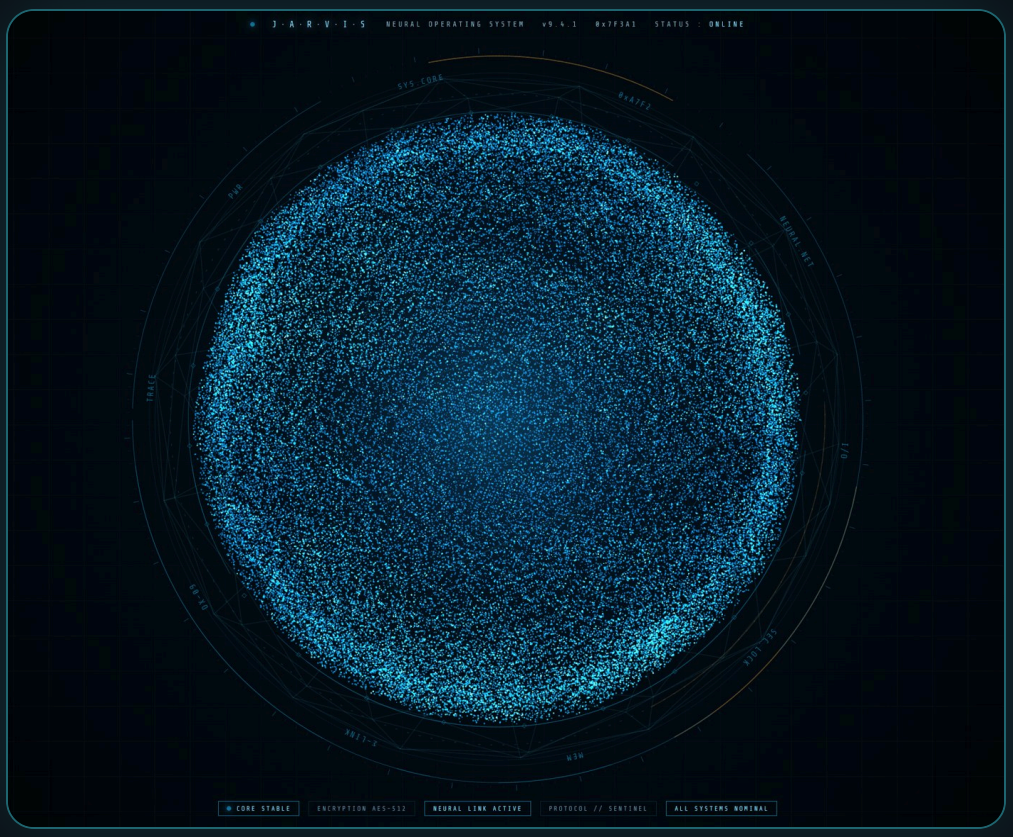
THE COMPLETE BUILDER BLUEPRINT · VERSION 2.0



BUILD YOUR OWN

J.A.R.V.I.S.

The Complete Builder Blueprint

From an empty folder to a voice-controlled, memory-bearing AI assistant — built by you.

VERSION 2.0

ManinaLabs

jarvis.driftworksstudios.com

FRONT MATTER

Copyright & License

Copyright

“Build Your Own J.A.R.V.I.S. — The Complete Builder Blueprint” and its contents, including all text, diagrams, prompts, and code samples, are the intellectual property of the author.

J.A.R.V.I.S. is used here as a generic descriptive term for an Iron-Man-style voice assistant archetype. This product is an independent educational work and is not affiliated with, endorsed by, or sponsored by any film studio or rights holder.

