

Three POCs

Week-one proofs of concept for a local AI assistant

Antonio Joboy · v1.0 · 2 September 2026 · Budget: 5–10 hours

These exist to answer three questions cheaply, before any platform code is written. Each has a pass/fail bar defined *before* building, so the result can't be rationalised afterwards.

The one rule that matters: do not build Coolwind this week. Three standalone scripts, roughly 100 lines each, no shared code, no config layer, no package structure. They get deleted afterwards. If you start on the platform you will spend the entire budget there and answer none of the three questions.

Why this order

Voice first, because it is the highest-risk item. You named sound quality and latency as a kill switch — so test the thing most likely to fail on day one, not in week three. Lookup is last precisely because it is easiest: it is least likely to fail, so it tells you the least.

One exception: POC 3 doubles as a model-quality test. If you would rather settle which model to build the others around before you start, pull it forward — it is only 90 minutes.

Audio setup (applies to POC 1)

Wired headset to start. Three rules so the later move to Bluetooth costs nothing:

- **Device names in config, never in code.** `input: null` means system default.
- **Never hardcode a sample rate.** Wired USB is usually 48 kHz, Bluetooth A2DP 44.1, HFP 16. Query the device, resample to what Whisper wants.
- **Handle the device disappearing.** Catch, re-resolve, retry. You will never hit this on wired and will hit it constantly on Bluetooth.

Push-to-talk removes the need for echo cancellation entirely, because you are never listening and speaking at the same time.

POC 1 — Voice

Question: is local voice fast enough and good enough that you would rather talk than type?

Budget: 2-3 hours.

Build

```
hotkey down  -> start capture
hotkey up    -> stop capture
              -> STT  (faster-whisper, base or small)
              -> LLM  (Ollama, whatever fits Sakura)
              -> TTS  (Piper)
              -> play
```

faster-whisper and Piper were the established local pair as of early 2026. Check what is current before committing — this area moves fast.

The one trick that decides this POC: stream TTS on the first complete sentence rather than waiting for the full LLM response. Buffer tokens, detect a sentence boundary, synthesise and play that sentence while the model is still generating the next. Waiting for generation to finish before speaking is what makes local voice assistants feel dead. This is the difference between roughly 6 seconds and roughly 1.5.

Instrumentation

Log four timestamps every turn. Without these you are guessing about where the time goes:

```
t0 hotkey released      (end of speech)
t1 STT returns text
t2 LLM first token
t3 first audio out      <-- the number that matters
```

Test protocol

1. Ten fixed utterances, mixed length. Record t0–t3 for each.
2. Check STT accuracy on all ten — note any word it gets wrong.
3. Then ten minutes of unstructured conversation. No script.

PASS: median t0–t3 under 2 seconds on wired, STT accurate enough that you are not repeating yourself, and after ten minutes you still want to talk to it.

FAIL: you catch yourself typing instead. That is the honest signal, and it is more reliable than the stopwatch.

Latency caveat. Your wired number is the *floor*. Bluetooth adds 100-300 ms. If wired lands at 1.9 s against a 2 s bar, you have not passed — you have passed on hardware you plan to stop using. Judge against your bar minus the Bluetooth penalty.

Do not, this week: wake-word detection, always-on listening, barge-in, echo cancellation, or voice-model auditioning. Wake word alone is months of work and it is where voice projects die. The hotkey gives you 80% of the feel; you may never need the rest.

POC 2 — Memory

Question: can it recall accurately under load, and correct itself when a fact changes?

Budget: 3 hours. The longest of the three, and the one that decides whether the project is real.

Build

Deliberately naive. Append every exchange to a markdown file; stuff the whole file into context each turn. No vectors, no database, no retrieval logic.

```
memory.md  append-only, plain text
each turn: read whole file -> prepend to context -> call model -> append exchange
```

The point is to find the failure mode, not to avoid it. Storage is trivial; **recall under load** is the actual question.

