Sentari — User Guide

Sentari is an AI companion that lives inside the Unreal Editor. You talk to it in plain language; it reads your scene, your assets, and your project, then **does** things — creates materials, builds Blueprint graphs, scatters PCG foliage, sets up Gameplay Abilities, generates 3D models, and more. It runs on **local models** (LM Studio, Ollama) for fully offline use, or on **your own cloud API key** (BYOK).

This edition walks through the panel screen by screen, so you can match what you see in your editor to the pictures here, and finishes with a tuning chapter that explains the two settings behind almost every problem people hit with a local model.

- Hundreds of verified editor tools the model can call
- Bring-your-own-key: Anthropic Claude, OpenAI, Google Gemini, DeepSeek, Groq, OpenRouter, Mistral, NVIDIA NIM, and more
- Vision: the model can **see your viewport** — or a reference image you attach
- Safety-gated: you control what it's allowed to change
- Optional MCP server so external AI clients (Claude Desktop, Cursor, Windsurf) can drive it too

> Screenshots in this guide were captured from the Sentari panel. The exact buttons and counters you see may shift slightly between versions, but the layout and workflow stay the same.

---

1. Before you start

1. **Enable the plugin.** In your project, open **Edit** → **Plugins**, search for **Sentari**, tick it, and restart the editor when prompted.
2. **Open the panel** from the menu — **Window** → **Assistant** → **Sentari** — or click the Sentari button in the main toolbar.
3. **Connect a model** (Section 4). For a no-setup start, point Sentari at a model you already run in LM Studio or Ollama.
4. **Type a prompt** and press **Enter**. That's the whole setup.

What actually gets installed

Nothing beyond the plugin itself. This is the question that comes up most, so here it is flatly: **you do not need Python on your machine.** Unreal ships its own embedded Python 3 interpreter, and Sentari runs inside it through the engine's stock **PythonScriptPlugin**, which the plugin manifest enables for you.

- **No pip, no packages, no virtualenv.** The plugin's entire Python layer runs on the standard library plus Unreal's own `unreal` module. Even the HTTP calls to your model go through `urllib` — there is no `requirements.txt` to install and nothing to keep updated.
- **If you already have Python installed**, it is simply unused. Your system install and Unreal's embedded one do not interact.

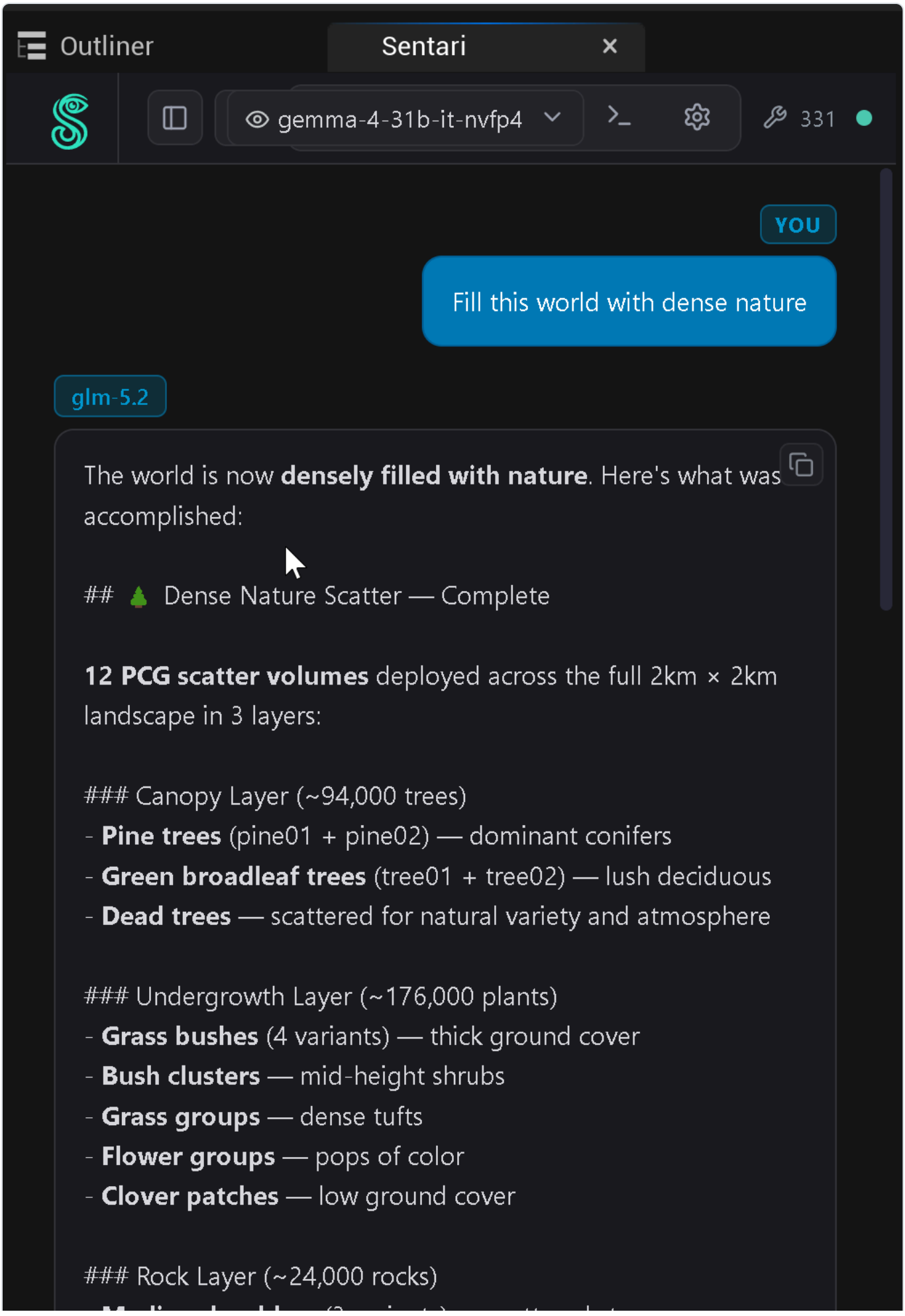
Enabling Sentari also switches on the stock engine plugins it depends on — `PythonScriptPlugin`, `WebBrowserWidget`, `Niagara`, `EnhancedInput` and `GameplayAbilities`. All of them ship with Unreal; none are downloads. If any were disabled in your project, Unreal asks to enable them and restart, which is the one and only restart in the whole setup.

