AI Financial Advisor Build Guide

A complete walkthrough for non-technical operators.

Built with Claude Code · Forked-friendly · Free to share

Contents

Section

Part 1 — Build Guide

A complete step-by-step. End state: a personal fiduciary advisor that reads your portfolio, knows your investing rules, remembers every conversation, and pings your phone with a morning brief.

You don't need to be a developer. If you can copy-paste, you can build this.

Two paths — pick one

	Tier 1	Tier 2
What you get	Chat with an AI that knows your rules + portfolio	Full dashboard, charts, live prices, Telegram alerts
Time to set up	5 minutes	30 minutes
Code written by you	None	None
Best for	Tonight, just to try it	The full version in the video

Both work with either AI builder: **Claude Code** (Anthropic) or **Codex** (OpenAI). Pick whichever you have access to.

Prerequisites

- macOS, Linux, or Windows (WSL recommended on Windows)
- Python 3.11+ — check with `python3 --version`
- Node.js 18+ — only needed to install the AI CLI
- One of:
- **Anthropic API key** — https://console.anthropic.com (for Claude Code)
- **OpenAI API key** — https://platform.openai.com (for Codex)
- Telegram account (free) — only for Step 5

Step 1 — Write your investor one-pager

This is the single most important step. The AI is downstream of this document — without it you have a chatbot, with it you have an advisor.

See @@0@@ for the full template, examples, and how to think about each section.

Output: a file named `philosophy.md` in your project folder. ~1 page. ~30 minutes of writing.

Step 2 — Create your project folder

In your terminal:

```
mkdir ai-advisor && cd ai-advisor
```

Create three files inside the folder:

`philosophy.md`

The one-pager you wrote in Step 1.

`portfolio.json`

Your current holdings. Use this template — replace the zeros with your real numbers:

```
{
  "holdings": [
    {"ticker": "BTC",  "asset_class": "crypto",    "shares": 0, "avg_cost": 0, "current_price": 0},
    {"ticker": "ETH",  "asset_class": "crypto",    "shares": 0, "avg_cost": 0, "current_price": 0},
    {"ticker": "QQQ",  "asset_class": "stock",     "shares": 0, "avg_cost": 0, "current_price": 0},
    {"ticker": "NVDA", "asset_class": "stock",     "shares": 0, "avg_cost": 0, "current_price": 0},
    {"ticker": "GLD",  "asset_class": "commodity", "shares": 0, "avg_cost": 0, "current_price": 0}
  ],
  "cash_usd": 0,
  "context": {
    "age": 0,
    "country": "",
    "reporting_currency": "USD",
    "income_source": "",
    "goals": "",
    "risk_tolerance": "high",
```

```
    "time_horizon": "30+ years",
    "current_advisor_offer": "",
    "notes": ""
  }
}
```

`asset_class` must be one of: `crypto`, `stock`, `commodity`. Add as many holdings as you want.

`.env`

```
ANTHROPIC_API_KEY=sk-ant-...
OPENAI_API_KEY=sk-...
TELEGRAM_BOT_TOKEN=
TELEGRAM_CHAT_ID=
```

You only need the API key for the AI builder you choose. Telegram lines stay empty until Step 5.

Step 3 — Install your AI builder

Pick ONE. Both produce the same result on the same prompt.

Option A — Claude Code (Anthropic)

```
npm install -g @anthropic-ai/claude-code
```

Then in your project folder:

```
claude
```

Option B — Codex (OpenAI)

```
npm install -g @openai/codex
```

Then in your project folder:

```
codex
```

Both tools open a terminal-based chat. They read all files in your folder and can write new ones.

Step 4a — Tier 1: Stop here if you want

You already have a working advisor. With your AI CLI open in the folder, paste this prompt:

> *I'm starting our advisory relationship. Read my philosophy.md and portfolio.json. Confirm you have my context, then ask me the one question that matters most given my current situation.*

The AI reads both files, internalizes them, and starts the relationship. Ask it questions. It answers, grounded in your philosophy and current holdings.

This is a real product. Anyone can do this in 5 minutes. No dashboard, no database, no Telegram. Just two files and a CLI.

To upgrade to the visual dashboard, continue to Step 4b.

