group is retained before budget packing. For LoCoMo, candidates are processed in ranking order and rejected if their calibrated similarity to any retained candidate exceeds the threshold. We use $\delta = 0.85$ for all benchmarks. This high threshold targets near-duplicate semantic content rather than general topical relatedness.

Static Utility (SU). SU compresses each requirement-support vector into a fixed scalar. For $|G_q| > 0$, we first compute the weighted mean support

$$\bar{u}_i = \frac{\sum_{g \in G_q} w_g u_{ig}}{\sum_{g \in G_q} w_g},$$

where G_q is the active-requirement set, $u_{ig} \in [0, 1]$ is the support of memory i for requirement g, and w_g is its weight. SU then assigns

$$s_i^{\mathrm{SU}} = \begin{cases} \gamma p_i + (1 - \gamma)\bar{u}_i, & |G_q| > 0, \\ p_i, & |G_q| = 0. \end{cases}$$

We set $\gamma = 0.20$, giving weights 0.20 and 0.80 to individual value and aggregated requirement support, respectively, when active requirements are present. Candidates are ranked once by s_i^{SU} and packed deterministically. SU intentionally does not inherit MemQuilt's requirement-activation gate, set-conditional gain, or gain-density stopping rule. It therefore approximates a stronger requirement-aware per-memory scorer while retaining pointwise Top-K allocation.

Strict component ablations. Each ablation changes exactly one component of MemQuilt while preserving the candidate pool, budget, cost exponent, and every other applicable signal and parameter.

Method	Sole change	Isolated component
MemQuilt	None	Full model
MemQuilt w/o IV	$p_i = 0$	Individual value
MemQuilt w/o DR	$\rho = 1$	Diminishing returns
MemQuilt w/o TS	$\tau = 0$	Threshold stopping

MemQuilt w/o IV. This variant removes the individual-value term while retaining concave requirement coverage:

$$F_q^{\text{w/o IV}}(S) = \lambda \sum_{g \in G_q} w_g \left(\sum_{i \in S} u_{ig} \right)^{\rho}.$$

It shares $\lambda, \rho, \alpha, \tau$, the candidate pool, and the budget with the corresponding MemQuilt configuration. This variant is implemented as Coverage-only in the experiment code.

MemQuilt w/o DR. This variant retains the individual-value and requirement-support signals but removes diminishing returns by setting $\rho = 1$:

$$F_q^{\text{w/o DR}}(S) = \sum_{i \in S} p_i + \lambda \sum_{g \in G_q} w_g \sum_{i \in S} u_{ig}.$$

All remaining parameters are shared with MemQuilt. This variant is implemented as Static Goal-Support in the experiment code.

MemQuilt w/o TS. This variant sets $\tau = 0$, removing the minimum gain-density cutoff while preserving the same objective and cost-aware greedy rule. Selection still terminates when no budget-feasible candidate has positive marginal gain or when no remaining candidate fits the budget.

Shared requirement activation. MemQuilt and its three strict ablations share the same upstream Requirement Analyzer and activation gate. When no active requirement is identified, i.e., $G_q = \varnothing$, selection terminates before allocation and returns $S_q = \varnothing$. This rule is distinct from gain-density stopping after allocation begins. SU instead remains a pointwise control and falls back to relevance ranking when $G_q = \varnothing$. Top-K, MMR, and Dedup are likewise independent of the activation gate.

No Memory. This baseline always returns $S_q = \varnothing$. It measures how much the selected long-term memory contributes beyond the current task state and the Reader's parametric knowledge.

MemQuilt parameters. The individual value is

$$p_i = \eta \widetilde{r}_i + (1 - \eta) a_i,$$

where $\widetilde{r}_i$ is normalized retrieval relevance and a_i is optional scalar task utility. At each greedy step, MemQuilt maximizes

$$d_q(i \mid S) = \frac{\Delta_q(i \mid S)}{c_i^{\alpha}}$$

over budget-feasible candidates and stops when the best feasible density is below τ. The mixing weight η balances retrieval relevance and scalar task utility. The parameter λ controls the strength of requirement coverage, $\rho \in (0, 1]$ controls the rate of diminishing returns, α controls sensitivity to context cost, and τ specifies the minimum acceptable gain density. We use uniform requirement weights $w_g = 1$.

Since the benchmarks differ in memory units, retrieval and support-score scales, active-requirement cardinalities, evidence structures, and context-cost distributions, one numerical configuration would not induce a comparable selection regime across all three tasks. We therefore use benchmark-specific MemQuilt parameters, fixed once per benchmark and applied to every query.

Setting	η	ρ	λ	α	τ
MS-TOD, retrieval relevance	1.00	0.55	2.0	0.25	0.40
MS-TOD, MemGuide-style value	0.20	0.55	2.0	0.10	0.50
Mem2Act	1.00	0.50	1.5	0.25	0.63
LoCoMo	1.00	0.30	8.0	0.25	0.20

The cross-benchmark MMR, Dedup, and SU parameters are globally shared. Benchmark-specific MemQuilt parameters are frozen before final evaluation. Except in model ablations, the Requirement Analyzer, Memory Judge, Reader, and MS-TOD LLMScore Evaluator use Gemini-3.8-Flash.

C.5 Evaluation Metrics

LLMScore. LLMScore measures the overall quality of an MS-TOD confirmation response. A fixed evaluator assigns each prediction an integer score $e_q \in \{1, \ldots, 10\}$, considering task-goal alignment, content accuracy, completeness, and language quality. We report

$$\text{LLMScore} = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} e_q,$$

where $\mathcal{Q}$ is the evaluation query set and e_q is the score for query q. Higher is better.

Slot Accuracy (SlotAcc). SlotAcc measures the fraction of target task attributes recovered in the generated response. Let A_q be the target-attribute set for query q, and define

$$m_{qg} = \mathbb{I}[\text{the response for } q \text{ contains a correct value or accepted alias for } g].$$

The per-query and dataset-level scores are

$$\text{SlotAcc}_q = \frac{1}{|A_q|} \sum_{g \in A_q} m_{qg}, \qquad \text{SlotAcc} = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \text{SlotAcc}_q \,.$$

Thus, $m_{qg} = 1$ when attribute g is correctly recovered and 0 otherwise. The quantity $\text{SlotAcc}_q \in [0, 1]$ is the completed fraction for one query.

