SOULWIRE STUDIO

AI AUTOMATION FOR OPERATORS

RAG vs. the Simple Chatbot

Both are chatbots. The question is where the answers come from.

A decision playbook for choosing the right knowledge source behind an AI assistant — before a single line of code gets written.

Three data buckets. Three builds. No vector database where a system prompt or a query will do.

SECTION I

The Core Idea

This is not a choice between "a chatbot" and "RAG." Both are chatbots. The only real difference is where the assistant gets its answers, and that is driven by what the underlying data looks like, not by which approach sounds more advanced.

Pick the knowledge source that matches the data. Anything more is cost and complexity carried for no reason.

THE ONE-LINE TEST

```
If the knowledge fits in a prompt  →  prompt it.
If it is records you filter  →  query it.
If it is a pile of documents you search  →  that is RAG.
```

> **WHY THIS MATTERS**
>
> RAG has real moving parts: an embeddings model, a vector table, a reranker, and a live database connection. That machinery earns its keep in exactly one of the three buckets on the next page. Reach for it because the data demands it, not by default.

SECTION II

The Three Buckets

01

Small, fixed facts

A finite, known set of answers — an FAQ, a service list, a set of policies that rarely change. Put the facts directly in the system prompt. No database, no embeddings, nothing to maintain. The answers should come back exact every time; retrieval only adds fuzziness here.

02

Structured records you filter

Rows with fields to sort and filter — listings, inventory, suppliers, pricing, member directories. Give the assistant a database lookup tool that queries the table directly. "Show records in region X under this price" is an exact filter; semantic search returns the most similar-sounding rows, not a clean filtered list.

03

Large, unstructured documents

Prose too big to paste into a prompt — contracts, manuals, transcripts, policy PDFs, reports. Build Retrieval-Augmented Generation over a vector store: chunk it, embed it, retrieve only the few relevant chunks per question. The knowledge is too large for context and too unstructured to query with fields.

Identify the bucket first. The build follows.

SECTION III

Quick Reference

DATA LOOKS LIKE	RIGHT BUILD	WRONG MOVE
Known, finite facts	Facts in the system prompt	RAG — overkill, adds fuzziness
Structured records to filter	Database lookup tool	RAG — imprecise on fields
Big pile of documents	RAG vector store	Prompt stuffing — won't fit

THE BUY SIGNAL

On a discovery call, listen for this sentence: *"The answer is in there somewhere, but nobody can find it."* That points at bucket three — a shelf of policy documents, years of contracts, a long program manual. If the client instead describes a set list of answers or a spreadsheet of records, they are in bucket one or two. Do not sell them RAG.

SECTION IV

Where RAG Pays Off

RAG carries real moving parts: an embeddings model, a vector table sized to match, a reranker, a live database connection, and the discipline to keep embedding dimensions consistent across ingestion and retrieval. A mismatch quietly breaks results. That cost is worth it for bucket three and only bucket three — for buckets one and two it is slower, costlier, and less precise than the simpler build.

01 **Professional-service firms**

Years of contracts and playbooks. "Have we done a deal like this" questions.

02 **Government & economic development**

Grant guidelines, zoning rules, RFP requirements.

03 **Healthcare operations**

Protocols, onboarding manuals, compliance documents — the paperwork, not clinical decisions.

04 **Nonprofits**

Program manuals, donor reports, board materials. Keeps institutional memory answerable through staff turnover.

05 **Any business with a real knowledge base**

Product manuals, warranty terms, an internal wiki.

When a simple build wins: a compliance FAQ with a fixed set of answers goes in the prompt. A catalog people filter and sort gets a database lookup. Anything that fits comfortably in one prompt does not need infrastructure.