Step 4b — Tier 2: Build the full dashboard

Open your AI CLI in your project folder. Paste the build prompt verbatim. **The full prompt is in @@0@@.**

Short version of what the prompt asks the AI to build:

- `app.py` — Streamlit dashboard (dark high-end theme)
- `analyzer.py` — model wrapper with morning brief, market scan, chat, and Telegram morning update functions
- `memory.py` — SQLite database for conversation history + portfolio snapshots
- `prompts.py` — McKinsey-grade system prompts that inject your philosophy + portfolio + chat history
- `prices.py` — live prices via yfinance (stocks/ETFs) + CoinGecko (crypto)
- `telegram_alert.py` — Telegram sender
- `requirements.txt` and `.streamlit/config.toml`

The build takes 5–10 minutes. Watch the AI work — it'll explain what it's doing as it goes.

When the AI says it's done, install dependencies and launch:

```
python3 -m venv .venv
```

```
.venv/bin/pip install -r requirements.txt
.venv/bin/streamlit run app.py
```

Your browser opens to `http://localhost:8501`. The dashboard is live.

For one-click launches on macOS, ask the AI to also generate a `Start Dashboard.command` file you can double-click from Finder.

Step 5 — Telegram integration

This is the hero shot — your phone buzzing with a morning brief.

Set up the bot (3 minutes)

1. **Open Telegram** → **search "BotFather"** (the verified blue-check result). Tap **Start**.
2. **Send** @@**0**@@. BotFather asks for a name (the display name) and a username (must end in `bot`, e.g. `myadvisor_bot`).
3. **Copy the token.** BotFather replies with a long string starting with numbers, e.g. `7891234567:AAEx...`. Treat this like a password.
4. **Message your bot first.** Tap the `t.me/yourbot` link in BotFather's reply. Tap **Start**, send any message. **A bot cannot message you until you message it first** — this is the #1 reason "it doesn't work."
5. **Get your chat ID.** Search Telegram for `@userinfobot`. Tap **Start**. It replies with your user ID (a number like `1994296173`). That's your `TELEGRAM_CHAT_ID`.

Wire it into your `.env`

```
TELEGRAM_BOT_TOKEN=7891234567:AAEx...
TELEGRAM_CHAT_ID=1994296173
```

Test

- On the dashboard, click **Send Morning Update**.
- Phone should buzz within 10 seconds.
- If silent → you almost certainly skipped step 4 above. Open the bot chat, send a message, retry.

Step 6 — How memory works

Three things get injected into every model call:

1. Your `philosophy.md` (your rules — static, edited rarely)
2. Your `portfolio.json` (your current state — edited daily-ish)
3. Last 30 messages from `memory.db` (your conversation history — auto-appended every turn)

That's the entire trick. No vector DB, no embeddings, no fine-tuning. Structured context, every call.

See @@0@@ for the full architecture, the SQLite schema, how to extend with semantic recall, and how to upgrade to Supabase + pgvector in production.

Step 7 — Use it

The dashboard has four buttons:

• **Generate Morning Brief** — full structured fiduciary analysis (5 insights an AUM advisor would have hidden)

• **Market Scan** — live web-search-grounded scan of overnight moves and how they affect your positions

• **Send Morning Update** — tight market-led brief (MARKETS / IMPACT / PORTFOLIO / SUGGESTED) sent live to your Telegram

• **Reset Memory** — clears chat history (keeps portfolio snapshots)

Below the buttons, a chat surface. Ask anything. Past conversations are saved automatically and referenced in future calls.

Try this on the second day:

> *"What did we decide about NVDA last time?"*

The AI pulls the prior recommendation verbatim, assesses whether it's still valid given today's market state, and follows up. **That's the moment your dashboard beats a $14K/year human advisor.**

Levelling up (V2)

When you're ready to take this further:

Live portfolio data

Replace hand-edited `portfolio.json` with one of:

- **Sharesight API** (paid, multi-broker aggregator — best for global investors with mixed brokers)
- **Plaid** (US) / **Basiq** (AU) — bank + broker aggregation
- **Direct broker/exchange APIs** — Interactive Brokers (TWS), Coinbase, Binance, Kraken, Hyperliquid
- **Google Sheet as source of truth** — Zapier pushes broker CSVs into a Sheet daily, dashboard reads from Sheets API

Semantic memory (Supabase + pgvector)

SQLite gives you "last 30 messages." pgvector gives you "find conversations *about* concentration risk." Migrate the schema to Supabase, add embeddings on save, and replace `get_recent_messages` with similarity search. See MEMORY_GUIDE.md.

