WINDOWS, MACOS OR LINUX · VERSION 1.0 · 7 NOVEMBER 2026

BUILD AN AI THAT SEARCHES EVERYTHING YOU'VE WRITTEN

A goal-driven build prompt for your coding agent, plus the mistakes and tests that turn a RAG demo into a search tool you can trust with your own notes.



WHAT IT TEACHES — Embeddings · Semantic search · Retrieval (RAG)

01 One build prompt you paste once
02 The mistakes that break this build
03 Acceptance tests to prove it works
04 A repair prompt for when it doesn't

BEFORE YOU START

START HERE

01

Create a new folder called My Knowledge Search. Inside it, create a folder called documents and copy about 20 of your notes, PDFs or Word files into it. Use copies, not originals.

02

Open a coding agent that can create files and run commands on your computer (for example Codex or Claude Code). Copy both parts of the build prompt into it as one message and send it. The agent works through every stage and stops only when it needs you.

03

Before the agent starts, write five questions you already know the answers to, with the file each answer is in. The agent will ask for them. They are your test set.

04

Run the acceptance tests before you rely on an answer. Keep the mistakes page and repair prompt for later.

WHAT YOU NEED

A computer with at least 8 GB of RAM (16 GB is more comfortable), a few GB of free disk space, an internet connection for setup, a coding agent with access to that computer, and Ollama, which runs AI models locally.

WHAT YOU ARE BUILDING

Type a question in plain English. Get the most relevant passages from your own files, each with the file name and page. Ask for an answer and get one written only from those passages, with every claim linked to its source. Your files and the models stay on your computer.

KEEP THE SCOPE SMALL

One folder. Text you can extract (no scanned images yet). Search first, answers second. Make search good before you add the chat model, because the answer can never be better than what search finds.

THE BUILD PROMPT · PART 1

PASTE BOTH PARTS AS ONE MESSAGE

Copy part 1 and part 2 into your coding agent as one message.

BUILD PROMPT · PART 1 COPY INTO YOUR AGENT

GOAL

Build a private search engine for the documents folder in this project. I ask a question in my own words and get the passages that answer it, even when my words differ from the words in the file. Then I can ask for a short answer written only from those passages, with a source for every claim. Everything runs locally with Ollama.

DONE MEANS

For my five test questions, the correct source passage appears in the top five results. Answers cite file name and page or section for every claim. When the answer is not in my files, the app says so instead of guessing. Adding a new file updates the index without rebuilding everything.

HOW TO WORK

Work through every stage below without waiting for me. Verify each stage with a real test before you start the next. Stop only when you need something only I can do: give test questions, judge a result, or approve a change. Explain each handoff in plain English. Keep a TEST_LOG.md with every check, its result and the exact error if it failed.

STAGE 1: INSPECT AND PREPARE

Inspect my operating system, CPU, RAM, GPU and free disk space. Record them. Check that Ollama is installed and running; if not, guide me through the official installer. Create an isolated Python environment for this project only.

Pull EmbeddingGemma (embeddinggemma) for embeddings. Read its current model card and record the exact input format it expects for search queries and for documents; use those formats. Pick a local chat model from the Ollama library that fits my RAM and record why. Install Chroma as the vector database, stored on disk in this project. Record all versions.

STAGE 2: EXTRACT AND SPLIT

Extract text from every file in the documents folder (TXT, MD, PDF, DOCX). Keep the file name, page number or heading, and file modified date with every piece of text. List files that produced no text, such as scanned PDFs, and skip them with a clear note.

Split the text into passages of a few hundred words, at paragraph boundaries, with a small overlap so an idea that spans two passages is not lost. Save a sample of passages to a file I can read.

STAGE 3: EMBED AND STORE

Turn each passage into an embedding (a list of numbers that represents its meaning) with EmbeddingGemma. Store each embedding in Chroma with its text, file name, page and a hash of the file. Use the same embedding model for documents and questions, always.

Make indexing incremental: on each run, embed only new or changed files and remove passages from deleted files. Show the number of files, passages and the time taken.

THE BUILD PROMPT · PART 2

CONTINUE THE SAME MESSAGE

Continue the same message.

BUILD PROMPT · PART 2 COPY INTO YOUR AGENT

STAGE 4: MAKE SEARCH GOOD FIRST

Build search: embed my question with the query format, retrieve the top results from Chroma, and show each passage with its file, page and similarity score. Add an Open file button for each result.

HANDOFF: ask me for my five test questions and the file each answer is in. Run them. Report for each question whether the right passage was in the top five. If fewer than four pass, adjust passage size and overlap, re-index, and rerun. Record each attempt and its score in the test log. Do not add the chat model until search passes.

STAGE 5: ADD GROUNDED ANSWERS

Add an Answer button. Send the chat model my question and only the top retrieved passages, each labelled with its source. Instruct it to answer only from those passages, to cite the source label after every claim, and to reply "I could not find this in your documents" when the passages do not contain the answer. Show the answer above the passages it used, so I can check it.

Limit the passages sent to what fits comfortably in the chat model's context window. Show how many passages were used.

STAGE 6: MAKE IT EVERYDAY

Create a simple local web page with a search box, results, the Answer button and a Re-index button. Show when the index was last updated. Let me change the documents folder in settings. Keep the database in a visible folder outside the build files.

Run every acceptance test in this pack. Mark each PASS, FAIL or NOT RUN in TEST_LOG.md. Repair failures before you add any feature.

DELIVER

The command or launcher I use to start the app, the project folder, the database folder, the models and versions used, my test questions with their scores, a requirements lock file, a short README and the test log. List which checks still need my judgement. Keep everything local; no cloud APIs and no telemetry.

WHEN SOMETHING BREAKS

MISTAKES TO AVOID

Give this page to the agent when something goes wrong.