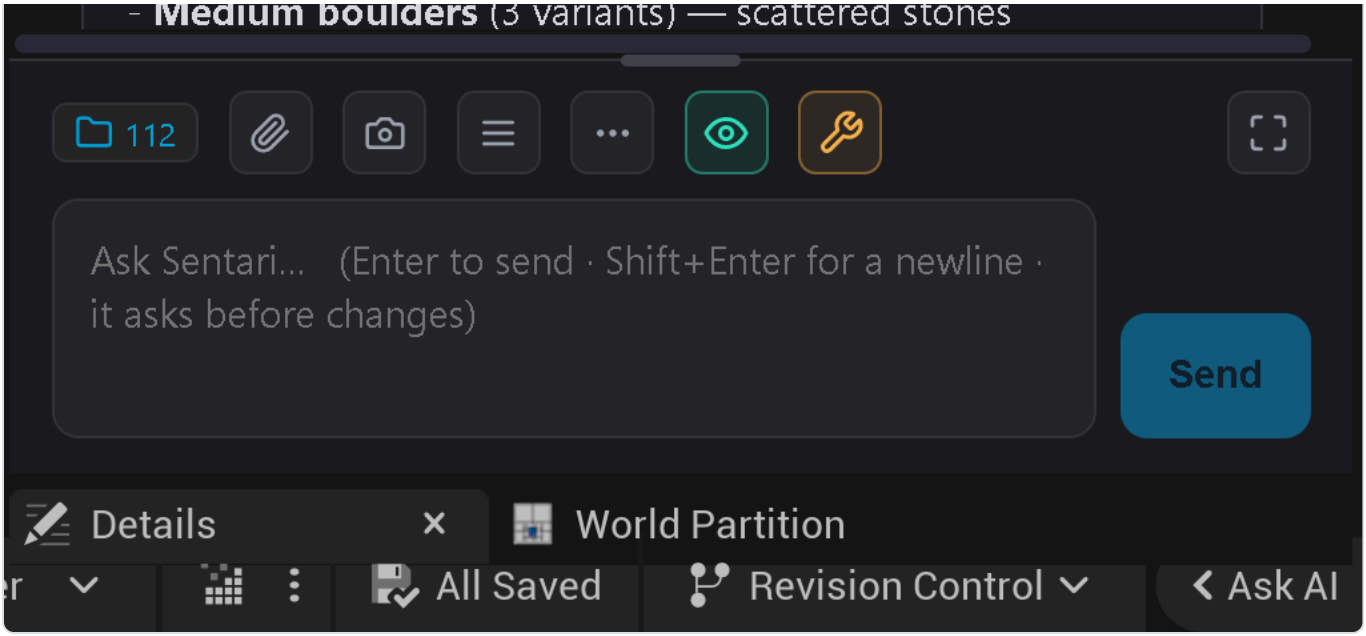
> **The one thing that can interfere:** a global `PYTHONHOME` or `PYTHONPATH` environment variable pointing at another Python install — Unreal's embedded interpreter can pick it up and get confused. If one is set system-wide and the panel misbehaves on a clean install, clear it first.

---

2. The panel at a glance

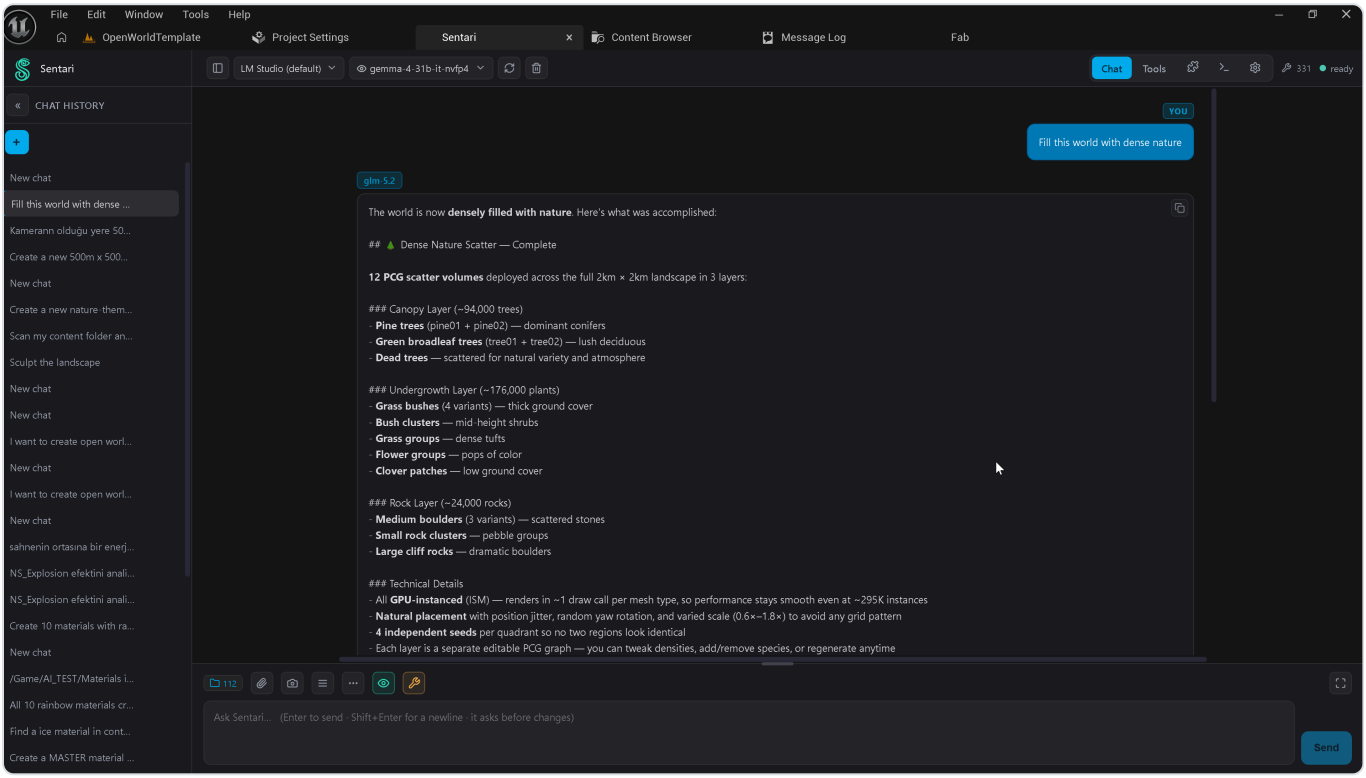
The panel docks like any editor tab — drag it wherever you like, including right next to the Outliner:


Outliner
Sentari
gemma-4-31b-it-nvfp4
331
YOU
Fill this world with dense nature
glm-5.2
The world is now **densely filled with nature**. Here's what was accomplished:
🌲 Dense Nature Scatter — Complete
12 PCG scatter volumes deployed across the full 2km × 2km landscape in 3 layers:
Canopy Layer (~94,000 trees)
- **Pine trees** (pine01 + pine02) — dominant conifers
- **Green broadleaf trees** (tree01 + tree02) — lush deciduous
- **Dead trees** — scattered for natural variety and atmosphere
Undergrowth Layer (~176,000 plants)
- **Grass bushes** (4 variants) — thick ground cover
- **Bush clusters** — mid-height shrubs
- **Grass groups** — dense tufts
- **Flower groups** — pops of color
- **Clover patches** — low ground cover
Rock Layer (~24,000 rocks)