Test protocol

1. Tell it 20 facts across a session, spread out — not in one block. Mix types: names, dates, preferences, numbers, project details.
2. Do unrelated conversation in between, so the facts are buried.
3. Come back later. Ask about all 20. Score each: correct, wrong, or "doesn't remember".
4. **Then contradict one.** "Actually, I moved teams." Ask again. Does it use the new fact or the old one?
5. Note where in the file the failures sit — beginning, middle or end.

PASS: 18 or more out of 20 recalled correctly, and the contradiction handled — new fact wins, old fact not repeated.

FAIL: recalls the recent and the first few, loses the middle. That is the classic lost-in-the-middle failure and it is the signal you need real retrieval rather than a bigger context window.

Failure here is *informative, not fatal*. It tells you the memory layer needs structure — selective retrieval, a separate write-back pass deciding what is durable, facts stored as discrete entries rather than raw transcript. That is the thing worth building properly, and this POC tells you whether you have to.

Do not, this week: install a vector database, build an embedding pipeline, or design a schema. If you jump straight to vectors you will never learn what the naive version actually fails at, and you will build the wrong retrieval for the wrong problem.

POC 3 — Lookup

Question: is a locally-runnable model accurate enough to trust with fetched content?

Budget: 1-2 hours.

This POC is not testing your architecture. It is testing your model. Search plumbing is a solved problem. What you are measuring is whether the model invents details the source page did not contain. If it fails, that is a model-size answer, not a design answer — and it tells you what Sakura needs to run.

Build

```
question -> search (SearXNG self-hosted, or any search API)
          -> fetch top 2-3 results
          -> strip scripts, nav, hidden elements
          -> summarise with source text in context
          -> answer
```

Test protocol

1. Ten factual questions with verifiable answers. Include some with specific numbers, dates or version strings — those are where models invent.
2. For each answer, open the source and check every specific claim.
3. Score: correct / wrong / **invented**. Track invented separately — it is the one that matters.

PASS: 9 or more out of 10 correct, and **zero invented specifics**. One fabricated number is worse than three "I don't know"s.

FAIL: it produces plausible detail not present in the source. Try a larger model before concluding anything about the design.

Out of scope for the POC, in scope for production: fetched pages are attacker-controlled input. The production design puts fetching in a quarantined process with no tools, no memory and no filesystem, so a hostile page captures something that can only return a string. Do not wire lookup into anything with write access, even as a test.

Results

Fill this in as you go. Written before building, so the bar cannot move afterwards.

POC	Metric	Bar	Actual	Pass?
1 Voice	Median end-of-speech to first audio	< 2.0 s		
1 Voice	Still want to talk after 10 min	Yes		
2 Memory	Facts recalled correctly	$\geq 18/20$		
2 Memory	Contradiction handled	Yes		
3 Lookup	Answers correct	$\geq 9/10$		
3 Lookup	Invented specifics	0		

Reading the result

Outcome	What it means
All three pass	Build it. Start on the platform skeleton and the config contract.
Voice fails	Not fatal. Memory with no voice is still useful; voice with no memory is a novelty. Drop to text and notifications, revisit voice later.
Memory fails	The most important result. It scopes the real work — retrieval and a write-back pass — which is the actual differentiator anyway.
Lookup fails	A hardware and model-size question, not a design question. Try a larger model on Sakura before drawing conclusions.
All three fail	Genuinely useful information for eight hours. Walk away, or wait for better local models.

Time budget

Item	Hours
POC 1 — Voice	2-3
POC 2 — Memory	3
POC 3 — Lookup	1-2
Testing and scoring	1
Total	7-9

That total only holds at this scope. The moment one of these becomes "while I'm here, let me structure this properly," the week is gone and you learn nothing. Throwaway means throwaway.

Separate from these POCs: claim `coolwind` on PyPI with a minimal stub while it is free. PyPI has no reservation mechanism — a name is claimed by uploading. Ten minutes, and it is the only scarce item in the naming exercise.