ADDING THE CHAT MODEL BEFORE SEARCH WORKS

If search returns the wrong passages, the answer is wrong however good the model is. Test retrieval on its own first, with questions you know the answers to.

PASSAGES THAT ARE TOO BIG OR TOO SMALL

Huge passages blur several ideas into one embedding. Tiny ones lose the context that makes them findable. Start with a few hundred words, test, then adjust.

LOSING THE SOURCE

A passage without its file name and page cannot be checked. Store the source with every passage from the first run.

MIXING EMBEDDING MODELS OR FORMATS

Documents and questions must be embedded with the same model. If you change the model, rebuild the whole index. EmbeddingGemma also expects different input formats for queries and documents; read the model card and use them.

REBUILDING EVERYTHING EVERY TIME

Re-embedding 2,000 files to add one note wastes minutes. Hash each file and update only what changed.

ANSWERS THAT SOUND RIGHT BUT ARE NOT IN YOUR FILES

A chat model fills gaps from its training. Tell it to answer only from the passages, cite every claim, and say when the answer is missing. Then check a few answers against the cited passage.

STUFFING TOO MANY PASSAGES INTO THE PROMPT

More context is not always better. Too many passages push past the context window or bury the right one. Send the best few.

SCANNED PDFS THAT LOOK INDEXED

A scanned page has no text to extract. Without a check, it silently disappears from search. List skipped files.

INDEXING PRIVATE OR SENSITIVE FILES BY ACCIDENT

Point the tool at a folder you chose, not your whole drive. Start with copies.

PROVE IT WORKS

ACCEPTANCE TESTS

Use your own documents and questions.

T01	CLEAN START	Start the app from the launcher. Expect the search page and the index status.	PASS · FAIL · NOT RUN
T02	INDEX REPORT	Run indexing. Expect a count of files and passages, and a list of skipped files with reasons.	PASS · FAIL · NOT RUN
T03	EXACT WORDS	Search a phrase copied from a document. Expect that passage first.	PASS · FAIL · NOT RUN
T04	DIFFERENT WORDS	Ask a test question using words that are not in the source. Expect the right passage in the top five. This is the test that proves semantic search.	PASS · FAIL · NOT RUN
T05	TEST SET	Run all five test questions. Expect at least four with the right source in the top five.	PASS · FAIL · NOT RUN
T06	GROUNDED ANSWER	Ask for an answer. Expect citations after every claim. Open one cited source and confirm it says what the answer says.	PASS · FAIL · NOT RUN
T07	NOT IN MY FILES	Ask something your documents do not cover. Expect "I could not find this in your documents".	PASS · FAIL · NOT RUN
T08	NEW FILE	Add one new file and re-index. Expect only that file to be processed, and its content to be searchable.	PASS · FAIL · NOT RUN
T09	DELETED FILE	Delete a file and re-index. Expect its passages to disappear from results.	PASS · FAIL · NOT RUN
T10	OFFLINE	Disconnect from the internet, restart the app, search and ask. Expect both to work.	PASS · FAIL · NOT RUN

For every test record: date, app version, steps, expected result, actual result, PASS / FAIL / NOT RUN.

FIX IT · LEARN FROM IT

REPAIR PROMPT AND EXPERIMENTS

USE THIS REPAIR PROMPT

REPAIR PROMPT — COPY INTO YOUR AGENT

The failed test is [test ID].

I did: [exact steps, including the question I asked].

Expected: [result and the file it should come from].

Actual: [what came back, or the exact error].

Find whether the failure is in extraction, splitting, retrieval or the answer step. Make the smallest repair and rerun this test plus my five test questions. Keep my documents untouched. Update the test log.

EXPERIMENTS THAT TEACH YOU HOW RETRIEVAL WORKS

Change one thing at a time and rerun your five test questions.

01 Search "heart attack" in notes that only say "myocardial infarction". That is meaning-based search working.

02 Double the passage size, then halve it. Watch your test score move.

03 Send the chat model one passage, then ten. See where answers get worse.

04 Ask a question with no answer in your files, with and without the "only from these passages" instruction. That difference is why grounding matters.

KEEP THIS WITH THE PROJECT

SOURCES, LIMITS AND BUILD RECORD

SOURCES

Ollama
https://ollama.com

EmbeddingGemma on Ollama
https://ollama.com/library/embeddinggemma

EmbeddingGemma model card
https://ai.google.dev/gemma/docs/embeddinggemma

Chroma
https://docs.trychroma.com

LIMITS

This pack is an agent-assisted build brief; run the tests on your own computer. Local models are smaller than cloud models, so answers can be weaker; the citations let you check them. Scanned documents need OCR, which is not included. Check each model's licence before you use the tool commercially. Coding-agent access has its own cost.

YOUR BUILD RECORD

Operating system / RAM / GPU _____

Embedding model / chat model / versions _____

Passage size and overlap _____

Test-set score (out of 5) _____

Documents folder _____

Database folder _____

First real question answered _____

BUILD SOMETHING. THEN UNDERSTAND HOW IT WORKS.

Change one thing at a time. Switch a model. Change an instruction. Measure how long something takes. Read the error. That is where the learning happens.

ALL FIVE STARTER PACKS · FREE · NO EMAIL NEEDED

01	AUDIOBOOK GENERATOR	Text-to-speech · Text preparation · Long-job engineering
02	OFFLINE DICTATION APP	Speech recognition · Local inference · Desktop packaging
03	PERSONAL KNOWLEDGE SEARCH	Embeddings · Semantic search · Retrieval (RAG)
04	AI RECEIPT TRACKER	Multimodal AI · Structured output · Validation in code
05	AI FILE ORGANISER	AI agents · Tool calling · Safe permissions

Get all five at drtolu.com/resources/guides-and-prompts