Scheduled morning briefs

Two ways to fire a daily brief without clicking the button:

- **macOS LaunchAgent** — local cron, only fires when laptop is awake
- **Cloud cron** (GitHub Actions, Fly.io, Cloudflare Workers) — fires reliably; needs your secrets in cloud env

---

Troubleshooting

"It doesn't work" — Telegram silent after clicking Send Morning Update

You skipped Step 5.4. The bot can't DM you until you DM it first. Open the bot chat, send "hi", retry.

Live prices showing as static / numbers wrong

- yfinance occasionally returns broken data for delisted/repurposed tickers (e.g. SNDK pre-2025 acquisition history). Drop the offending ticker.
- Crypto: only the five tickers in `prices.py` (BTC, ETH, SOL, HYPE, TAO) are mapped to CoinGecko IDs by default. Add more if needed in `CRYPTO_IDS`.

Web search unavailable error

The Anthropic web_search tool needs to be enabled on your account. Go to console.anthropic.com → Settings → check Web Search is on. Without it, the dashboard falls back to model knowledge (no live headlines).

Streamlit port 8501 in use

Either kill the existing process (`lsof -i :8501` to find PID, then `kill <PID>`) or run on a different port: `streamlit run app.py --server.port 8502`.

"Module not found" errors

Activate the venv before running: `source .venv/bin/activate` (macOS/Linux) then `streamlit run app.py`.

What you've built

A locally-running, persistent, market-aware AI fiduciary advisor that:

- Knows your written rules
- Reads live market data
- Remembers every conversation
- Talks back through your phone

Total cost: API spend (~$5–20/month at typical use) + Telegram (free) + Yahoo Finance / CoinGecko (free).

Total code you wrote: zero.

Section

Part 2 — The Investor One-Pager (Step 1 deep dive)

This is the most important step in the whole build. Skip it and you have a chatbot. Do it well and you have an advisor.

The AI is downstream of this document. Every recommendation, every insight, every portfolio call is filtered through the rules you write here. A model with no philosophy doc gives generic advice. A model with a sharp one gives advice that actually fits *you* — your goals, your risk, your tax situation, your blind spots.

What it is

One markdown file. ~1 page. ~30 minutes of writing. Saved as `philosophy.md` in your project folder.

You write it once. You revise it quarterly (or after a meaningful life change). The AI reads it on every call.

How to think about it

You're not writing a thesis. You're not writing for an audience. You're writing the rulebook *you* would hand to a human advisor on day one — the things that, if they didn't know, would result in advice that doesn't fit your life.

Be specific. "Long-term thinker" is useless. "30-year horizon, no withdrawals before age 50, will not panic-sell below -40% drawdown" is useful.

Every rule in your one-pager should be testable. If the AI gives you advice and you have to ask "does this match my philosophy?" — your philosophy isn't sharp enough.

The structure

Use this exact structure. The AI is trained to look for these section headings.

```
# Investor One-Pager — [Your Name]

**Last updated:** [date]
**Operating base:** [city, tax jurisdiction]

## North Star
[1 paragraph. What are you optimising for, over what horizon, with what floor?]

## Core Philosophy
[3-5 bullet rules. Conviction style, cycles vs tickers, liquidity stance, etc.]

## Time Horizon
[How you separate short / medium / long-term capital. What rules apply where.]

## Mindset Rules
1. [Numbered list of pre-commitments. Be specific.]
2. ...

## Personal Nuances
- **Tax:** [your jurisdiction + relevant treatments]
- **Income & liquidity:** [monthly cash flow, do you draw from the portfolio]
- **Time budget:** [hours/week available for management]
- **Public-narrative blind spot:** [things you over-buy because you talk about them]
- **Family:** [non-negotiable allocation constraints]
- **[Other rules specific to you]**

## Anti-Portfolio
What you won't buy regardless of upside:
- [bullet]
- [bullet]

---
*Reviewed [cadence]. Material changes require [cooldown] before execution.*
```

How to fill in each section

North Star

The single sentence that, if you forget everything else, you should remember.

Bad: *"Achieve financial freedom through diversified investing."*

Good: *"Compound aggressively through this decade, then transition toward capital preservation and yield. Optimise for asymmetric upside in technology shifts while keeping a floor of uncorrelated assets large enough that I could sleep through a 60% drawdown without changing how I live."*

What makes the second one work: it has a phase change baked in (compound → preserve), it names the strategy (asymmetric upside), and it's testable (could you sleep through 60% down?).

Core Philosophy

3-5 bullet rules about HOW you invest, not WHAT you buy. Style, not picks.