Task Success (TaskSucc). TaskSucc measures complete task fulfillment. The success indicator for query q is

$$\mathrm{Succ}_q = \mathbb{I}[\mathrm{SlotAcc}_q = 1] = \mathbb{I}\left[\sum_{g \in A_q} m_{qg} = |A_q|\right],$$

and the dataset-level success rate is

$$\mathrm{TaskSucc} = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \mathrm{Succ}_q \,.$$

A query contributes one only when every target attribute is correct. If any target attribute is missing or incorrect, then $\mathrm{SlotAcc}_q < 1$ and $\mathrm{Succ}_q = 0$.

F1. Mem2Act and LoCoMo use the same precision–recall harmonic-mean form. Let U_q^p and U_q^r be the predicted and reference evaluation units, and let h_q be their overlap count. Then

$$P_q = \frac{h_q}{|U_q^p|}, \qquad R_q = \frac{h_q}{|U_q^r|}, \qquad F_{1,q} = \frac{2P_qR_q}{P_q + R_q}, \qquad F_1 = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} F_{1,q}.$$

For Mem2Act, an evaluation unit is a normalized argument key–value pair, with structured values serialized canonically before matching. For LoCoMo, the units are normalized and Porter-stemmed answer-token occurrences, and h_q is the cardinality of their multiset intersection. Hence, the two benchmarks share the F1 form but use benchmark-appropriate matching units.

BLEU-1. BLEU-1 measures unigram agreement between predicted and reference Mem2Act arguments. After serializing and tokenizing the argument objects, the clipped unigram precision is

$$p_{1,q} = \frac{\sum_w \min(C_q^p(w), C_q^r(w))}{\sum_w C_q^p(w)},$$

where $C_q^p(w)$ and $C_q^r(w)$ are the predicted and reference counts of token w. Let c_q and r_q be the predicted and reference sequence lengths. The brevity penalty and per-query score are

$$\mathrm{BP}_q = \begin{cases} 1, & c_q \geq r_q, \\ \exp(1 - r_q/c_q), & c_q < r_q, \end{cases} \qquad \mathrm{BLEU1}_q = \mathrm{BP}_q \, p_{1,q}.$$

We report the macro-average of $\mathrm{BLEU1}_q$ over queries. The brevity penalty prevents a short prediction with a small number of correct tokens from receiving an artificially high score.

Memory-Context Tokens (MemTok). MemTok measures the average context cost of the selected memories:

$$\mathrm{MemTok} = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \sum_{i \in S_q} c_i,$$

where S_q is the selected memory set and c_i is the context cost of memory i. The same c_i is used for budget feasibility and efficiency reporting.

Number of Selected Memories ($\#\mathrm{Mem}$). This metric reports the average selected-set cardinality:

$$\#\mathrm{Mem} = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} |S_q|.$$

Lower MemTok and $\#\mathrm{Mem}$ indicate a more compact memory context, provided that downstream task quality is maintained.

C.6 Controlled Comparison with Model-Directed Selection

Motivation. Model-directed selection has recently emerged as an alternative to score-based memory selection. Instead of assigning an explicit value to each candidate before allocation, an LLM or learned policy directly decides which retrieved memories enter the context. Memory-R1 (Yan et al., 2026) filters an already retrieved pool, while APEX-MEM (Banerjee et al., 2026) and CompassMem (Hu et al., 2026) interleave model reasoning with evidence retention or navigation. Here, we compare this route with the score-based selectors studied in the main text under a controlled fixed-pool setting.

Selection interface. For a query q, the Direct Set Selector (DSS) jointly reads the current task state, the retrieved candidate memories V_q, their exact context costs c_i, and the hard budget B_q:

$$\begin{aligned}\text{memory bank} \to \text{retrieval} \to V_q \\ \to \mathrm{DSS}\big(q, \{(\mathrm{id}_i, m_i, c_i)\}_{i\in V_q}, B_q\big) \to \widehat{S}_q \\ \to \mathrm{Repair}_{B_q} \to S_q \to \text{memory context.}\end{aligned}$$

In contrast, score-based selection first estimates candidate-level individual values and requirement-support vectors, then constructs a feasible set through an explicit allocation rule:

$$V_q \to \{p_i, \mathbf{u}_i, c_i\}_{i\in V_q} \to \mathrm{ExplicitSelector}(B_q) \to S_q.$$

DSS receives raw memory content rather than p_i, $\mathbf{u}_i$, retrieval scores, or active requirements. It also has no access to gold slots, gold evidence, reference responses, or downstream evaluation signals. Candidate identifiers are stable and independent of display position.

Controlled protocol. To limit the cost of controlled model replacement, we sample 300 queries uniformly without replacement from the 982-query MS-TOD evaluation set and fix this manifest before running any compared method. We use the same 300-query manifest, canonical candidate order, and fixed budget $B = 35$ as the corresponding score-based study. All methods share candidate retrieval, memory-context costs, the Gemini-3.8-Flash Reader and evaluator, and chronological Reader order. We conduct two model-controlled comparisons. In the Gemini setting, both the score-based Memory Judge and DSS use Gemini-3.8-Flash. In the GPT setting, the two corresponding selection roles use GPT-4o-mini. The Requirement Analyzer, Reader, and evaluator remain fixed to Gemini-3.8-Flash in both settings, so model identity is controlled at the selection stage rather than across the entire pipeline.

Direct Set Selector

```
[System]
You are a memory-context selector for a task-oriented dialogue
  agent.
Output valid JSON only. Do not answer the user's task.

[User]
Given the current task state and the retrieved candidate
  memories below,
select the subset that is collectively most useful for
  completing the
current task.

The total memory-context cost of the selected candidates must
  not exceed
the stated budget. Consider the candidates jointly. Select only
   memories
that add useful information, and return an empty list when no
  external
memory is needed.
```

Table 4: Controlled comparison under Gemini-3.8-Flash selection. The score-based Memory Judge and Direct Selector both use Gemini-3.8-Flash. Direct-selector task metrics are computed after deterministic feasibility repair, while raw violation is measured before repair.