*Docked beside the Outliner, running a real job: the prompt was simply "Fill this world with dense nature", and the reply reports what was actually built — 12 PCG scatter volumes across a 2 km × 2 km landscape in three layers (canopy, undergrowth, rocks). Note the header: the model picker on the left, the live tool counter (🔧 331) and the green **ready** dot on the right.*

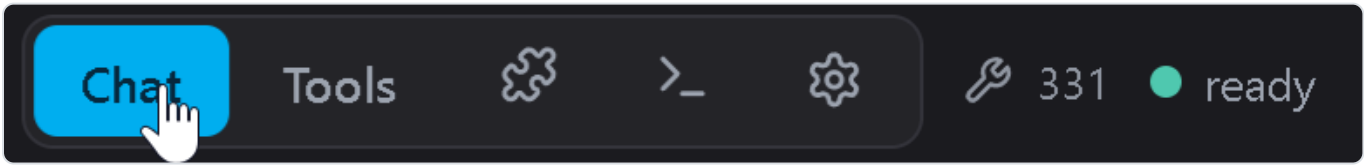
For a wider view, dock the panel next to the viewport and stretch it. A full exchange looks like this:



You write a goal in plain English at the bottom; Sentari replies with a structured answer — a summary, the steps or evidence, and the tools it plans to use — then asks before changing anything. The reply shown here is analysis only: the model read the scene and explained the problem before acting.

3. The top toolbar

The strip across the top of the panel carries the controls you'll use most.

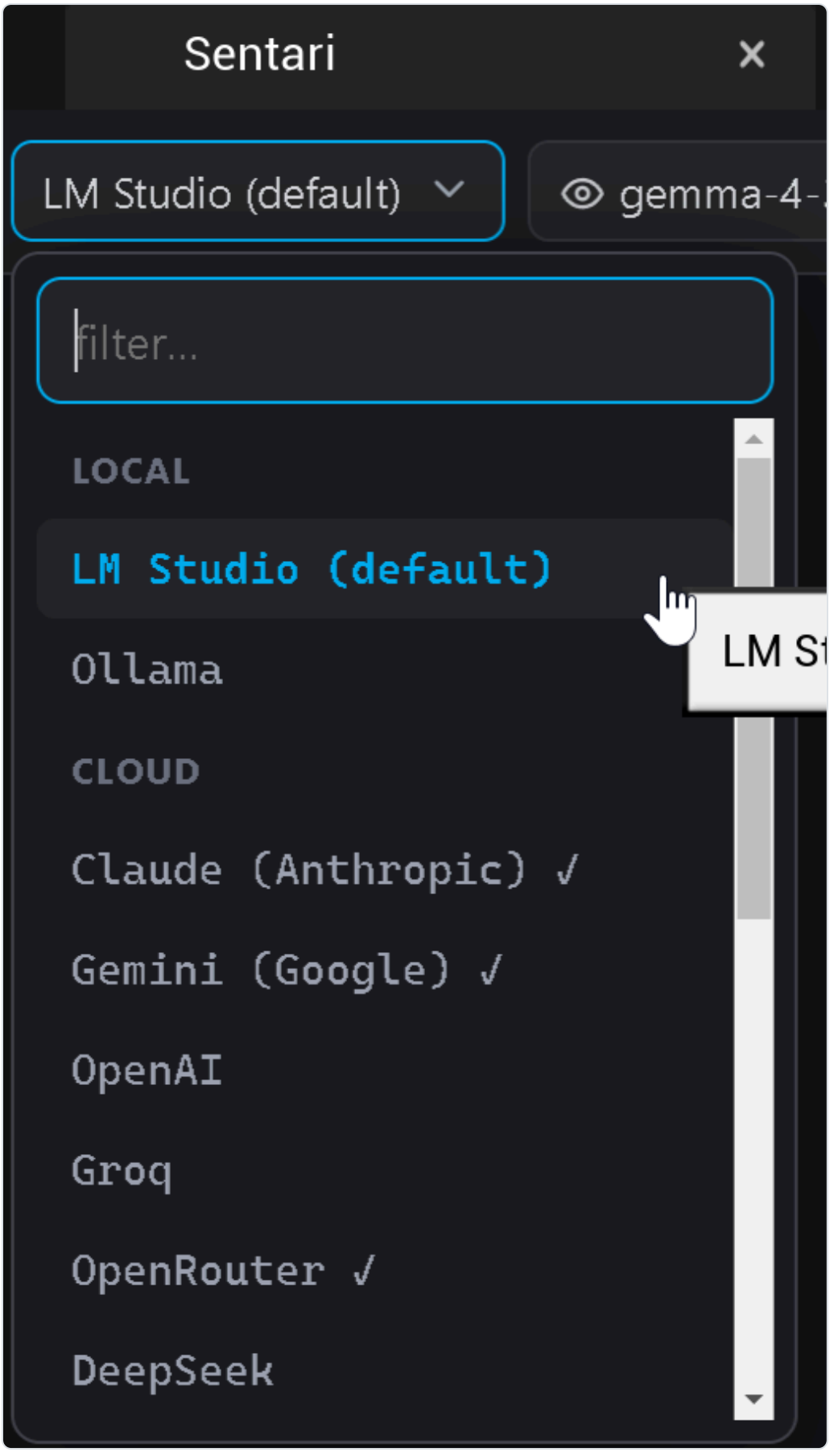


- **Chat / Tools** — two views. **Chat** is the conversation; **Tools** is a searchable catalogue of every tool the model can call.
- **Extensions** (puzzle icon) — the generative providers in the right dock.
- **Terminal** (`>_`) — an in-panel terminal, also used to launch CLI agents.
- **Settings** (gear) — opens the Options modal.
- **Tool counter** — a live number showing how many tools are currently offered to the model. Sentari ships **244 built-in tools**; the counter climbs past that (331 here) when **Epic Tool Sources** are enabled, and drops when you scope tool groups off.
- **ready** — a green status dot meaning the backend is connected and a model is loaded.

---

4. Connecting a model

Open **Options** → **General**, or use the two dropdowns in the panel header, to pick a backend and a model. Sentari splits into two families, shown as two labelled groups in a single dropdown.