Personal Use License

This blueprint is licensed to a single purchaser for personal, non-commercial use. By purchasing or using this document you agree to the following:

- **PERMITTED.** You may build, modify, and operate your own private assistant for personal use based on this material.
- **NO REDISTRIBUTION.** You may not share, post, upload, or otherwise distribute this document or any substantial portion of it.
- **NO RESALE.** You may not resell, sublicense, or repackage this material, in whole or in part.
- **NO SHARING DOWNLOADS.** You may not share your download link, account, or files with any other person.
- **ATTRIBUTION OF THIRD-PARTY TOOLS.** This guide directs you to third-party software and services (e.g. Python, Anthropic's Claude, Coqui XTTS, ChromaDB). Each is governed by its own license and terms; you are responsible for complying with them, including any voice-data rights (see “A Note on Voice”).

- **NO WARRANTY.** This material is provided “as is,” without warranty of any kind. You assume all responsibility for how you build and operate your assistant, including security, privacy, and costs incurred from paid APIs.

> ⚠ **WARNING**
>
> This project can run commands on your computer, control your screen, and access your files. Build it on a machine you control, keep your API keys private, and review the Security chapter (Page 90) before granting it broad automation power.

Support

Questions, errata, and build help: support@driftworksstudios.com

Please include your build milestone (see Page 107) and the exact error text when asking for help.

EDITION	Second Edition (v2.0) · 2026
PUBLISHED BY	ManinaLabs — a Driftworks Studios project
WEB	jarvis.driftworksstudios.com

FRONT MATTER

A Letter From the Creator

I built JARVIS because I wanted the thing every Iron Man fan secretly wants: a calm, capable voice that knows me, runs my machine, and answers before I finish the question. Not a chatbot in a browser tab — a presence in the room.

What started as a weekend experiment with a text-to-speech model turned into a year-long obsession. I cloned a voice. I wired up a microphone that listens for its name. I gave it memory so it stopped forgetting me between sessions. I taught it to open apps, drive a browser, watch my system for intrusions, and speak up when something needed my attention. I built it a face — a glowing orb that breathes with the sound of its own voice. Piece by piece, it stopped feeling like software and started feeling like JARVIS.

Along the way I made every mistake there is to make. I let it transcribe its own voice and answer itself in a loop. I watched it hang forever on a network call with no timeout. I shipped a personality prompt so eager to be witty that it sounded like a stand-up comedian instead of a butler. I corrupted model downloads by storing them in a cloud-synced folder. Each failure taught me a rule, and those rules are the real value of this book.

The Goal of This Blueprint

This blueprint exists so you don't have to repeat that year. It is the build I wish I'd been handed on day one: the architecture, the exact prompts, the model choices, the voice-training recipe, and — just as importantly — the specific gotchas that will otherwise cost you days.

It is written to be built **WITH** an AI coding assistant (Claude Code). You will not be copying thousands of lines of code by hand. Instead, you'll give your AI assistant precise, engineered instructions — provided here, ready to paste — and it will generate the implementation. Your job is to understand each piece, run it, verify it, and move on. That's why this guide explains WHY before HOW: when you understand the design, you can direct the build and fix it when reality disagrees with the plan.

What to Expect

This is a technical product for a technical reader. You do not need to be an expert — if you can install Python, edit a text file, and run a command in a terminal, you can finish this. But you should expect real work: a focused builder gets a talking assistant in a weekend and a polished, voice-cloned JARVIS in a week or two of evenings.

You will build it in twelve phases, each with a clear milestone you can test before moving on. By the end you will have something genuinely yours — not a clone of my assistant, but your own, in a voice you choose, that knows your name.

Let's build.