Method	LLMScore	SlotAcc	TaskSucc	MemTok	Raw violation
Top-K	7.200	0.716	0.473	34.30	0%
Dedup	6.883	0.694	0.457	33.86	0%
MMR	6.940	0.717	0.460	34.29	0%
MemQuilt	7.337	**0.736**	0.477	20.73	0%
Direct Gemini Selector	**7.500**	0.713	**0.507**	**19.20**	0%

```
[Current Task State]
<DIALOGUE_BEFORE_CONFIRMATION>

[Memory-Context Budget]
<BUDGET>

[Candidate Memories]
[<CANDIDATE_ID>]
Memory-context cost: <COST>
Memory:
<CANDIDATE_QA_MEMORY>

Return exactly one JSON object:
{"selected_ids": ["<CANDIDATE_ID>", "<CANDIDATE_ID>"]}

Rules:
- List selected IDs in decreasing retention priority.
- Use each candidate ID at most once.
- Do not output IDs outside the candidate pool.
- The summed memory-context cost must not exceed <BUDGET>.
- Return {"selected_ids": []} when no candidate should be
  retained.
```

Deterministic feasibility repair. We retain both the raw model output $\widehat{S}_q$ and the repaired set S_q. Repair first removes invalid and repeated identifiers. It then scans the remaining identifiers in the model-specified priority order, retains an item only when its addition keeps cumulative cost within B_q, and otherwise skips that item before continuing the scan. Repair neither reranks nor reoptimizes the output and never inserts an omitted candidate. We report raw budget violations before repair, while all downstream task metrics use the repaired set. Thus, repair restores feasibility but does not recover utility lost through omissions, poor ordering, or over-selection.

Gemini-controlled comparison. A strong model-directed selector is competitive under the shared protocol. As shown in Table 4, Direct Gemini attains the highest LLMScore and TaskSucc while using slightly fewer memory tokens than MemQuilt. MemQuilt obtains the highest SlotAcc. Both methods satisfy the budget for every raw selection. These results demonstrate the potential of joint model-directed allocation when the selector follows the structured task and budget instructions reliably.

GPT-4o-mini-controlled comparison. Explicit allocation shows greater stability under this controlled model replacement. As shown in Table 5, MemQuilt achieves higher LLMScore, SlotAcc, and TaskSucc while using fewer memory tokens. Direct GPT-4o-mini exceeds the raw budget on 71% of queries, although the prompt explicitly states the hard constraint. Its repaired selections remain below MemQuilt on all three task metrics. This result indicates that, when allocation is per-

Table 5: Controlled comparison under GPT-4o-mini selection. The score-based Memory Judge and Direct Selector both use GPT-4o-mini. The Requirement Analyzer, Reader, and evaluator remain Gemini-3.8-Flash.

Method	LLMScore	SlotAcc	TaskSucc	MemTok	Raw violation
Top-K	6.780	0.663	0.410	34.27	0%
Dedup	6.713	0.665	0.423	33.84	0%
MMR	6.763	0.668	0.430	34.32	0%
MemQuilt	**7.150**	**0.713**	**0.447**	**21.20**	**0%**
Direct GPT-4o-mini Selector	6.780	0.621	0.417	29.74	71%

Table 6: Budget compliance of Direct GPT-4o-mini by candidate-pool size. The analysis reuses the saved Full300 outputs and requires no additional model calls.

Candidate count N	Queries	Raw violation	Repair rate
$N < 75$	63	68.3%	74.6%
$75 \leq N < 80$	103	76.7%	79.6%
$N = 80$	134	67.9%	71.6%
All	**300**	**71.0%**	**75.0%**

formed implicitly through generation, constraint compliance and set quality can depend strongly on selector capability. Deterministic repair recovers feasibility but not the utility of a better allocation.

Budget compliance across candidate-pool sizes. We group the Full300 queries by the number of candidates shown to Direct GPT-4o-mini. As shown in Table 6, raw budget violations are frequent in every group, ranging from 67.9% to 76.7%, with similarly high repair rates. The observed budget-compliance difficulty is therefore present across different candidate-pool sizes.

Discussion. The two controlled comparisons do not suggest that model-directed selection is uniformly weaker: Direct Gemini is fully budget-compliant and competitive. Instead, they indicate that a hard constraint expressed only through prompting can become model-dependent. MemQuilt retains model-based semantic assessment, but makes set construction, budget checking, and stopping explicit. We therefore focus the main comparison on score-based selectors, whose selection process is traceable, deterministically feasible, and stable under the controlled judge replacement, while reporting the complementary model-directed route here.

D Additional Effectiveness Analysis

D.1 Cross-Benchmark Results and Evidence Structures

MemQuilt remains effective across distinct memory units and evidence structures. Tables 7 and 8 report additional diagnostic metrics on Mem2Act and LoCoMo. Relative to Top-K, MemQuilt improves F1 and BLEU-1 on Mem2Act by 5.29 and 4.79 percentage points while reducing MemTok by 66.2%. On LoCoMo, it improves F1 by 12.44 percentage points with 18.2% fewer memory-context tokens. These results reinforce the task-performance and memory-context trade-off observed in the main comparison.

Stronger pointwise assessment remains insufficient for context-efficient allocation. SU uses the same Requirement Analyzer and support signals, candidate pool, budget, Reader, and evaluator as MemQuilt, but compresses requirement support into a fixed scalar and performs pointwise allocation without MemQuilt's activation gate. On Mem2Act, SU recovers more annotated evidence but uses 95.68 MemTok. MemQuilt improves F1 and Exact by 0.35 and 0.79 percentage points, respectively, while using 65.2% fewer tokens. On LoCoMo, MemQuilt improves F1 and Exact by 0.27 and 0.36 percentage points with 16.8% fewer tokens. Thus, stronger per-memory assessment is useful, but it cannot replace requirement-aware set-level optimization.

Table 7: Additional results and evidence diagnostics on Mem2Act. Gold-Int. selects annotated evidence appearing in the frozen Top-N candidate pool. Bold values indicate the best non-oracle result.