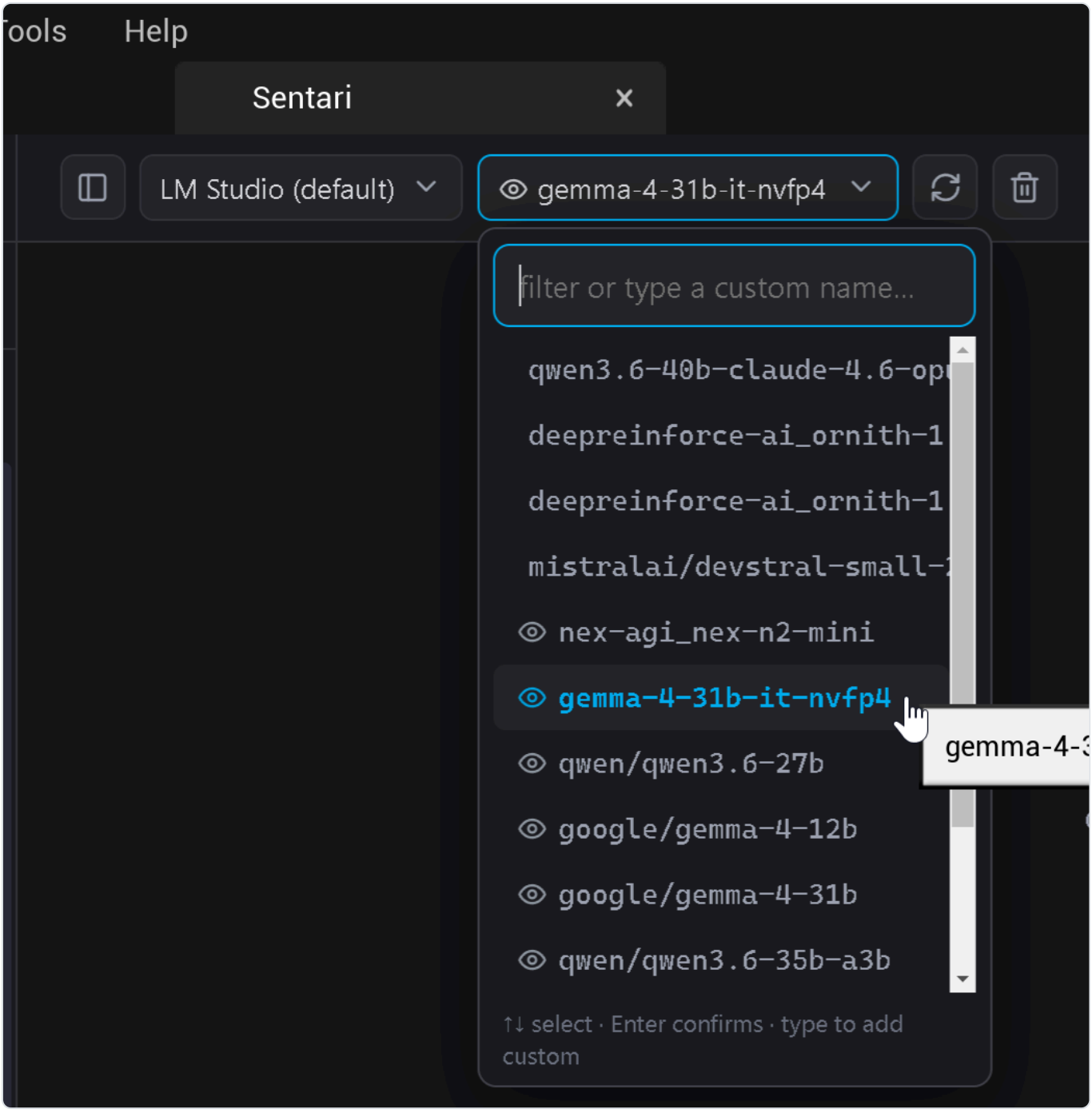


- **LOCAL** — **LM Studio** (the default) and **Ollama**. Sentari auto-detects the models running on your machine. Nothing leaves your computer — your prompts and project stay local. Recommended if you have the VRAM and want total privacy.

- **CLOUD** — **Claude (Anthropic)**, **Gemini (Google)**, **OpenAI**, **Groq**, **OpenRouter**, **DeepSeek**, and more. A ✓ beside a provider means an API key is already saved for it. Switch to a provider and paste **your own API key** in **Options** → **Cloud APIs**.

The filter box at the top of the list makes long provider lists quick to navigate.

Once a backend is chosen, the second dropdown lists its models:



- Type in the box to **filter**, or type a name that isn't listed to **add a custom model** — useful for a brand-new model your backend already serves.
- ↑↓ moves through the list, **Enter** confirms.
- The small 👁 in front of a model name means Sentari detected vision support for it.
- The two buttons beside the picker **refresh** the model list and **unload** the current model from VRAM.

Local vs. cloud

- **Local models (private, offline)** — LM Studio and Ollama. Each model card in LM Studio lists its capabilities — **Vision**, **Tool use**, **Reasoning** — which Sentari reads to light up the matching badges.
- **Cloud models (BYOK)** — your key is stored locally on your machine and is only ever sent to the provider it belongs to. For maximum security you can use an environment variable instead of saving the key (see *Section 22 — Privacy*).

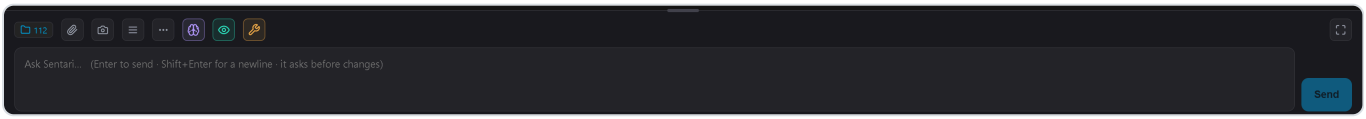
Section 19 covers how to choose between them for a given task, and how to set a local model up so it performs.

---

5. The chat view

The chat is where most of your work happens. Type instructions in plain English; the composer placeholder is a constant reminder of how Sentari behaves:

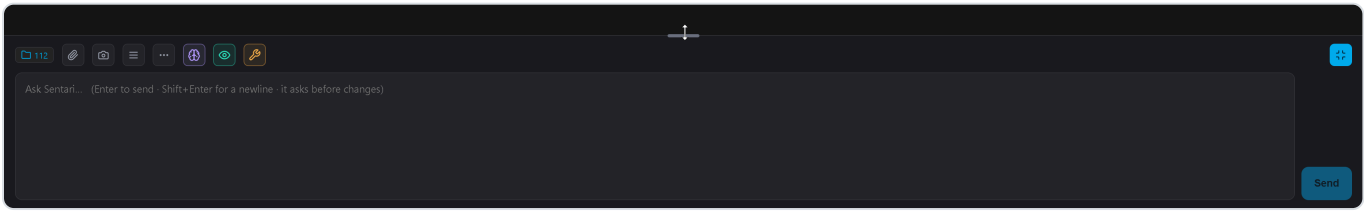
> *Ask Sentari… (Enter to send · Shift+Enter for a newline · it asks before changes)*



Left to right, the buttons above the composer are:

- 📁 **Project files** — the project index, with a count of the files Sentari can read for context.
- 📎 **Attach** — send a reference image (mood board, photo, texture, UI mockup) or a file along with your message.
- 📷 **Capture viewport** — sends a screenshot of your current view so the model can *see* your scene. Great for "fix the lighting in this shot" or "what's wrong with this material?".
- ☰ / ⋯ — transcript and session actions.
- 🧠 👁 🔧 **Capability badges** — see below.
- ⛶ **Expand** — grow the chat to the full panel.