Examples that work:

- *Conviction over diversification — 5-7 high-conviction positions beat 30 mediocre ones.*
- *Cycles, not tickers — trade the macro regime; ignore daily ticker noise.*
- *Liquidity is a feature, not a drag — dry powder IS a position.*
- *Boredom is alpha — most days the correct action is nothing.*
- *The portfolio is downstream of the thesis — no thesis → no position.*

If your bullets sound like generic finance Twitter, rewrite them. They should sound like *you*.

Time Horizon

How you slice your capital by holding period. Most people run 2-3 books in parallel and don't realise it.

Example: *"Three books. Tactical (months) — small. Core (years) — mid-size. Generational (decade+) — largest. Each has its own exit triggers and psychological posture. Mixing them is how people blow up."*

Add a paragraph explaining when you'd allow capital to flow between books. (Probably: never, except after thesis completion or a regime change.)

Mindset Rules

Numbered list. Pre-commitments to your future self. The AI uses these to gut-check every recommendation.

Examples that work:

1. *Never sell into capitulation. Never buy into euphoria.*
2. *Pre-commit exit levels in writing before entering any position.*
3. *If a thesis takes more than two sentences to explain, it isn't a thesis yet.*
4. *Cut losers on thesis break, not on price action.*
5. *Sleep test — if a position is keeping me up at night, it is already too big.*

6. *No leverage on long-term holds. Ever.*
7. *The market doesn't owe me a recovery. Size accordingly.*

Aim for 5-9 rules. More than 10 and you won't follow them.

Personal Nuances

This is where most one-pagers fall flat — and where the AI gets its biggest edge over a human advisor.

Cover these explicitly:

- **Tax** — your jurisdiction, relevant treatments (US capital gains, AU CGT discount, UK ISAs, Portugal NHR, Singapore exemptions, etc.). The AI will reason about timing realisations only if it knows the rules you're operating under.
- **Income & liquidity** — what your business / job throws off, whether you DCA from cashflow, whether you ever draw from the portfolio. Determines whether the AI suggests holding cash vs deploying it.
- **Time budget** — hours/week you can spend managing. Disqualifies any recommendation that requires more than your bandwidth.
- **Public-narrative blind spot** — assets you talk about publicly (Twitter, podcasts, YouTube). You over-weight what you champion. Tell the AI to flag this.
- **Family** — spouse risk tolerance, dependants, non-negotiable allocations. A working "boring" allocation for your spouse's peace of mind is a feature, not a constraint.
- **Anything else specific to you** — morning routine ("don't check prices before noon"), conference behaviour ("anything I get excited about at a conference goes in a 30-day cooldown"), past mistakes you don't want to repeat.

The more specific these are, the better the advice gets.

Anti-Portfolio

What you'll never buy regardless of upside. This is where you save your future self.

Examples:

- *Anything I can't explain to a non-finance friend in 60 seconds.*
- *Founders I wouldn't share a meal with.*
- *Assets that require leverage to be interesting.*
- *"Story" stocks with no plausible cash-flow path within 5 years.*
- *Anything pitched to me at a conference dinner.*

- *Anything where the loudest believers are people I don't respect.*

This list is the most underrated part of the doc. The AI will refuse to recommend things that violate it.

Common mistakes

1. Writing for an imagined reader

You're not writing for a financial blog audience. Cut anything that sounds like content marketing. If a sentence wouldn't cause your future self to make a different decision, delete it.

2. Hedging

"I might consider rebalancing if the market becomes overvalued" → useless. The AI can't act on that. Either commit ("I will trim positions above 25% of NW unless thesis active") or remove the rule.

3. Over-engineering

Five rules you actually follow > twenty rules you don't. Be honest about what you'll commit to.

4. Generic risk language

"High risk tolerance" is meaningless. Write the actual numbers: max acceptable drawdown, max position size, max single-asset-class concentration.

5. Forgetting tax

This is the single highest-leverage section. A US resident, an Aussie with US stocks, a Portugal NHR, a Singapore PR — the optimal portfolio is wildly different. The AI can only reason about tax if you tell it your situation explicitly.