Method	Evid. Recall	Complete	F1	BLEU-1	Exact	MemTok	#Mem
Top-K	0.4428	0.4252	0.3615	0.5222	0.2651	98.51	5.00
MMR	0.4564	0.4331	0.3779	0.5341	0.2756	97.07	4.99
Dedup	0.4187	0.3858	0.3625	0.5243	0.2651	97.20	5.02
SU	**0.6314**	**0.6115**	0.4108	0.5700	0.3045	95.68	5.08
MemQuilt	0.5215	0.5013	**0.4144**	**0.5702**	**0.3123**	**33.31**	**1.88**
Gold-Int. Oracle	0.7514	0.7349	0.5954	0.6232	0.4777	16.70	0.88

Additional diagnostic metrics. Let E_q denote the annotated gold-evidence set for query q, and let S_q be the selected memory set under the benchmark-specific evidence-unit mapping. We define per-query evidence recall and its macro-average as

$$\mathrm{EvidRecall}_q = \frac{|S_q \cap E_q|}{|E_q|}, \qquad \mathrm{EvidRecall} = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \mathrm{EvidRecall}_q \,.$$

For Mem2Act, an evidence unit is an atomic fact matched by its source identifier and normalized content. For LoCoMo, it is an annotated dialogue turn. **Complete** measures the fraction of queries for which all annotated evidence is selected:

$$\mathrm{Complete} = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \mathbb{I}[E_q \subseteq S_q].$$

Exact instead evaluates the downstream Reader output:

$$\mathrm{Exact} = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \mathbb{I}[\mathrm{Norm}(\hat{y}_q) = \mathrm{Norm}(y_q)] \,,$$

where $\hat{y}_q$ and y_q are the predicted and reference outputs. For Mem2Act, exactness requires equality of the normalized argument key–value sets. For LoCoMo, it follows the benchmark evaluator and requires equality of the normalized answer-token sets. Thus, Complete measures evidence recovery, whereas Exact measures strict downstream correctness.

Different evidence structures highlight both the necessity and generality of requirement-aware set selection. Within the frozen Top-50 pools, the Gold-Int. oracles select 0.88 and 0.96 memories per query on Mem2Act and LoCoMo, respectively. The oracle set contains at most one memory for 92.1% and 96.0% of queries, showing that reachable annotated evidence is usually sparse. A zero-sized oracle set indicates that the annotated evidence is absent from the candidate pool, not necessarily that the query requires no memory. Despite this common sparsity, the two benchmarks exhibit different evidence-aggregation behavior. In Mem2Act, retrieving more memories may increase the probability of including the gold evidence, but stale or conflicting values can obscure the unique argument required for tool invocation. Performance on Mem2Act therefore depends on selecting the target memory precisely rather than merely expanding the context, highlighting the importance of MemQuilt in task-oriented settings.

LoCoMo exhibits a complementary pattern. Multiple dialogue turns, including those outside the annotated gold evidence, may provide corroborating context or resolve ambiguity in an open-ended answer. MemQuilt can therefore benefit from selecting several complementary memories, and its F1 point estimate exceeds that of the gold-intersection oracle. Together, these results show that MemQuilt can select narrowly when conflicting evidence is harmful and aggregate complementary evidence when additional context is useful.

Table 8: Additional results and evidence diagnostics on LoCoMo. Gold-Int. selects annotated evidence appearing in the frozen Top-N candidate pool. Bold values indicate the best non-oracle result.

Method	Evid. Recall	Complete	F1	Exact	MemTok	#Mem
Top-K	0.6901	0.6754	0.5074	0.2937	268.88	5.00
MMR	0.6877	0.6718	0.5088	0.2961	264.88	4.90
Dedup	0.6312	0.6147	0.4884	0.2842	262.22	4.95
SU	**0.8654**	**0.8526**	0.6292	0.3627	264.44	5.05
MemQuilt	0.8631	0.8478	**0.6319**	**0.3662**	**220.07**	**4.26**
Gold-Int. Oracle	0.9086	0.8977	0.6201	0.3710	57.16	0.96

Table 9: Results for the 337-query MS-TOD *No Requirement Identified* group.

Method	LLMScore	SlotAcc	TaskSucc	MemTok	#Mem
No Memory	9.7804	0.9633	0.8546	0.00	0.00
Top-K	9.7211	0.9641	0.8546	46.81	5.00
MMR	9.6499	0.9644	0.8546	46.02	5.45
Dedup	9.6291	0.9623	0.8516	45.80	5.39
MemQuilt	9.8012	0.9594	0.8427	0.00	0.00

The Gold-Int. oracle is an evidence-label diagnostic rather than a theoretical upper bound on Reader performance. Annotated evidence may omit dialogue turns that remain useful for contextualization, corroboration, or disambiguation.

D.2 Results by Memory Requirement

MemQuilt benefits both queries that do not need memory and those that require it. We partition MS-TOD into a 337-query *No Requirement Identified* group and a 645-query *Active Requirement Identified* group based on whether the Requirement Analyzer identifies a nonempty active requirement set before selection. As shown in Table 9, for the *No Requirement Identified* group, MemQuilt reduces MemTok from 46.81 under Top-K to zero, with only small reductions of 0.47 percentage points in SlotAcc and 1.19 percentage points in TaskSucc. Requirement-aware skipping can therefore eliminate unnecessary context while largely preserving task performance.

Requirement coverage. For a query in the *Active Requirement Identified* group, let G_q be the active requirement set and u_{ig} be the fixed Judge support assigned to memory i for requirement g. We define

$$\mathrm{GapCov}_q = \frac{1}{|G_q|} \sum_{g \in G_q} \mathbb{I}\left[\max_{i \in S_q} u_{ig} > 0\right], \qquad \mathrm{GapCov} = \frac{1}{|\mathcal{Q}_{\mathrm{req}}|} \sum_{q \in \mathcal{Q}_{\mathrm{req}}} \mathrm{GapCov}_q,$$

where $\mathcal{Q}_{\mathrm{req}} = \{q : |G_q| > 0\}$. GapCov measures the fraction of predicted active requirements receiving positive support from at least one selected memory. It is a post-selection diagnostic based on the frozen Judge signals: high GapCov does not guarantee that the supported value is correct, nor does it rule out conflicting selected memories.

As shown in Table 10, for the *Active Requirement Identified* group, MemQuilt improves SlotAcc by 17.25 percentage points and TaskSucc by 11.16 percentage points over Top-K, while reducing MemTok from 44.85 to 42.42. GapCov simultaneously increases from 41.95% to 90.33%. MemQuilt thus gains from two complementary behaviors: avoiding memory when no requirement must be filled and allocating context more effectively when external evidence is necessary.

