7 Types of Agent Memory — the Full Breakdown

The carousel gave you the map. This is the field guide: what each memory type is, when you actually need it, how to build it, and which one to build first.

The problem

Everyone says "my agent needs memory," then bolts on a vector database and calls it done. But "memory" is seven different things, each solving a different failure: forgetting mid-conversation, forgetting the user, repeating old mistakes, re-deriving known procedures, missing outside knowledge, trusting stale training data, and dropping planned work. Pick the wrong type and you've built infrastructure for a problem you don't have.

The 7 types

1 · Working memory (in-context)

What: the active context window — everything the model can see right now: current messages, system prompt, tool outputs, reasoning steps. **The failure it fixes:** the agent losing the thread inside one conversation or task. **Build it:** you already have it — the work is managing it. Use a checkpointer keyed by thread ID; decide your truncation strategy (summarize old turns, drop tool noise) before you hit the limit, not after. **Rule of thumb:** if the information matters for *this* conversation only, it belongs here and nowhere else.

2 · Semantic memory

What: a persistent store of facts, preferences, and domain knowledge — "this user prefers metric units," remembered next session. **The failure it fixes:** the agent treating a returning user like a stranger. **Build it:** a profile store (structured rows for known fields) plus a vector DB for free-form facts. Write on signal ("remember that…", corrections, repeated choices), retrieve at session start. **Rule of thumb:** facts about *who* — users, accounts, domains — that must outlive the session.

3 · Episodic memory

What: a record of past events and task outcomes — full conversations, successes, failures, timestamps — in an event-log DB. **The failure it fixes:** repeating mistakes it already made. **Build it:** log every task run as a structured event (input, actions, outcome, result). On new tasks, retrieve similar past episodes and inject the lessons. The published proof this works: Reflexion (Shinn et al., NeurIPS 2023) — the agent writes a self-reflection after each failed attempt, stores it, reuses it, and hit 91% on a coding benchmark where GPT-4 alone scored 80%. **Rule of thumb:** if you'd want an audit trail or a "we tried this before" check, it's episodic.

4 · Procedural memory

What: knowledge of *how* tasks get done — skills, workflows, tool-usage patterns, behavioral rules — stored as prompt templates, tool schemas, scripts, skill files. **The failure it fixes:** re-deriving the same steps every single run. **Build it:** every time the agent (or you) solves something twice, freeze the solution into a reusable skill or template with a name and a version. Voyager, the Minecraft agent, is the canonical example — it builds a library of executable skills, then reuses and composes them for new tasks. **Rule of thumb:** the second time you explain a procedure, you should be saving it instead.

5 · Retrieval memory (RAG)

What: outside knowledge in a vector DB, fetched at inference: embed the query → similarity search → inject the top chunks into context. **The failure it fixes:** the context window being too small for your docs, wikis, and knowledge bases. **Build it:** chunk your documents, embed them, index with metadata (source, date), retrieve top-K per query. Your chunking strategy and freshness policy matter more than which vector DB you pick. **Rule of thumb:** knowledge that's too big to carry and changes independently of the agent → retrieval.

6 · Parametric memory

What: what's baked into the model's weights from training — language, reasoning patterns, world knowledge. No retrieval, instant, free. **The failure it fixes:** none directly — it's the default you're always using. The real skill is knowing its edges: capped at the training cutoff, hard to update, hard to audit. **Build it:** you don't. You *decide against* it: anything that must be current, verifiable, or client-specific goes in types 2–5 instead. The model knows what a REST API is; it doesn't know the library that shipped last month. **Rule of thumb:** if being wrong about it would cost you, don't leave it in the weights.

7 · Prospective memory

What: the agent's record of future intentions — pending tasks, goal stacks, scheduled follow-ups, each with a trigger (time, event, or agent). **The failure it fixes:** long-horizon agents dropping the plan when interrupted. Finishes step 2 of 5, gets cut off, and on restart continues at step 3 with the right inputs — instead of starting over. **Build it:** a task queue + a goal stack the agent writes to before stopping and reads on wake. Pairs with episodic memory (what happened) to give the full picture (what happened

+ what's still owed). **Rule of thumb:** if the agent runs longer than one sitting, it needs this before it needs anything fancy.

Which one do you build first?

Match the symptom, not the hype:

- Loses the thread mid-conversation → **working** (fix your context management first — it's free).
- Forgets the user between sessions → **semantic**.
- Repeats old mistakes → **episodic**.
- Re-solves solved problems → **procedural**.
- Doesn't know your docs → **retrieval**.
- Confidently wrong about recent things → stop relying on **parametric**; add 2 or 5.
- Drops multi-day work when interrupted → **prospective**.

Most real agents need two or three of these, not seven. Start with the one whose failure you're actually seeing this week.

Where this fits

Memory is one of the load-bearing walls of any production agent — the others being the loop that drives it, the checks that catch bad work, and the connectors that let it act. Every system I ship for clients is some arrangement of those four. Get memory right and the other three get dramatically easier.

Your next step

Pick the ONE symptom from the list above that your agent showed this week, and build only that memory type. A week from now you'll have a

visibly smarter agent — instead of a half-configured vector database and the same old failures.

Follow **@saad.b.ai** — every breakdown I share works like this: real, tested, usable today.