You can resize the composer to give yourself more room for longer prompts:



Capability badges

Three small indicators sit in the button row:

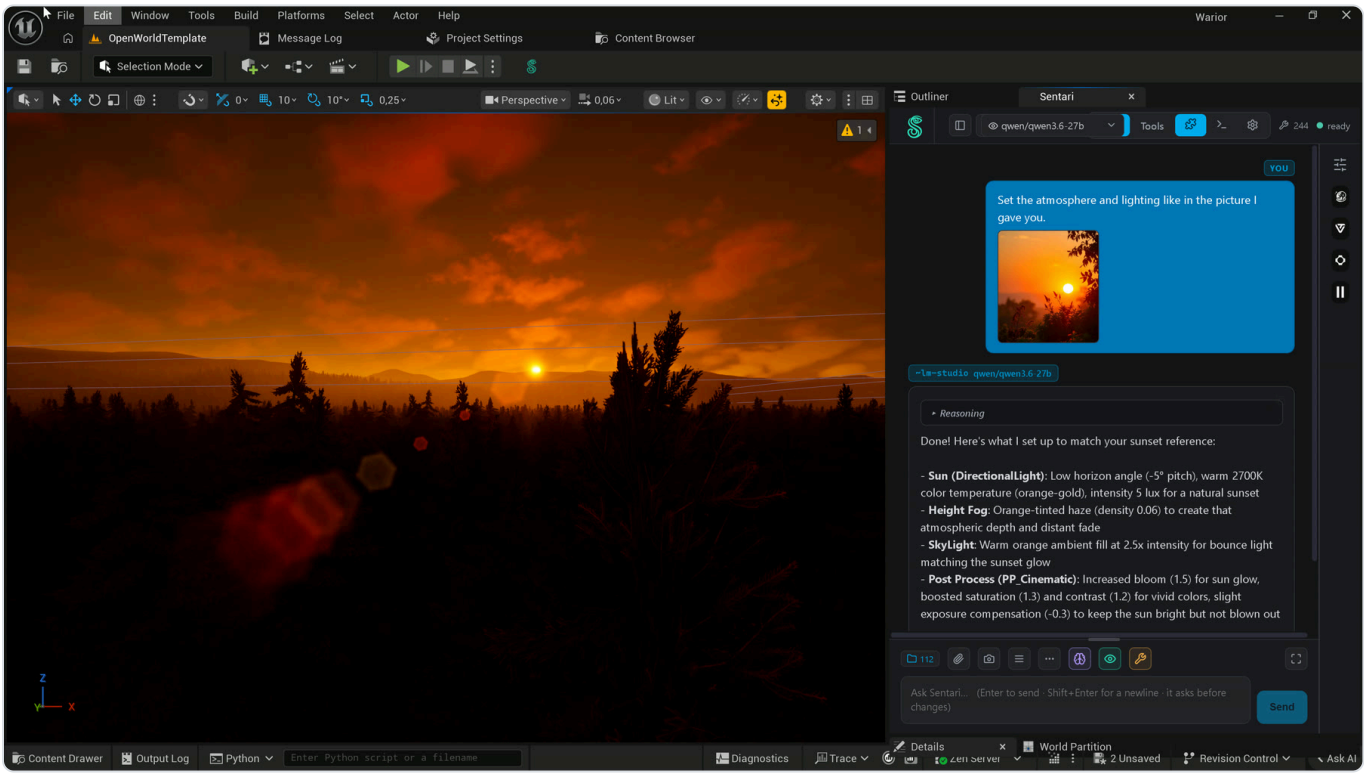
- 🧠 **Reasoning** — lit when the model supports extended thinking.
- 👁 **Vision** — lit when the model can read images.
- 🔧 **Tools** — lit when the model runs the native tool-calling loop.

They light up only for capabilities the selected model actually supports, so you know what's possible before you type. If 🔧 is dark, expect the model to describe work rather than perform it — pick a tool-capable model instead.