Table 10: Results for the 645-query MS-TOD *Active Requirement Identified* group.

Method	LLMScore	SlotAcc	TaskSucc	GapCov	MemTok	#Mem
No Memory	5.4140	0.2423	0.0589	0.0000	0.00	0.00
Top-K	5.5395	0.4713	0.2403	0.4195	44.85	5.00
MMR	5.1829	0.4209	0.1442	0.3265	44.12	5.38
Dedup	5.2915	0.4489	0.1860	0.3718	43.93	5.36
MemQuilt	6.3705	0.6437	0.3519	0.9033	42.42	6.64

Small differences among zero-context runs may arise from independent Reader invocations and are not interpreted as selection effects. The remaining trade-off may also reflect imperfect requirement identification or useful context not captured by the active-requirement decision.

D.3 Context-Budget Analysis

MS-TOD provides an informative setting for evaluating allocation under varying budgets. The benchmark-specific Requirement Analyzers identify exactly one active requirement for 74.0% of Mem2Act queries and 85.7% of LoCoMo queries. The Gold-Int. statistics in Appendix D.1 further show that their reachable annotated evidence is usually singleton. MS-TOD, in contrast, commonly requires several slot values to be recovered from historical memories. A query in the *Active Requirement Identified* group contains 3.60 active requirements on average, while a memory answer unit costs 9.20 tokens on average. Using one memory unit per requirement as a descriptive reference gives a characteristic context cost of $3.60 \times 9.20 \approx 33.12$ tokens. We therefore evaluate $B \in \{20, 25, 30, 35, 40, 45\}$, covering both constrained and relatively ample budgets around this scale. This estimate is not an oracle minimum, since one memory may support multiple requirements and one requirement may require multiple pieces of evidence.

MemQuilt maintains its task-performance and memory-context trade-off throughout the evaluated budget range. As shown in Fig. 3, relevance-based MemQuilt improves LLMScore by 0.47–0.57, SlotAcc by 10.58–11.33 percentage points, and TaskSucc by 6.42–8.35 percentage points over relevance-based Top-K, while saving 6.78–15.90 tokens. The paired 95% confidence intervals exclude zero at every budget.

With MemGuide-style individual values, MemQuilt saves 7.94–18.55 tokens while maintaining comparable task performance. The confidence intervals for the context reduction exclude zero throughout, whereas several task-performance intervals overlap zero. The strongest conclusion in this setting is therefore a consistently improved performance–cost trade-off rather than uniform gains on every task metric.

Stronger per-memory assessment does not remove the need for explicit set-level optimization. Relevance-based MemQuilt achieves higher LLMScore and SlotAcc than MemGuide-style Top-K at all six budgets while obtaining comparable TaskSucc. This indicates that modeling set-conditional contributions can compensate for a weaker individual-value signal and remains important even when stronger per-memory assessment is available.

D.4 Complete Results Across Context Budgets

Tables 11 and 12 report the complete numerical results underlying Fig. 3. All memory-selection methods share the same candidate pool and nominal budget at each operating point. Since No Memory is independent of both the individual-value family and context budget, it is included once in each table as a shared reference and excluded from budget-wise paired comparisons. MemTok denotes the realized memory-context cost rather than the nominal budget. Bold values indicate the best task performance or lowest context cost within each budget.

Taken together, these complete results further support the finding in Appendix D.3 that MemQuilt maintains a favorable task-performance and memory-context trade-off across both constrained and ample budgets.

Table 11: Complete MS-TOD results across budgets with relevance-based individual values.

Budget	Method	LLMScore ↑	SlotAcc ↑	TaskSucc ↑	MemTok ↓	#Mem
–	No Memory	6.912	0.490	0.332	0.00	0.00
20	Top-K	6.595	0.587	0.377	19.21	2.94
	MMR	6.476	0.563	0.351	19.16	2.84
	Dedup	6.546	0.582	0.368	18.78	2.77
	MemQuilt	**7.067**	**0.693**	**0.441**	**12.43**	2.31
25	Top-K	6.661	0.609	0.394	24.29	3.59
	MMR	6.526	0.581	0.367	24.34	3.44
	Dedup	6.615	0.599	0.370	23.85	3.34
	MemQuilt	**7.215**	**0.721**	**0.477**	**15.63**	2.80
30	Top-K	6.827	0.625	0.412	29.28	4.12
	MMR	6.593	0.587	0.374	29.25	3.92
	Dedup	6.702	0.608	0.389	28.81	3.89
	MemQuilt	**7.305**	**0.733**	**0.496**	**18.81**	3.23
35	Top-K	6.863	0.634	0.438	34.21	4.66
	MMR	6.616	0.600	0.390	34.26	4.45
	Dedup	6.654	0.619	0.401	33.86	4.40
	MemQuilt	**7.432**	**0.745**	**0.508**	**21.94**	3.62
40	Top-K	6.962	0.640	0.447	39.29	5.21
	MMR	6.648	0.604	0.391	39.23	4.99
	Dedup	6.754	0.621	0.406	38.71	4.93
	MemQuilt	**7.536**	**0.754**	**0.518**	**25.19**	4.08
45	Top-K	7.033	0.649	0.461	44.23	5.72
	MMR	6.711	0.609	0.393	44.22	5.51
	Dedup	6.819	0.626	0.415	43.68	5.47
	MemQuilt	**7.601**	**0.761**	**0.535**	**28.34**	4.51

E Component Ablations Across Benchmarks

Following Section 5.1, we evaluate three single-change variants of MemQuilt to isolate the contributions of its design components. We report the MS-TOD results in the main text and extend the analysis to Mem2Act and LoCoMo here. All variants share the frozen candidate pool, active requirements, requirement-support signals, per-query budget, Reader, evaluator, and benchmark-specific MemQuilt parameters. **MemQuilt w/o IV** sets the individual value to zero, **MemQuilt w/o DR** sets $\rho = 1$, and **MemQuilt w/o TS** sets $\tau = 0$.

E.1 Mem2Act Component Ablation

On Mem2Act, individual value protects task accuracy, while diminishing returns and threshold stopping control context expansion. As shown in Table 13, removing IV reduces F1 and BLEU-1 by 6.43 and 6.65 percentage points, respectively. Although this variant uses 59.0% fewer memory tokens, the resulting quality loss shows that requirement coverage alone cannot reliably prioritize the most useful fact.