— Sam Manina, ManinaLabs

FRONT MATTER

Contents

GETTING STARTED

THE BUILD

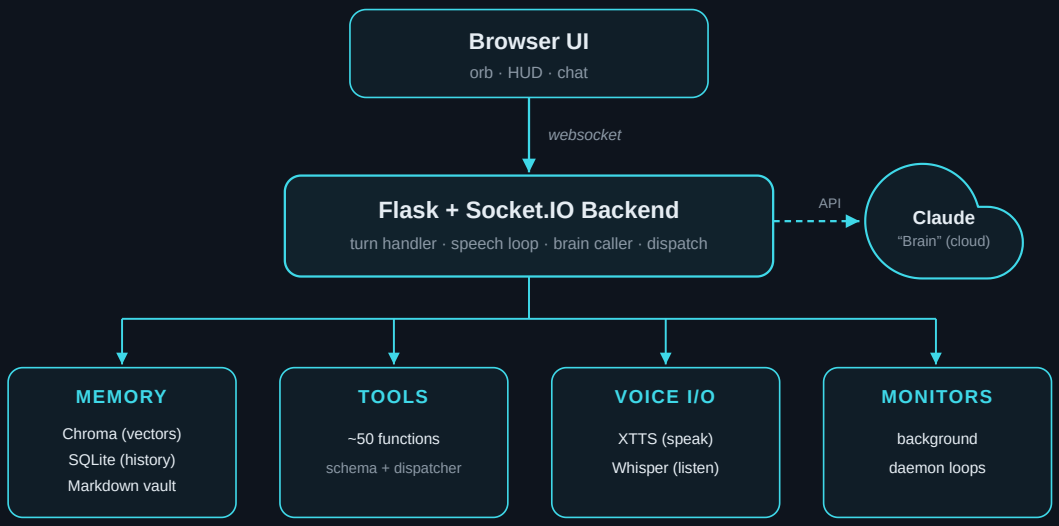
REFERENCE & APPENDICES

FRONT MATTER

What You Are Actually Building

Before any setup, let's be concrete about the finished product. This is not a toy. By the end of this blueprint you will have a complete, always-on assistant running on your own PC with the following capabilities.

- **VOICE-CONTROLLED AI ASSISTANT.** It boots quietly, listens for its name, and holds a real spoken conversation in a cloned voice you choose. You speak; it answers in character, out loud, within a few seconds.
- **PERSISTENT MEMORY.** It remembers facts across restarts — your preferences, your projects, things you told it last week — and brings the relevant pieces back into each conversation automatically.
- **TOOL CALLING.** The brain doesn't just talk; it acts. It can run system commands, check your calendar and email, set reminders, look up weather and news, do math, and more — choosing the right tool on its own.
- **BROWSER CONTROL.** It drives a real web browser to accomplish goals you state in plain language — searching, navigating, and acting on pages.
- **ORB INTERFACE.** A holographic web UI with a particle orb that reacts to the assistant's own voice, live telemetry panels (CPU, GPU, network, weather), and a chat thread — with standby, awake, and workspace modes.
- **RESEARCH CAPABILITY.** It can answer questions with live web data and run a longer background research task while you do other things.
- **MODULAR ARCHITECTURE.** Everything is split into clean, single-responsibility modules so you can understand, extend, and debug each part in isolation.



Four supporting pillars branch from one backend process; the cloud brain is called over the network.

Figure 1. High-level architecture. The Browser UI connects over a websocket to a central Flask + Socket.IO backend. Below the backend, four pillars branch out — MEMORY (Chroma + SQLite + Markdown vault), TOOLS (~50 functions), VOICE I/O (XTTS + Whisper), and MONITORS (background daemon loops) — while the Anthropic Claude “Brain” (cloud) is called from the backend over the API.

> **NOTE**
>
> This guide deliberately ships the COMPLETE CORE assistant — and nothing half-finished. A handful of advanced add-ons (camera vision, engineering/CAD agents, meeting integration, multi-hour academic research, production cloud infrastructure) are intentionally out of scope. See “Excluded Features” (Page 98) for exactly what's not here and why. What IS here is whole.

FRONT MATTER

Before You Begin

Read this chapter fully before buying hardware or installing anything. It sets honest expectations on skills, time, and cost.

Required Skills

Here's the honest truth: you do not need to know how to code — not at all. Every line of this project is written **for** you by a “vibe coding” tool — an AI coding assistant such as Claude Code — that you direct in plain English. You describe what you want, paste in the prompts this guide provides, and it generates the implementation. You are the architect giving directions, not the bricklayer.