Example — full populated one-pager

Pulled from the working demo (anonymised). Use as a structural reference, not as your actual rules — yours need to be yours.

```
# Investor One-Pager — Jordan Reeve
```

```
**Last updated:** 14 Apr 2026
**Operating base:** Lisbon (NHR resident through 2027)

## North Star
Compound aggressively through this decade, then transition toward capital preservation and yield. Optimi

## Core Philosophy
- Conviction over diversification. 5–7 high-conviction positions beat 30 mediocre ones.
- Cycles, not tickers. Trade the macro regime; ignore daily ticker noise.
- Liquidity is a feature, not a drag. Dry powder IS a position.
- Boredom is alpha. Most days the correct action is nothing.
- The portfolio is downstream of the thesis. No thesis → no position. No exception.

## Time Horizon
Three books in parallel — tactical (months), core (years), generational (decade+). Each has its own rule

## Mindset Rules
1. Never sell into capitulation. Never buy into euphoria.
2. Pre-commit exit levels in writing before entering a position.
3. If a thesis takes more than two sentences to explain, it isn't a thesis yet.
4. Cut losers on thesis break, not on price action. Let winners run to thesis completion.
5. Sleep test: if a position is keeping me up at night, it is already too big.
6. No leverage on long-term holds. Ever.
7. The market doesn't owe me a recovery. Size accordingly.

## Personal Nuances
- **Tax:** Portugal NHR through 2027 — meaningfully changes optimal timing of realisations.
- **Income & liquidity:** Business throws off ~$15k/mo gross, ~$10k net. Covers living costs with margin
- **Time budget:** I cannot hold anything that demands more than ~2 hrs/week of active management.
- **Public-narrative blind spot:** I overweight things I write about publicly. Any position I've champio
- **Family:** Spouse is risk-averse. A meaningful share of NW stays in "boring" assets she'd consider sa
- **Mornings:** I don't check prices before noon. Mornings are for building, not reacting.
- **Conferences:** Anything I get excited about at a conference goes in a 30-day cooldown folder before

## Anti-Portfolio
- Anything I can't explain to a non-finance friend in 60 seconds
- Founders I wouldn't share a meal with
- Assets that require leverage to be interesting
- "Story" stocks with no plausible cash-flow path within five years
- Anything pitched to me at a conference dinner
- Anything where the loudest believers are people I don't respect

---
*Reviewed every Sunday evening. Material changes require a 48-hour cooldown before execution.*
```

When you're done

Save the file as `philosophy.md` in your project folder. Move to BUILD_GUIDE.md Step 2.

The first time you talk to the AI advisor after writing this, ask it:

> *"Read my philosophy.md. Quote three rules back to me verbatim and explain how each one will shape your future advice."*

If it can't do that crisply, your one-pager isn't sharp enough yet. Revise.

Section

Part 3 — The Build Prompt to Paste

Open Claude Code in an empty folder. Drop in your `portfolio.json` and `philosophy.md` (templates below). Paste this prompt. Hit enter.

The prompt to paste

> *Build me a personal AI fiduciary advisor as a Streamlit dashboard with persistent memory. I have two input files in this folder:* `portfolio.json` *(my current holdings + cash + context) and* `philosophy.md` *(my investor one-pager — rules, anti-portfolio, time horizons, tax situation).*
>
> ***Stack:*** *Python 3.11+,* `streamlit`*,* `anthropic` *(model* `claude-opus-4-7`*, max_tokens 4096),* `python-telegram-bot==20.7`*,* `plotly`*,* `python-dotenv`*. SQLite for memory (no external DB).*
>
> ***Files to create:*** *-* `app.py` *— Streamlit dashboard, dark high-end theme -* `analyzer.py` *— Claude wrapper with two functions:* `analyze_portfolio()` *(structured JSON morning brief) and* `chat_stream()` *(streaming Q&A) -* `memory.py` *— SQLite (*`memory.db`*) with three tables:* `messages`*,* `portfolio_snapshots`*,* `events`*. Helpers:* `init_db`*,* `save_message`*,* `get_recent_messages`*,* `save_snapshot`*,* `count_messages`*,* `clear_conversation`*,* `load_philosophy` *-* `prompts.py` *—* `FIDUCIARY_SYSTEM_PROMPT` *(returns JSON: summary + 5 insights + morning_brief_markdown) and* `CHAT_SYSTEM_PROMPT` *(free-form). Both inject* `philosophy.md` *as* `{philosophy}` *and the portfolio as* `{portfolio_json}`*. -* `telegram_alert.py` *— async send + sync wrapper, MarkdownV2-safe -* `.streamlit/config.toml` *— dark theme*
>
> ***Dashboard sections, top to bottom:*** *1. Hero metrics (Total Value · Unrealized P/L · Cost Basis · Cash) 2. Portfolio Value time-series chart (Plotly, range selector 1W/1M/3M/6M/1Y/ALL, area fill) 3. Asset class donut + per-class drilldowns (cards + horizontal bars) 4. Memory panel — pill* `SQLite · memory.db`*, stats: Philosophy words*