Removing DR changes F1 and BLEU-1 by $+0.40$ and -0.79 percentage points, while increasing MemTok by 62.1%. The full model therefore maintains comparable task performance with substantially less context. Removing TS obtains higher task-metric point estimates, but increases MemTok from 33.31 to 93.10. This amounts to 179.5% more context for improvements below two percentage points on both task metrics. Threshold stopping therefore controls the performance–context trade-off rather than maximizing task performance regardless of cost.

Table 12: Complete MS-TOD results across budgets with MemGuide-style individual values.

Budget	Method	LLMScore ↑	SlotAcc ↑	TaskSucc ↑	MemTok ↓	#Mem
–	No Memory	6.912	0.490	0.332	0.00	0.00
20	Top-K	6.941	0.680	**0.438**	19.29	2.89
	MMR	6.833	0.683	0.434	19.23	2.89
	Dedup	6.810	0.674	0.428	18.84	2.74
	MemQuilt	**7.087**	**0.694**	0.436	**11.35**	1.69
25	Top-K	7.129	0.704	0.468	24.31	3.54
	MMR	7.045	0.711	**0.473**	24.31	3.48
	Dedup	6.968	0.693	0.441	23.92	3.35
	MemQuilt	**7.244**	**0.720**	0.470	**14.23**	2.03
30	Top-K	7.276	0.721	0.485	29.25	4.12
	MMR	7.091	0.722	0.471	29.24	4.01
	Dedup	7.086	0.708	0.453	28.86	3.88
	MemQuilt	**7.367**	**0.737**	**0.496**	**17.21**	2.40
35	Top-K	7.388	0.733	**0.503**	34.29	4.65
	MMR	7.216	0.734	0.486	34.27	4.58
	Dedup	7.118	0.721	0.482	33.86	4.41
	MemQuilt	**7.475**	**0.747**	0.500	**20.17**	2.72
40	Top-K	7.442	0.743	0.502	39.28	5.24
	MMR	7.296	0.745	0.498	39.25	5.14
	Dedup	7.188	0.724	0.482	38.72	4.93
	MemQuilt	**7.545**	**0.754**	**0.512**	**22.94**	3.06
45	Top-K	7.594	0.756	**0.533**	44.30	5.78
	MMR	7.408	0.754	0.516	44.32	5.69
	Dedup	7.268	0.728	0.487	43.77	5.54
	MemQuilt	**7.631**	**0.758**	0.524	**25.76**	3.37

Table 13: Component ablations of MemQuilt on Mem2Act. All variants are evaluated on the same 381 Mem2Act queries. Bold values indicate the best task metric or lowest context cost.

Method	F1 ↑	BLEU-1 ↑	MemTok ↓	#Mem ↓
MemQuilt w/o IV	0.350	0.504	**13.66**	**0.81**
MemQuilt w/o DR	0.418	0.562	53.99	3.04
MemQuilt w/o TS	**0.434**	**0.587**	93.10	5.31
MemQuilt	0.414	0.570	33.31	1.88

E.2 LoCoMo Component Ablation

On LoCoMo, all three components contribute to a stronger performance–context balance. As shown in Table 14, removing IV reduces F1 by 2.46 percentage points. Its lower context cost does not compensate for the loss in answer quality, showing that individual value remains important for prioritizing useful dialogue turns among requirement-relevant candidates.

Removing DR leaves MemTok nearly unchanged at 222.55, but reduces F1 by 1.02 percentage points. With nearly the same context, concave accumulation better favors memories that provide complementary support instead of repeatedly satisfying an already-covered requirement. Removing TS increases MemTok by 19.6% and #Mem by 26.7%, while slightly reducing F1. Threshold stopping thus removes low-density additions that consume context without improving the final answer.

Overall, the three components play distinct and complementary roles across evidence structures. Together with the MS-TOD ablation in the main text, these results show that IV priori-

Table 14: Component ablations of MemQuilt on LoCoMo. All variants are evaluated on the same 841 category-4 questions. Bold values indicate the best task metric or lowest context cost.

Method	F1 ↑	MemTok ↓	#Mem ↓
MemQuilt w/o IV	0.607	**136.95**	**2.41**
MemQuilt w/o DR	0.622	222.55	4.29
MemQuilt w/o TS	0.631	263.23	5.39
MemQuilt	**0.632**	220.07	4.26

tizes individually useful memories, DR models their set-conditional contributions, and TS prevents low-value additions from exhausting the available context. Their effects need not improve every task metric monotonically because MemQuilt targets the balance between task performance and memory-context cost. Across all three benchmarks, removing IV trades task quality for lower cost, removing DR weakens quality or context efficiency, and removing TS consistently expands the selected context for small or non-uniform performance changes.

F Theory-Guided Analysis Details

F.1 Data and Shared Protocol

Fig. 5 is constructed from the MS-TOD Full982 manifest under a fixed context budget of $B = 35$. It uses the same frozen candidate pools, Requirement Analyzer outputs, requirement-support signals, memory-context costs, and MemGuide-style individual values as the corresponding main experiment. Since selection terminates immediately when no active requirement is identified, both panels analyze the 645 queries with $G_q \neq \varnothing$. All statistics are computed offline from saved selector snapshots and selection outputs, without additional model calls.

F.2 Conditional-Gain Trajectories

For each query, we replay MemQuilt from $S = \varnothing$ using the frozen candidate assessments and the same selector parameters as the main experiment. At each greedy step, the conditional gain density of candidate i is

$$d_q(i \mid S) = \frac{\Delta_q(i \mid S)}{c_i^{\alpha}},$$

and the plotted value is its threshold-normalized form $d_q(i \mid S)/\tau$. The replay exactly reproduces the official selected set before threshold termination.

To expose the declining utility of later candidates, the diagnostic replay continues after the best feasible density first falls below τ, while preserving the original feasibility checks, support updates, and deterministic tie-breaking. These post-threshold additions are diagnostic only and are not included in the memory context. Consequently, values below 1 represent candidates rejected by the deployed selector. At each step, we report the median normalized density and its interquartile range. A step is shown only when it is reached by at least 5% of active-requirement queries.