What genuinely matters is understanding the **structure** — how the pieces fit together and why. When something misbehaves (and it will), that mental model is what lets you tell *which* box is broken and describe it clearly enough for the AI to fix it. Coding is optional; understanding the puzzle is the real skill, and it's the one that makes troubleshooting possible. Here is the realistic skill map by build phase.

> **BEGINNER-FRIENDLY — you can do these on day one**
>
> - Installing software and Python.
> - Copying files and editing a text file.
> - Running a command in a terminal and reading its output.
> - Pasting the provided prompts into your AI coding assistant.

> **INTERMEDIATE — you'll grow into these as you build**
>
> - Reading a Python error message and pasting it back for a fix.
> - Understanding how modules import each other (we explain this).
> - Editing a configuration (.env) file and keeping secrets out of git.
> - Basic troubleshooting: “is the service running? is the port open?”

ADVANCED — only for optional polish and the expansion paths

- Tuning model/voice latency settings to taste.
- Fine-tuning the voice model (we give you the exact recipe — it's guided).
- Extending the system with your own tools or a local model later.

WHY

We build WITH an AI coding assistant precisely so a capable non-expert can finish. You direct; it writes. The skill that matters most is not coding — it is reading carefully, testing each step, and not skipping the verification checklists.

Expected Time Commitment

`First voice ("it talks back")`	a few focused hours.
`Core assistant (voice + memory + tools)`	a weekend.
`Full build incl. UI + monitors`	1–2 weeks of evenings.
`Custom voice fine-tune`	+1 evening of prep, +1–2 hrs train.

These assume you test as you go. Skipping verification to "save time" reliably costs more time later.

Estimated Build Cost

ONE-TIME

If you don't already own a capable PC — hardware dominates; see below. The software stack is free and open source.

ONGOING

- **Anthropic API usage (the brain).** With the model split + prompt caching taught in this guide, light personal use is typically a few dollars to low tens of dollars per month. Heavy use costs more. You control this.
- Everything else (voice, speech-to-text, memory, UI) runs LOCALLY and free.

NOTE

The single biggest cost lever is the model split (strong model for speech, cheap fast model for background utility work) plus prompt caching. Both are built into this blueprint. Don't skip them.

Hardware Requirements

- A **Windows 11 PC.** (The architecture is portable; the guide is Windows-first because the desktop-automation and Office integration use Windows APIs.)
- An **NVIDIA GPU with CUDA.** The cloned voice and speech recognition run on the GPU.

`8 GB VRAM`	the floor. Works well: cloud brain + local voice + local STT.
`12–16 GB VRAM`	comfortable; leaves room to add a local model later.

- A decent microphone and speakers/headphones.
- A local SSD with several GB free for models.

WARNING

Do NOT install the project inside a cloud-synced folder (OneDrive, Dropbox, Google Drive). These folders don't support the symlinks that model caches use and WILL corrupt model downloads and cripple performance. Use a local path such as `C:\AI\jarvis`.

Software Requirements

- **Python 3.10 or 3.11** (NOT 3.12+ — some audio/TTS dependencies lag behind).
- **Git.**
- **Google Chrome** (used for browser automation via remote debugging).
- **The Claude Code CLI** (your AI coding assistant — see next chapter).
- A code/text editor (VS Code recommended).

Claude Subscription & API Requirements

This project uses Anthropic's Claude in two distinct ways. Don't confuse them:

- **1. CLAUDE CODE (the builder).** The CLI tool you use to GENERATE the project. This runs during the build, on your machine, on your Claude plan.
- **2. THE ANTHROPIC API (the brain).** At RUNTIME, your finished assistant calls the Claude API to think and talk. This needs an API key and is billed per use.

You will create an Anthropic API key, place it in a private `.env` file, and the running assistant will use it. The key is never hardcoded and never committed.