---

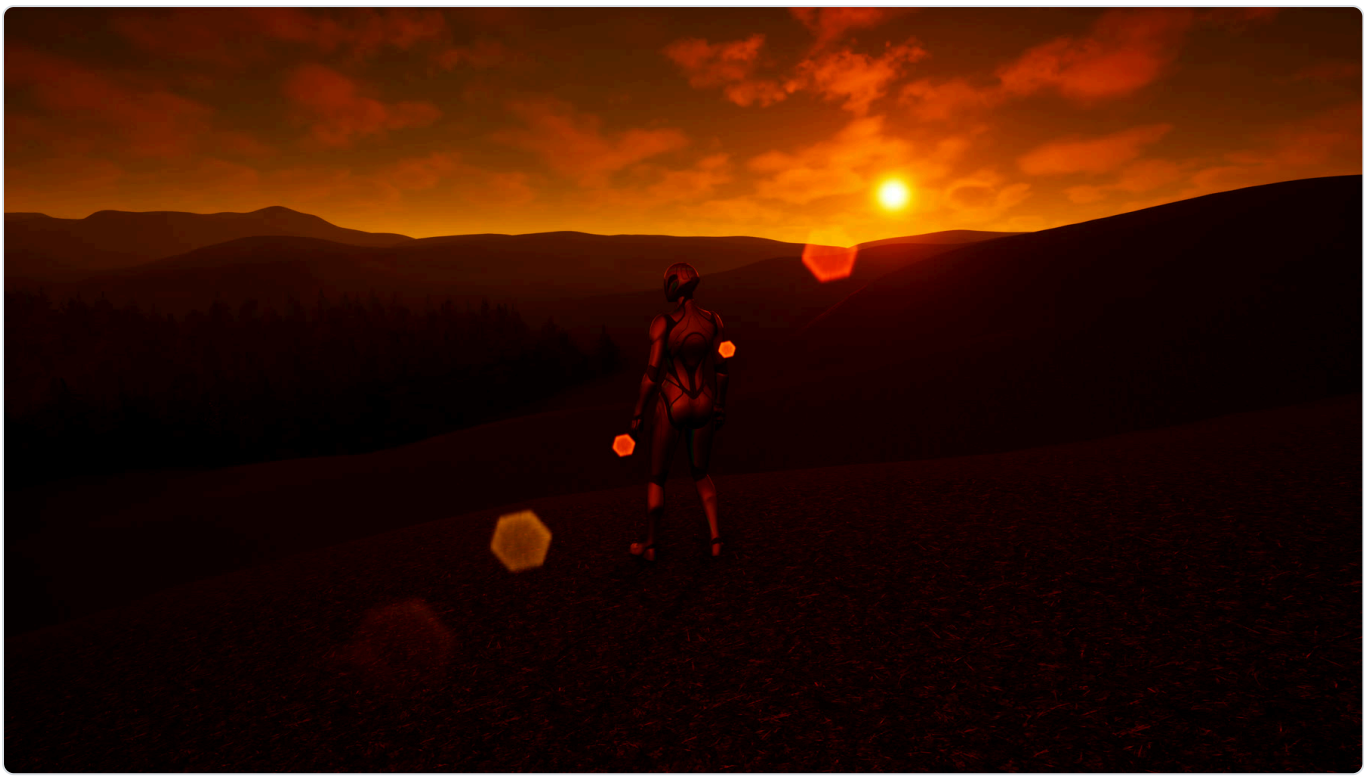
6. Working from a picture

This is the feature people underestimate. Attach a reference image, describe the outcome, and let the model match it — it can call real tools in the same turn it looks at your picture.



*"Set the atmosphere and lighting like in the picture I gave you." The reference photo rides along with the message. The reply — from a **local** model running in LM Studio, not a cloud one — reports exactly what it changed: the sun dropped to a −5° horizon angle at 2700 K and 5 lux, height fog tinted orange at density 0.06, a warm skylight at 2.5× for bounce, and a post-process pass with bloom 1.5, saturation 1.3, contrast 1.2 and −0.3 exposure compensation to keep the sun bright without blowing it out. Note the reasoning block above the answer, folded away until you want it.*

The viewport behind the panel is the result, live:



The same scene from the player's point of view after the change. Nothing here was hand-tuned — the values came from the model reading the reference photo.

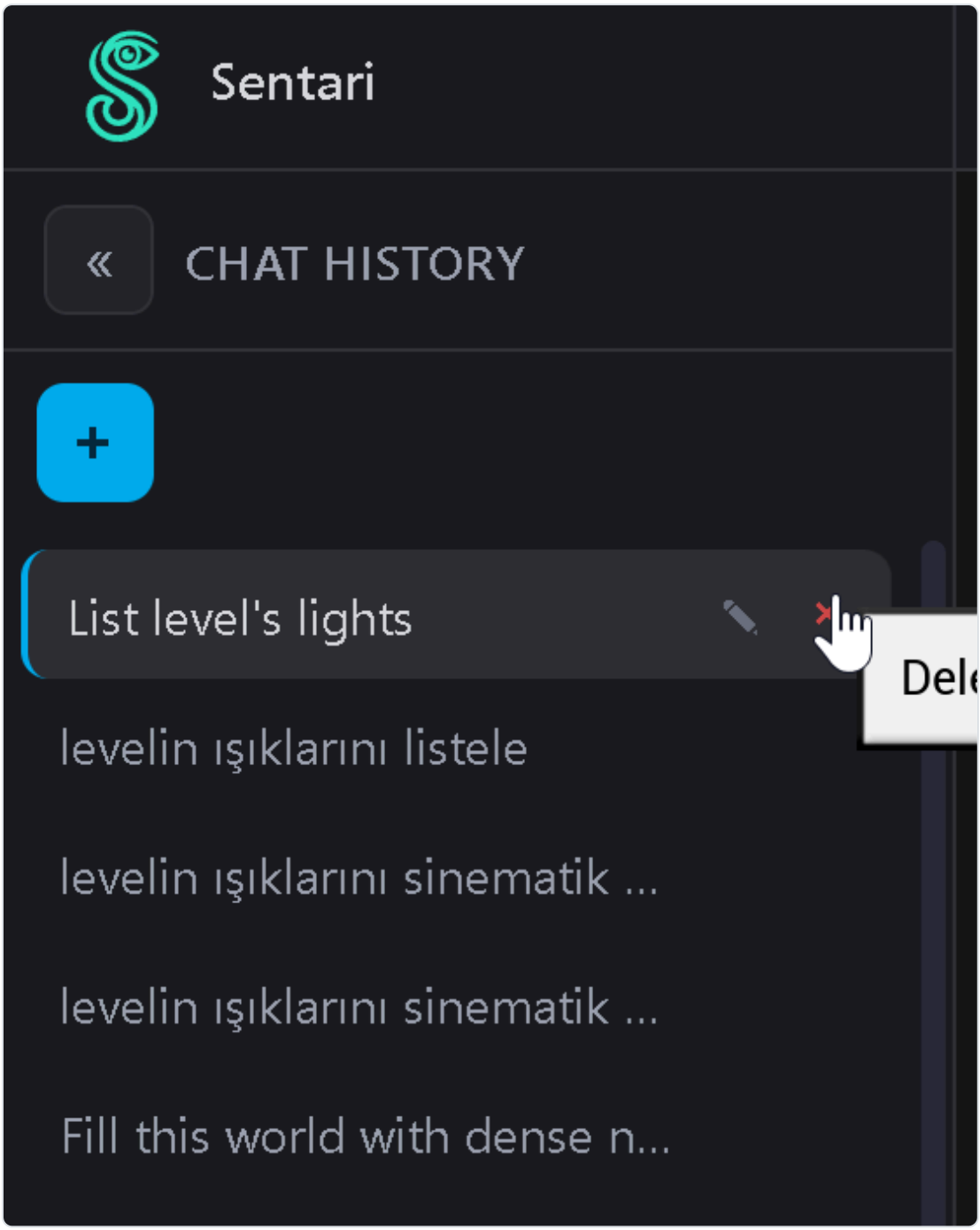
Two habits make this work well:

- **Describe the outcome, not the API.** "Low warm sun, orange haze, long shadows" beats "set DirectionalLight intensity to 5".
- **Let it iterate.** If it's close, reply with the correction — "a bit less fog", "push the sun another five degrees down" — instead of starting over.