F.3 Budget Utilization by Task Demand

We group queries according to the number of active requirements identified by the frozen Requirement Analyzer: 1, 2, 3, 4, and ≥ 5. For each method and group, budget utilization is computed as

$$\mathrm{Util}(S_q) = \frac{c(S_q)}{B}.$$

The right panel reports the mean utilization within each group. To account for multiple queries belonging to the same user history, uncertainty is estimated by a persona-cluster bootstrap. We resample the 132 personas with replacement while retaining all queries associated with each sampled persona, repeat this procedure 10,000 times, and use the 2.5% and 97.5% percentiles as the 95% confidence interval. These intervals characterize aggregation uncertainty and are not used for null-hypothesis significance claims.

G CASE STUDIES

The cases below preserve every selected memory and the complete Reader output. Within each method, `M1`, `M2`, and so forth enumerate memories in the chronological order presented to the Reader.

G.1 MS-TOD: COMPOSING DISTRIBUTED SLOT EVIDENCE

Case G.1: Multi-Slot Car Reservation

Case ID	`query_index=202` (Persona 20, Session 22)
Service / intent	RentalCars / ReserveCar
Dialogue context	User: "No, that's all I need." Assistant: [response to generate]
Gold response	"Your reservation for a Standard car has been successfully placed. The pickup will be at Canterbury Village on March 1st at 6:30 pm and dropoff on March 6th. Please confirm if all details are correct."
Gold slots	`pickup_location=Canterbury Village`; `pickup_date=March 1st`; `pickup_time=6:30 pm`; `dropoff_date=March 6th`; `car_type=Standard`.

Top-K SlotAcc=0.20 TaskSucc=0 MemTok=39

Selected memories

`M1`: Q: What is the name of the user? A: The user's name is John Doe.

`M2`: Q: What is the user's phone number? A: The user's phone number is 123-456-7890.

`M3`: Q: Where did the user make a reservation? A: The user made a reservation at the Doubletree By Hilton Hotel Fresno Convention Center.

`M4`: Q: When is the user checking in? A: The user is checking in on March 6th.

`M5`: Q: How many rooms did the user reserve? A: The user reserved three rooms.

Reader output

Your booking has been successfully confirmed, Mr. John Doe!
Here are your reservation details: • **Guest Name:** John Doe
• **Contact Number:** 123-456-7890
• **Location:** Doubletree By Hilton Hotel Fresno Convention Center
• **Check-in Date:** March 6th
• **Rooms Reserved:** 3 rooms
Additionally, your car rental arrangements, including the **pickup location**, **pickup date**, **pickup time**, **dropoff date**, and **car type**, have been confirmed to coordinate with your stay.
Thank you for booking with us, and please have a safe and pleasant trip!

MMR SlotAcc=0.40 TaskSucc=0 MemTok=39

Selected memories

`M1`: Q: What is the name of the user? A: The user's name is John Doe.

`M2`: Q: When is the drop-off date for John Smith's car reservation? A: The drop-off date for John Smith's car reservation is March 11th.

`M3`: Q: Where did the user make a reservation? A: The user made a reservation at the Doubletree By Hilton Hotel Fresno Convention Center.

M4 : Q: When is the user checking in? A: The user is checking in on March 6th.

Reader output
Your car reservation has been successfully confirmed! Here are the details of your booking: • **Name:** John Doe
• **Pickup Location:** Fresno (Doubletree By Hilton Hotel Fresno Convention Center)
• **Pickup Date:** March 6th
• **Pickup Time:** As requested
• **Drop-off Date:** March 11th
• **Car Type:** Confirmed as requested
Everything is all set for your trip. Thank you, and please let me know if you need anything else!

Dedup SlotAcc=0.20 TaskSucc=0 MemTok=39

Selected memories
M1 : Q: What is the name of the user? A: The user's name is John Doe.
M2 : Q: What is the user's phone number? A: The user's phone number is 123-456-7890.
M3 : Q: Where did the user make a reservation? A: The user made a reservation at the Doubletree By Hilton Hotel Fresno Convention Center.
M4 : Q: When is the user checking in? A: The user is checking in on March 6th.
M5 : Q: How many rooms did the user reserve? A: The user reserved three rooms.

Reader output
Your booking has been successfully confirmed! Here are the details of your reservation: • **Name:** John Doe
• **Phone Number:** 123-456-7890
• **Hotel:** Doubletree By Hilton Hotel Fresno Convention Center
• **Check-in Date:** March 6th
• **Rooms Reserved:** 3 rooms
Additionally, your car rental booking details (including your **pickup location**, **pickup date**, **pickup time**, **dropoff date**, and **car type**) have been processed and confirmed.
Thank you for choosing our service, and please let me know if you need anything else!

MemQuilt SlotAcc=1.00 TaskSucc=1 MemTok=35

Selected memories
M1 : Q: When does the user need the car? A: The user needs the car on March 1st at 6:30 PM.
M2 : Q: How long is the user's car rental period? A: The user's car rental period is from March 1st to March 6th.
M3 : Q: What type of car did the user request? A: The user requested a standard car.
M4 : Q: What is the pickup location for the car reservation? A: The pickup location is Canterbury Village.

Reader output
Your booking has been confirmed! Here are the details of your reservation: • **Car Type:** Standard car
• **Pickup Location:** Canterbury Village
• **Pickup Date & Time:** March 1st at 6:30 PM
• **Drop-off Date:** March 6th
Thank you for booking with us. Please let me know if you need anything else!

MemQuilt composes complementary memories to satisfy multiple requirements. Top-K and Dedup select memories from an unrelated hotel reservation, while MMR increases diversity but retains an incompatible rental date. MemQuilt instead selects four QA memories that jointly supply all five required slot values with less memory context.

G.2 MEM2ACT: ISOLATING AN ARGUMENT-BEARING FACT

Case G.2: Mutual-Fund Symbol Recovery

Case ID	`qa_284`
Query	"Can you pull up the financial ratios for that mutual fund we were just talking about?"
Target tool	`Get Company Financial Ratios`
Gold output	{`"symbol": "VFIAX"`}

Top-K F1=0.00 BLEU-1=0.50 MemTok=114

Selected memories

`M1`: User is requesting to pull the latest financial statement of the company and save it as `technova_financials.pdf` in the reports folder.

`M2`: User requests the latest earnings report for Apple (`AAPL`).

`M3`: User requests the latest financial ratios for Apple (`AAPL`).