API Cost Expectations

- Spoken replies use a strong model (quality matters for the voice). This is where most of your token spend goes.
- Background/utility calls (parsing, summaries, vision-coordinate lookups) use a fast, cheap model.
- The large personality prompt and tool list are sent as CACHED blocks, so after the first call you pay only a small fraction for them.
- Net effect: personal use is inexpensive. You set the models in one config line and can downgrade anytime.

Common Misconceptions

- ✗ **“I need to be a great programmer.”** — No. You need to follow instructions and test carefully. The AI writes the code.
- ✗ **“This runs entirely offline / free.”** — The BRAIN is a cloud API (paid). The voice, speech recognition, memory, and UI are local and free.
- ✗ **“More VRAM makes the replies smarter.”** — Reply quality comes from the cloud model. VRAM affects the LOCAL pieces (voice, STT) and a future local model.
- ✗ **“It'll sound like the movie out of the box.”** — The character comes from the personality prompt + a strong model; the VOICE timbre comes from fine-tuning on audio you provide. Both are covered.
- ✗ **“I can build this in a OneDrive folder.”** — You cannot. See the warning above.

Validation Checklist — Readiness

- ☐ Windows 11 PC with an NVIDIA CUDA GPU (8 GB VRAM minimum).
- ☐ Python 3.10 or 3.11 installed and on PATH.
- ☐ Git and Google Chrome installed.
- ☐ Claude Code CLI installed and working.
- ☐ An Anthropic API key created and saved somewhere private.
- ☐ A LOCAL project folder chosen (NOT in OneDrive/Dropbox/Drive).
- ☐ Microphone and speakers tested in Windows.

CHAPTER 1

System Architecture Overview

> **Chapter goal:** *Understand the entire system before building a single piece of it. If you grasp how the parts fit, every later chapter becomes "fill in this box," and debugging becomes "which box is misbehaving?"*

1.1 The Big Picture

JARVIS is a Python program that runs continuously on your PC. It opens a web page (the UI) in your browser, listens to your microphone, thinks using a cloud AI model, speaks back in a cloned voice, and can act on your computer through a set of tools. Around all of this run background "senses" — loops that watch your system and speak up when something matters.

Everything communicates through one backend process. The UI talks to it over a websocket. The microphone feeds it audio. The brain (Claude) is called over the network. Tools are Python functions it can invoke. Memory is a local database it reads and writes.