/ Messages / Snapshots / Last activity 5. Quick actions: Generate Morning Brief · Send to Telegram · Reset Memory 6. Latest brief insight cards (when present) 7. Conversation thread (rendered from SQLite) 8. `st.chat_input` *anchored at the bottom*

>

Memory rules: *- Every chat turn auto-saves both Q and A - Every Morning Brief click saves a portfolio snapshot - Every Claude call's system prompt = philosophy + portfolio + last 30 messages - Stream the chat response live so I see Claude generating it*

>

Env vars in @@0@@: `ANTHROPIC_API_KEY`*,* `TELEGRAM_BOT_TOKEN`*,* `TELEGRAM_CHAT_ID`*.*

>

Create a `Start Dashboard.command` *file (chmod +x) that activates a venv and runs* `streamlit run app.py` *so I can launch by double-clicking. Build it. Ask only if something is genuinely ambiguous.*

Input file templates

`portfolio.json`

```
{
  "holdings": [
    {"ticker": "BTC",  "asset_class": "crypto",    "shares": 0, "avg_cost": 0, "current_price": 0},
    {"ticker": "QQQ",  "asset_class": "stock",     "shares": 0, "avg_cost": 0, "current_price": 0},
    {"ticker": "GLD",  "asset_class": "commodity", "shares": 0, "avg_cost": 0, "current_price": 0}
  ],
  "cash_usd": 0,
  "context": {
    "age": 0,
    "country": "",
    "reporting_currency": "USD",
    "income_source": "",
    "goals": "",
    "risk_tolerance": "",
    "time_horizon": "",
    "current_advisor_offer": "",
    "notes": ""
  }
}
```

philosophy.md

Use the structure below. Replace every section with your real answers. Keep it to one page.

```
# Investor One-Pager — [Your Name]

**Last updated:** [date]
**Operating base:** [city, tax jurisdiction]

## North Star
[One paragraph: what you're optimising for, over what horizon, with what floor.]

## Core Philosophy
- [3–5 bullet rules — conviction style, cycles vs tickers, liquidity stance, etc.]

## Time Horizon
[Tactical / core / generational books — how you separate them.]

## Mindset Rules
1. [Numbered list of pre-commitments. Be specific.]

## Personal Nuances
- **Tax:** [your jurisdiction]
- **Income & liquidity:** [monthly cash flow, whether you draw from the portfolio]
- **Time budget:** [hrs/week available for management]
- **Public-narrative blind spot:** [things you over-buy because you talk about them]
- **Family:** [non-negotiable allocation constraints]
- **[Other rules specific to you]**

## Anti-Portfolio
What you won't buy regardless of upside:
- [bullet]
- [bullet]

---
*Reviewed [cadence]. Material changes require [cooldown] before execution.*
```

Why this works

The advisor isn't smart because of the model. It's smart because of the **context**: it has your written rules (philosophy), your current state (portfolio), and your conversation history (memory). Every Claude call gets all three injected into the system prompt. That's the entire trick.

Section

Part 4 — Memory System Deep Dive

The AI's edge over a $14K/year human advisor isn't the model. It's the memory.

A stateless prompt is a search engine. An advisor with persistent context across conversations is something else entirely. This document explains how that works, what the system does on every call, and how to extend it.

The mental model

Three things get injected into every model call:

```
[ system prompt ]
   ■■■ Your investor philosophy (philosophy.md)
   ■■■ Your current portfolio (portfolio.json)
   ■■■ Last 30 messages (memory.db)

[ your question ]
```

That's the entire trick. No vector database, no embeddings, no fine-tuning, no agents. Just structured context, every call.

When you ask *"what did we decide about NVDA last time?"* — Claude or Codex doesn't *remember* in any neural sense. It reads the last 30 messages from your SQLite file, sees the prior NVDA discussion, and responds with reference to it. Memory is just data, fed back in.

The three layers

Layer 1 — Philosophy (static identity)

`philosophy.md` is a markdown file you write once and edit rarely. It contains your North Star, mindset rules, anti-portfolio, tax situation. See STEP_1_PHILOSOPHY.md for how to write it well.