`M4`: User is requesting detailed financial information on Microsoft, particularly regarding its performance as a publicly traded company.

`M5`: User is requesting the latest financial information about Apple Inc.

Reader output {`"symbol": "AAPL"`}

MMR F1=0.00 BLEU-1=0.50 MemTok=114

Selected memories

`M1`: User is requesting to pull the latest financial statement of the company and save it as `technova_financials.pdf` in the reports folder.

`M2`: User is requesting to retrieve the top business news.

`M3`: User requests the latest financial ratios for Apple (`AAPL`).

`M4`: User is requesting the latest scheme categories for equity mutual funds.

`M5`: User is requesting detailed financial information on Microsoft, particularly regarding its performance as a publicly traded company.

Reader output {`"symbol": "MSFT"`}

Dedup F1=0.00 BLEU-1=0.50 MemTok=113

Selected memories

`M1`: User is requesting to pull the latest financial statement of the company and save it as `technova_financials.pdf` in the reports folder.

`M2`: User is requesting to retrieve the top business news.

`M3`: User requests the latest financial ratios for Apple (`AAPL`).

`M4`: User is requesting detailed financial information on Microsoft, particularly regarding its performance as a publicly traded company.

`M5`: User is requesting the latest financial information about Apple Inc.

Reader output {`"symbol": "AAPL"`}

MemQuilt F1=1.00 BLEU-1=1.00 MemTok=18

Selected memories

`M1`: User is requesting ratings for the mutual fund with the symbol `VFIAX`.

Reader output `{"symbol": "VFIAX"}`

Precise memory selection protects a unique tool argument from topical distractors. All three baselines select finance-related memories containing conflicting company symbols. MemQuilt selects only the fact that explicitly carries `VFIAX`, recovering the correct tool argument with 18 memory tokens.

G.3 LoCoMo: Aggregating Complementary Conversation Turns

Case G.3: Resolving an Implicit Activity

Case ID	`conv-49:q109`
Query	"What fun activity did Evan mention doing in July 2023?"
Gold answer	`skiing`
Annotated gold memory	`D8:26`

Top-K F1=0.00 MemTok=226

Selected memories

`M1` `(D4:16)`: Evan: "That's awesome, Sam! It's such a rewarding and tough activity - keep going and have fun!"

`M2` `(D6:1)`: Evan: "Hey Sam, long time no talk! Hope you're doing great. I just got back from a rad vacay with my new SO in Canada. Tried some awesome activities too - think hiking, biking... all that cool stuff. We loved exploring the outdoors together, it was so awesome!" [Image caption: "a photo of a tent pitched up in a grassy field"]

`M3` `(D8:26)`: Evan: "Yep, last month I drove somewhere fun. The views were amazing!" [Image caption: "a photo of a person on skis on a snowy trail"]

`M4` `(D8:30)`: Evan: "Skiing, snowboarding, and ice skating are all fun winter activities I enjoy."

`M5` `(D16:24)`: Evan: "Awesome, let's do it! Let's plan it for next month, I'm already excited about exploring it together!"

Reader output "drove somewhere fun"

MMR F1=0.00 MemTok=226

Selected memories

`M1` `(D4:16)`: Evan: "That's awesome, Sam! It's such a rewarding and tough activity - keep going and have fun!"

`M2` `(D6:1)`: Evan: "Hey Sam, long time no talk! Hope you're doing great. I just got back from a rad vacay with my new SO in Canada. Tried some awesome activities too - think hiking, biking... all that cool stuff. We loved exploring the outdoors together, it was so awesome!" [Image caption: "a photo of a tent pitched up in a grassy field"]

`M3` `(D8:26)`: Evan: "Yep, last month I drove somewhere fun. The views were amazing!" [Image caption: "a photo of a person on skis on a snowy trail"]

`M4` `(D8:30)`: Evan: "Skiing, snowboarding, and ice skating are all fun winter activities I enjoy."

`M5` `(D16:24)`: Evan: "Awesome, let's do it! Let's plan it for next month, I'm already excited about exploring it together!"

Reader output "drove somewhere fun"

Dedup F1=0.00 MemTok=205

Selected memories

M1 (D6:1): Evan: "Hey Sam, long time no talk! Hope you're doing great. I just got back from a rad vacay with my new SO in Canada. Tried some awesome activities too - think hiking, biking... all that cool stuff. We loved exploring the outdoors together, it was so awesome!" [Image caption: "a photo of a tent pitched up in a grassy field"]

M2 (D8:30): Evan: "Skiing, snowboarding, and ice skating are all fun winter activities I enjoy."

M3 (D13:9): Evan: "No worries, Sam! It's a fun way to get in some exercise and enjoy nature. Let me know when you're ready to give it a try and I can hook you up with a good spot."

M4 (D16:24): Evan: "Awesome, let's do it! Let's plan it for next month, I'm already excited about exploring it together!"

Reader output "No information available."

Gold-Intersection Oracle F1=0.00 MemTok=40

Selected memories

M1 (D8:26): Evan: "Yep, last month I drove somewhere fun. The views were amazing!" [Image caption: "a photo of a person on skis on a snowy trail"]

Reader output "drove somewhere fun"

MemQuilt F1=1.00 MemTok=211

Selected memories

M1 (D4:16): Evan: "That's awesome, Sam! It's such a rewarding and tough activity - keep going and have fun!"

M2 (D8:26): Evan: "Yep, last month I drove somewhere fun. The views were amazing!" [Image caption: "a photo of a person on skis on a snowy trail"]

M3 (D8:28): Evan: "Yeah, it was great for skiing! The snow was amazing and I had a lot of fun. Can't wait to go back next year! Did you try any winter sports, Sam?" [Image caption: "a photo of a group of people skiing down a snow covered slope"]

M4 (D8:30): Evan: "Skiing, snowboarding, and ice skating are all fun winter activities I enjoy."

M5 (D25:10): Evan: "I had a great time kayaking and watching the sunset last summer - it was truly unforgettable. Being out on the water is so peaceful."

Reader output "skiing"

Complementary conversational evidence resolves an answer that is implicit in the annotated turn. The dialogue text of the annotated memory identifies only a vague event. Top-K and MMR include generic winter-activity context but omit the turn explicitly connecting that event to skiing. MemQuilt selects this bridging turn and recovers the gold answer.