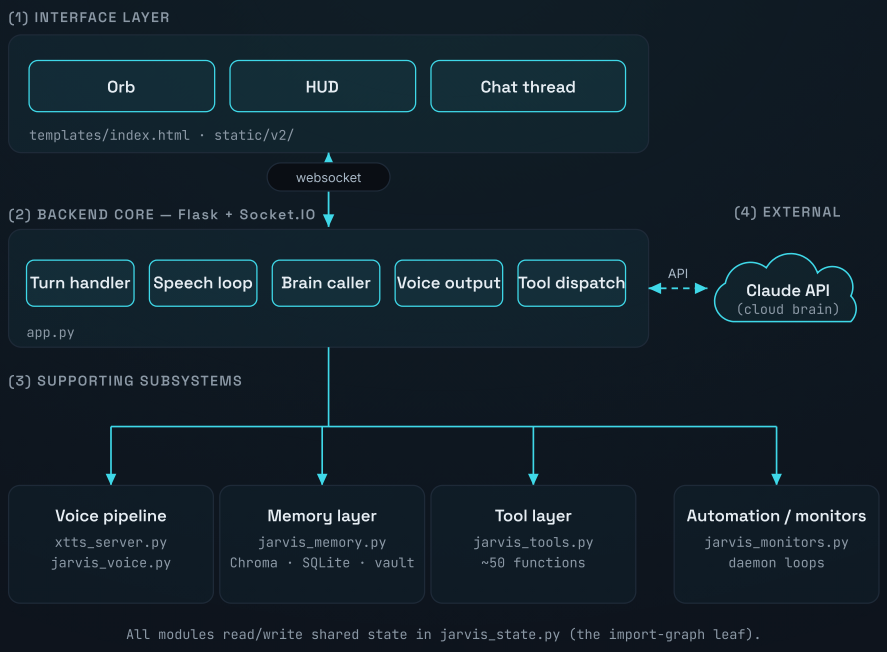


Figure 1.1. High-level architecture. The browser UI connects over a websocket to the Flask + Socket.IO backend (`app.py`); four supporting pillars — voice pipeline, memory layer, tool layer, and automation/monitors — branch beneath the core, and the Claude "brain" is reached as a cloud network call.

1.2 The Six Subsystems

1) Core brain

The decision-maker. It receives your words plus relevant memory, calls the Claude API, streams back a reply, and decides when to call tools. It uses two models: a strong one for spoken replies, a fast cheap one for background work. (Building the Brain, Page 33.)

2) Voice pipeline

Two halves. **Input:** microphone → speech recognition (Whisper) → text. **Output:** reply text → cloned voice (XTTS) → audio. The output runs in its own subprocess so a voice crash never takes down the assistant. (Voice System Page 39, Input System Page 46.)

3) Memory layer

Three tiers: recent conversation (SQLite), semantic recall (a vector database that finds relevant past facts), and a human-readable markdown "vault" of saved knowledge. On each turn, relevant memories are retrieved and given to the brain. (Memory System, Page 53.)

4) Tool layer

The hands. A catalogue of ~50 functions the brain can call — run a command, set a reminder, get the weather, control music, drive the browser, and so on. A schema tells the brain what's available; a dispatcher runs each call. (Tool Calling, Page 59.)

5) User interface

The holographic web page: a voice-reactive particle orb, live telemetry panels, and a chat thread, with three display modes. It is driven by the backend, not by keystrokes. (Orb Interface, Page 77.)

6) Automation layer (the senses)

Background daemon loops that run forever: a security monitor, a system-health monitor, clipboard and file watchers, a weather updater, and the telemetry broadcaster that feeds the UI. They speak through one shared channel so they never talk over a reply. (Diagnostics Page 85.)

1.3 How a Single Turn Flows

Understanding one conversational turn end-to-end is the key mental model.

1. You say *"JARVIS, what's the weather?"*
2. The **mic** captures audio; the **speech loop** detects you finished speaking.
3. Whisper transcribes the audio to text.
4. The **turn handler** retrieves relevant **memory** and prepends it.
5. The **brain** calls Claude, streaming the reply token by token.
6. The brain decides to call the `get_weather` **tool**; the dispatcher runs it and feeds the result back to the brain.
7. The brain produces the final spoken sentence(s).
8. As each sentence completes, it's sent to the **voice output**, synthesized in the cloned voice, and played — while the next sentence is still forming.
9. The UI **orb** reacts to the audio; the chat thread shows the text; the turn is saved to **memory**.