In code: `memory.load_philosophy()` reads the file fresh on every call. You can edit `philosophy.md` between conversations and the next call picks it up automatically.

Injected into every system prompt. The AI is told: *"This is the operator's written investment philosophy. Treat it as governing law for every recommendation."*

Layer 2 — Portfolio (current state)

`portfolio.json` is your holdings. Edited daily-ish (or auto-updated from broker APIs once you wire that up).

Live prices flow through `prices.py`:

- **Stocks/ETFs** → yfinance (free, no API key)
- **Crypto** → CoinGecko free public API

The dashboard fetches live prices on every page load (cached 5 min) and overwrites the static `current_price` in your portfolio dict. The model sees the current state, not the static snapshot.

Layer 3 — Conversation history (chat memory)

This is where the SQLite database comes in.

File: `memory.db` in your project folder. Created automatically on first launch.

Tables:

```
CREATE TABLE messages (
  id              INTEGER PRIMARY KEY AUTOINCREMENT,
  ts              TEXT    NOT NULL,
  role            TEXT    NOT NULL,    -- 'user' or 'assistant'
  content         TEXT    NOT NULL,
  conversation_id TEXT    NOT NULL DEFAULT 'default'
);

CREATE TABLE portfolio_snapshots (
  id              INTEGER PRIMARY KEY AUTOINCREMENT,
  ts              TEXT    NOT NULL,
  total_value_usd REAL    NOT NULL,
  total_gain_pct  REAL,
  json_snapshot   TEXT    NOT NULL    -- full portfolio.json contents
);

CREATE TABLE events (
  id      INTEGER PRIMARY KEY AUTOINCREMENT,
  ts      TEXT    NOT NULL,
  kind    TEXT    NOT NULL,           -- 'morning_brief', 'telegram_sent', etc.
  payload TEXT                        -- JSON blob
);
```

On every chat turn:

1. User question gets saved to `messages` (role = 'user')
2. The last 30 messages are pulled from `messages` and passed to the model as conversation history
3. The model's response streams back, gets accumulated, and saved to `messages` (role = 'assistant')

On every Generate Morning Brief click:

1. Full portfolio analysis runs
2. A row is added to `portfolio_snapshots` with the total value and full portfolio JSON
3. An event is logged
4. Over time, snapshots accumulate into a real performance history

On Reset Memory click: all rows in `messages` are deleted. Snapshots are kept (they're historical record, not chat).

Where it lives in the code

```
# memory.py
def init_db(): ...
def save_message(role, content): ...
def get_recent_messages(n=30): ...
def save_snapshot(portfolio, total_value, total_gain_pct): ...
def load_philosophy(): ...
def clear_conversation(): ...
def log_event(kind, payload): ...


# analyzer.py — chat_stream
philosophy = memory.load_philosophy()           # Layer 1
portfolio_json = json.dumps(portfolio)          # Layer 2
history = memory.get_recent_messages(n=30)      # Layer 3

system_prompt = CHAT_SYSTEM_PROMPT.format(
    philosophy=philosophy,
    portfolio_json=portfolio_json,
    today=date.today(),
)
messages = history + [{"role": "user", "content": question}]

# Stream response
with client.messages.stream(
    system=system_prompt,
```

```
        messages=messages,
        tools=[{"type": "web_search_20250305", ...}],
    ) as stream:
        for text in stream.text_stream:
            yield text
```

That's the whole thing. Read it once, you understand the system.

Why this works without semantic search

For most personal-advisor use cases, **chronological recall is sufficient**. The advisor only needs to remember what you discussed in the last few sessions. "What did we decide about NVDA last time?" is answered by scanning the last 30 messages — they're already loaded.

Semantic search (pgvector, embeddings) starts to matter when:

- You have hundreds of past sessions and want to surface old discussions
- You ask exploratory questions like "find conversations *about* concentration risk"
- You want similar-positions retrieval ("when have we faced this kind of macro setup before?")

For Tier 2 (the dashboard build), SQLite + last-30-messages is the right call. For Tier 5 (production), see "Levelling up" below.

How to inspect the memory

The cinematic move is opening `memory.db` in a SQLite viewer alongside the dashboard.

Free tools:

- **DB Browser for SQLite** (Mac/Windows/Linux) — https://sqlitebrowser.org
- **TablePlus** (free tier) — https://tableplus.com
- **Beekeeper Studio** (free, cross-platform)

Open `memory.db`, browse the `messages` table, see the rows. Cut to dashboard, ask a question, watch a new row appear in real time. That's the proof the memory is real, not theatre.