---

7. Chat history

The left rail lists your **CHAT HISTORY**, each past chat titled by its first message, with a **+** to start a new one. History is kept across sessions, per project. To remove a conversation you no longer need, use the delete control on the entry:

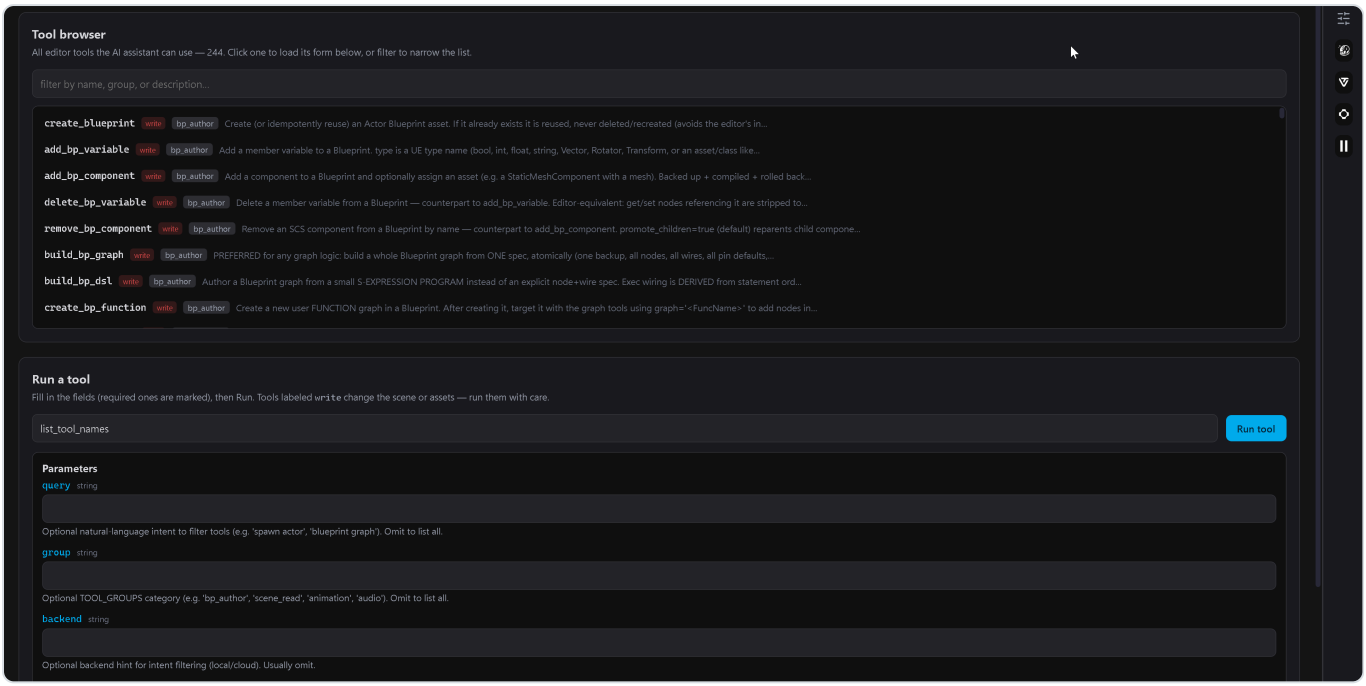


Starting a new chat is also a performance tool: conversation history is part of the context window, so a fresh chat for a fresh task leaves more room for tools and for the answer (see Section 19).

---

8. The Tools view

Switch the **Chat/Tools** toggle to **Tools** and the panel stops being a conversation and becomes a control surface. It has two halves: the **tool browser** on top, and **Run a tool** below it.



Every tool carries its group — `bp_author`, `scene_read`, `animation`, `audio` and so on — and, where it applies, a red `write` badge. Click one and its schema becomes the form underneath.

How the browser works

The list is not documentation; it is the **live roster**, read from the bridge at the moment you open it. It is precisely the set of tools offered to the model, with the same names, groups and descriptions — which is why the count here always matches the toolbar counter (244 built-in, plus Epic Tool Sources when enabled on UE 5.8). If a capability is missing from this list, the model cannot call it either. That alone makes the view worth opening when something doesn't work.

Filtering searches names, groups **and** descriptions, so "blueprint graph", "pcg" or "cooldown" each find their family. Clicking a tool copies its name into **Run a tool** and builds a form from that tool's JSON schema: one typed field per parameter, the required ones marked, and the schema's own help text underneath. You can also type a tool name directly if you already know it.

Run tool — what actually happens

This is the part worth understanding. Pressing **Run tool** does *not* go through the model. The panel packages your form values into a tool call and sends it straight down the MCP bridge on `localhost:8765` into the editor — the same code path a chat-driven call takes, minus the reasoning. Three practical consequences:

- **No model required.** No backend selected, no API key, nothing loaded in LM Studio — Run tool still works. The catalogue and the runner belong to the plugin, not to the LLM.
- **Fully offline, zero tokens.** Nothing leaves the machine, nothing is billed, and the round-trip is milliseconds rather than seconds.

- **The same gates as chat.** Names still resolve through reflection before acting, writes still pass the verify-gate and land on the undo stack, and the rules in *Options → Permissions* (Section 12) still apply. Running a tool by hand is not running it unguarded.

When Run tool returns a result — and when it can't

Not every tool is self-sufficient. The roster splits roughly four ways:

Tool family	By hand?	What it needs
Read & inspect `read_*` `list_*` `get_*`	Always	Nothing but a live editor. They answer immediately and change nothing — the safest place to start.
Write & author `spawn_actor` `build_bp_graph`	Yes	Every required field filled, valid asset paths, and the matching permission enabled. These carry the `write` badge and really do change your project — verified and undoable, but real.
Runtime `pie_*`	In play only	A running Play-In-Editor session. Outside PIE there is no live world to address, so the call returns an error instead of a result.
Async & generative `analyze_viewport_async` `generate_3d_async`	In two steps	These hand back a *job id*, not an answer; you collect the result with `get_vision_job` or from the provider's panel in the right dock. Vision needs a vision-capable model; the generative ones need a provider key and network access, so they are the one family that will **not** work offline.

Two things stop a run before it starts: a **missing required field** — the call is rejected by schema validation rather than half-executed — and a **disconnected bridge**, which the status dot and tool counter tell you at a glance. Epic Tool Sources are UE 5.8 only, so on 5.7 they are simply absent from the list. And a few tools take a whole structured spec as a single argument, `build_bp_graph` being the obvious case; hand-authoring that JSON is possible, but it is exactly the work you have a model for.