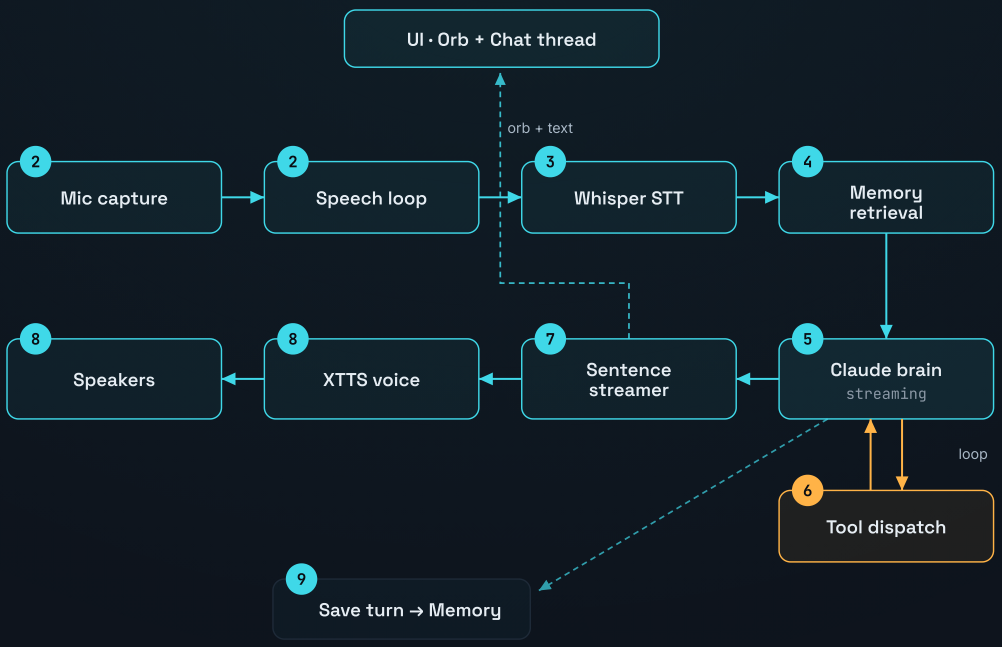


Figure 1.2. One turn, end to end. Audio flows mic → speech loop → Whisper → memory retrieval → the streaming Claude brain (with a tool-dispatch side-loop) → sentence streamer → XTTS voice → speakers. The brain's output feeds the voice *before* the reply is complete; branches push text to the UI and the finished turn to memory.

1.4 Five Design Principles

- **Modular.** Logic is split into single-purpose modules; one module holds all shared state, clients, and locks. This prevents the "7,000-line file" problem and the circular-import problems that follow it.
- **Stream everything.** The brain streams; sentences are spoken as soon as they complete. The assistant starts talking ~1–2 seconds sooner than if it waited for the whole reply.
- **Two-model split.** Spend model quality where the user hears it (speech); save cost/latency where they don't (background parsing and summaries).
- **Personality is a cached prompt.** The character lives in one large system prompt, anchored on real reference lines, sent with caching on every call.
- **Fail soft.** Any subsystem can die without crashing the assistant. A dead tool returns "couldn't do that"; a dead voice falls back to a backup voice; a crashed monitor loop is isolated from the rest.

⚠ Common Mistakes (architecture level)

- Treating the shared-state module as a dumping ground that imports other modules — it must be the **leaf** of the import graph (it imports nothing of yours). Violating this is the #1 source of circular-import errors later.
- Building features before the skeleton. Build the scaffold and brain first; bolt features on after each milestone passes.
- Blocking the whole app on one slow call (no timeouts). Every network/model call in a background loop must have an explicit timeout.

ⓘ Troubleshooting (architecture level)

- `ImportError: cannot import name … (most likely due to a circular import)` → A module imported the shared-state module's dependents, or two feature modules import each other at the top level. Move one import inside the function that uses it (a "lazy import").
- "The app hangs and stops responding." → A loop is blocked on a call without a timeout. Find the most recent network/model call and add one.

VALIDATION CHECKLIST — UNDERSTANDING

- [ ] I can name the six subsystems and what each does.
- [ ] I can trace a single turn from microphone to spoken reply.
- [ ] I understand why the shared-state module imports nothing of ours.
- [ ] I understand the two-model split and why it exists.
- [ ] I understand "fail soft" and can give two examples.

END OF FREE PREVIEW

This is just Chapter 1.

The complete **Build Your Own J.A.R.V.I.S. Blueprint** is **119 pages across 13 chapters** — the full architecture, the exact copy-paste prompts, the voice-cloning recipe, persistent memory, tool-calling, browser & desktop automation, the security layer, and the complete build checklist.

Pick up exactly where this preview leaves off and build the whole thing.

Purchase the full textbook →

jarvis.driftworksstudios.com/get-access