How to extend

Resetting memory between sessions

The Reset Memory button on the dashboard does this. Programmatic equivalent:

```
import memory
memory.clear_conversation()
```

Multiple conversation threads

Every message has a `conversation_id` (default = `"default"`). To run separate threads — say, one for tactical book and one for generational book — pass different IDs:

```
memory.save_message("user", question, conversation_id="tactical")
history = memory.get_recent_messages(n=30, conversation_id="tactical")
```

Add a thread selector to the dashboard if you want this in the UI.

Editing memory directly

Open `memory.db` in any SQLite viewer and edit rows. The next chat call reads whatever is there. Useful for:

- Removing a bad answer the model gave
- Correcting a hallucinated past decision
- Trimming long messages that bloat the context

Pruning old messages

The system loads the last 30 messages. Older ones still exist in `memory.db` but aren't passed to the model. To bound the database size, periodically:

```
DELETE FROM messages
WHERE id NOT IN (
  SELECT id FROM messages
  ORDER BY id DESC
  LIMIT 1000
);
```

(Keeps the most recent 1,000 messages, drops the rest.)

Levelling up — Supabase + pgvector for semantic recall

When you're running this in production for real and have months of conversations, semantic recall starts to pay off.

Migration in three steps

1. Spin up a Supabase project (free tier) — https://supabase.com

2. Migrate the schema:

```
create extension if not exists vector;

create table messages (
  id              bigserial primary key,
  ts              timestamptz not null default now(),
  role            text not null,
  content         text not null,
  embedding       vector(1536),
  conversation_id text not null default 'default'
);

create table portfolio_snapshots (
  id              bigserial primary key,
  ts              timestamptz not null default now(),
  total_value_usd numeric not null,
  total_gain_pct  numeric,
  json_snapshot   jsonb not null
);

create index on messages using ivfflat (embedding vector_cosine_ops);
```

3. On @@0@@, also embed:

```
from anthropic import Anthropic
import os
import numpy as np

# Use a text-embedding API (OpenAI, Voyage, Cohere, etc.)
def embed(text: str) -> list[float]:
    # Implementation depends on provider
    ...

def save_message(role, content, conversation_id="default"):
    embedding = embed(content)
```

```
    supabase.table("messages").insert({
        "role": role,
        "content": content,
        "embedding": embedding,
        "conversation_id": conversation_id,
    }).execute()
```

4. On retrieval, use semantic search instead of last-30:

```
def get_relevant_messages(question: str, k: int = 8) -> list[dict]:
    query_embedding = embed(question)
    # Find top-k most similar past messages
    response = supabase.rpc("match_messages", {
        "query_embedding": query_embedding,
        "match_count": k,
    }).execute()
    return response.data
```

Combine with last-N for hybrid recall: pull last 10 messages chronologically (for short-term continuity) PLUS top 8 most semantically similar from history (for long-term recall).

When this is worth doing

- You've accumulated 200+ past messages
- You ask exploratory questions that aren't answered by recent context
- You want the AI to surface "we made this same mistake six months ago" moments

For most people running this for personal use, SQLite is fine for the first year.

Common pitfalls

Memory growing too large

After hundreds of messages, the chat history can crowd context. Mitigations:

- Periodically summarise older messages and replace 50 chronological messages with one "summary" message
- Use semantic recall (above) so old messages are retrieved by relevance, not chronology
- Cap conversation length and start a new `conversation_id` quarterly

Memory that's too literal

The model can over-anchor on past discussions ("you said you'd trim NVDA but you haven't yet"). Counter this in the system prompt: *"Past conversations are reference, not commitments. Re-evaluate current state independently."* The build prompt (BUILD_PROMPT.md) already includes this framing.

Cross-conversation leakage

If you run multiple `conversation_id` threads (tactical / generational), one thread can leak into another if you forget to filter. Always pass the conversation_id explicitly when saving and retrieving.

What you actually built

A 3-layer memory system that:

- Reads your written rules on every call (Layer 1: philosophy)
- Reads your live state on every call (Layer 2: portfolio + live prices)
- Reads your conversation history on every call (Layer 3: SQLite messages)

Total infrastructure: one markdown file, one JSON file, one SQLite file, three Python functions.

That's it. That's the trick.

Back to BUILD_GUIDE.md.