Prompt Engineering / AI Agents

Define when and how an agent should query memory (short-term, long-term, episodic) for different task types.
Difficulty: Advanced
Model: GPT-4 / Claude / Gemini
Use Case: Agent Memory Systems, RAG Optimization, Context Management
Updated: May 2026
Why This Prompt Exists
Agents either retrieve too much memory (context overload, slow, expensive) or too little (missing critical information). Both break the agent.

You get:

- agents that retrieve every conversation turn (context explosion)
- agents that retrieve nothing (hallucination, repeating mistakes)
- no strategy for when to use short-term vs. long-term memory
- no retrieval cues (agent doesn’t know what to search for)
- memory pollution (irrelevant information from past tasks)

But memory retrieval can be strategic:

- short-term: current conversation only (fast, limited)
- long-term: past interactions (slow, broad, needs search)
- episodic: specific past events (targeted, high relevance)
- semantic: facts and knowledge (declarative, stable)
- procedural: how to do things (skills, workflows)

Without strategy, agents drown in memory or starve for context.

This prompt designs optimal memory retrieval strategies for agents.

The Prompt

```
Assume the role of an agent memory architect who optimizes retrieval
strategies.

Your task is to specify when and how an agent should access different
memory types.

Generate:

1. TASK TAXONOMY
   - Types of tasks the agent performs
   - Information needed per task type
   - How far back relevant information might be

2. MEMORY TYPES AVAILABLE
   - Short-term (current session)
   - Long-term vector store (semantic search)
   - Episodic (timestamped events)
   - Procedural (workflows, how-to)
   - Semantic (facts, knowledge base)

3. RETRIEVAL STRATEGY PER TASK TYPE

| Task Type | Memory Type | Retrieval Trigger | Max Results |
Freshness Requirement |
|-----------|-------------|-------------------|-------------|---------
```

```
-------------|
| [type] | [short/long/episodic] | [always / on-demand / confidence <
X] | [N] | [recent/any] |

4. RETRIEVAL QUERY OPTIMIZATION
   - How to convert task context into search queries
   - Query templates per task type
   - Example: "For customer support, search: 'customer_id: {id} issue:
{summary}'"

5. MEMORY UPDATE STRATEGY
   - What gets saved to memory (after each task? only successes?)
   - Retention policy (how long before deletion)
   - Summarization before storage (compress long conversations)

6. FALLBACK BEHAVIOR
   - What if retrieval returns nothing? (proceed, ask user, use
default)
   - What if retrieval returns conflicting information? (resolve,
flag, use most recent)

7. READY-TO-USE MEMORY PROMPT
   - A prompt snippet that implements this retrieval strategy

INPUTS:

Agent domain:
[CUSTOMER SUPPORT / RESEARCH / CODING / PERSONAL ASSISTANT / OTHER]
```

```
Typical conversation length:
[SHORT (1-5 turns) / MEDIUM (5-20) / LONG (20+)]

Memory store available:
[VECTOR / SQL / KEY-VALUE / NONE]

Latency budget:
[<500ms / <2s / <10s / unlimited]

RULES:
- Short-term memory is fast but limited (use for recent context only)
- Long-term retrieval requires good queries (invest in query generation)
- Don't retrieve for every turn — use confidence thresholds
- Summarize before storing long conversations (or you'll retrieve noise)
- Test retrieval recall (are you finding what you need?) and precision (are you retrieving noise?)
```

How To Use It

- Start with the task taxonomy — different task types need different retrieval strategies.
- Use "on-demand" retrieval (only when needed) to save latency and tokens.
- Set a max results limit (5-10 is usually enough; more just adds noise).
- Test retrieval recall — if you can't find relevant memories, improve your query generation.
- Implement memory summarization for long conversations before storage.

Example Input

Agent domain:

"Customer support for an e-commerce site"

Typical conversation length:
"MEDIUM (5-20 turns)"

Memory store available:
"VECTOR (past conversations, product knowledge base)"

Latency budget:
"<2s"

Why It Works
Most agents use a single retrieval strategy — usually "search everything every time" — which is slow, expensive, and noisy.

This framework improves outcomes by forcing:

- task taxonomy (different tasks need different memory)
- per-task retrieval strategy (not one-size-fits-all)
- retrieval triggers (don't search when you don't need to)
- query optimization (garbage in, garbage out for search)
- fallback behavior (what to do when retrieval fails)

Great memory retrieval doesn't store everything — it retrieves the right thing at the right time.

Build Better AI Systems

Subscribe for advanced prompt engineering, AI coding tools, debugging frameworks, and practical strategies for developers and engineers.

Carefully engineered prompts for people doing real work.

Share this:

- Share on Facebook (Opens in new window) Facebook
- Share on X (Opens in new window) X

See also Tool Selection Optimizer