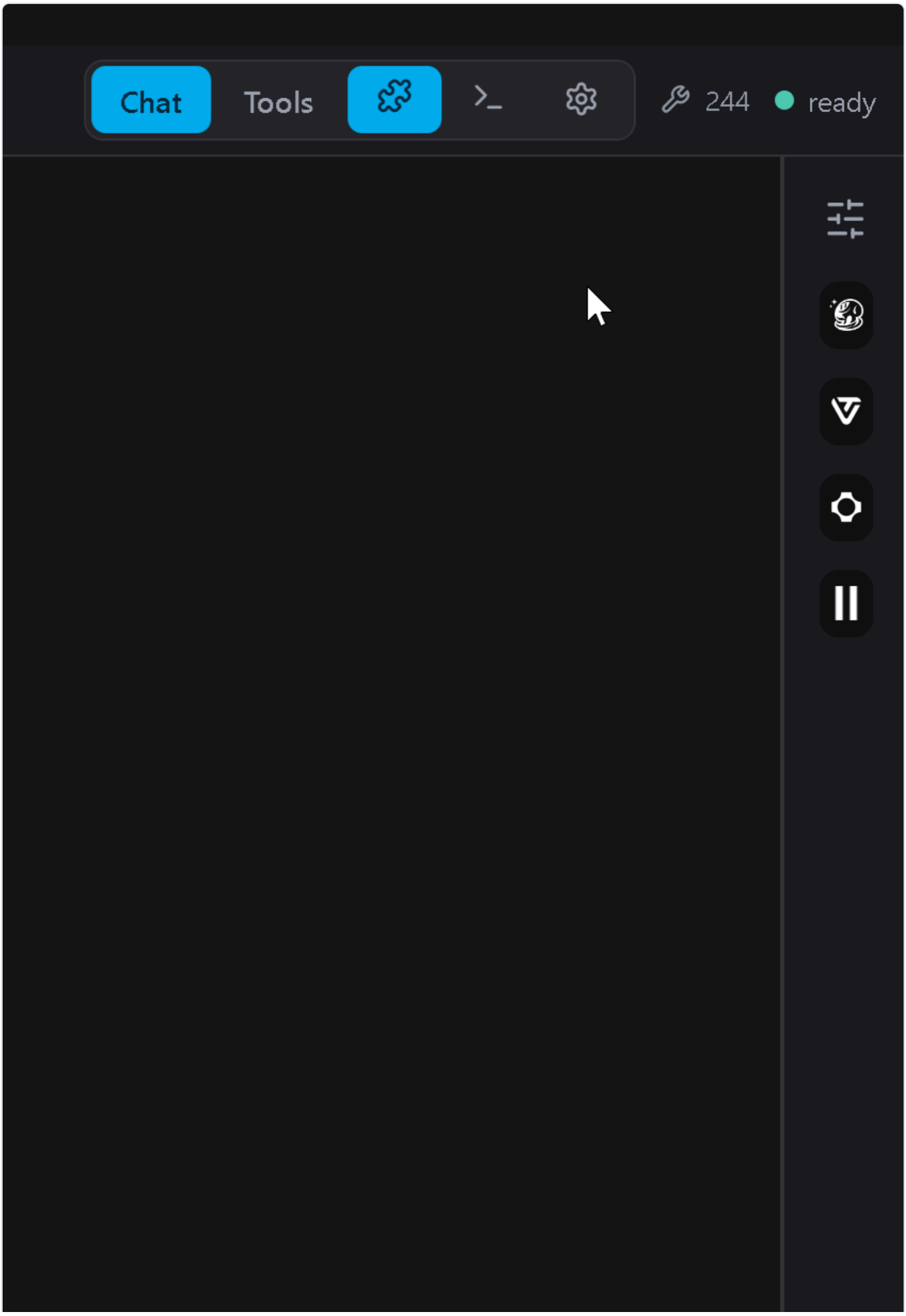
Why use it at all

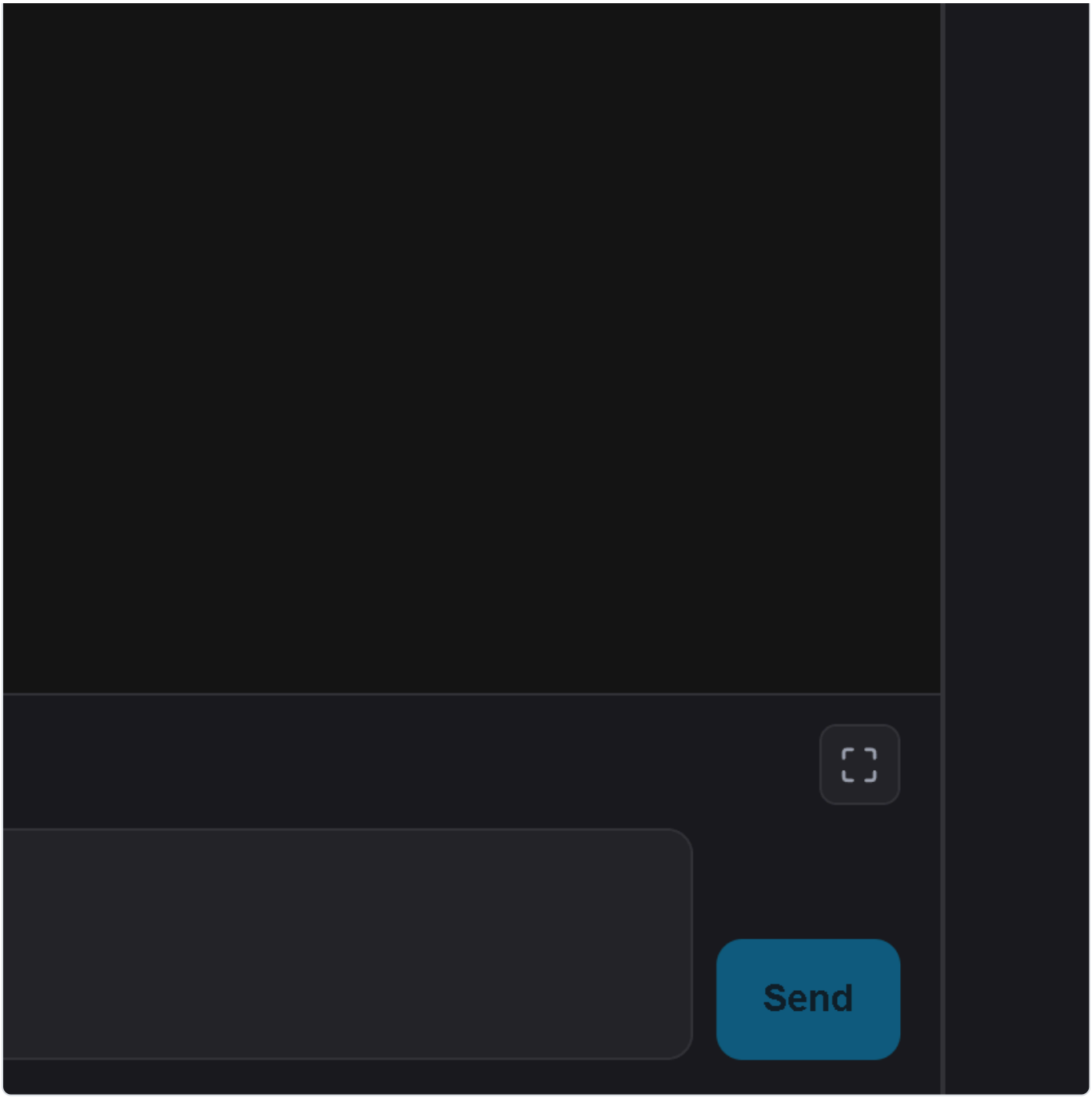
Three reasons, in the order they tend to matter:

1. **It is a catalogue.** The honest answer to "what can this thing actually do?", at the level of individual tool names and descriptions.
2. **It is a diagnostic.** If a tool succeeds here but the model won't call it, the problem is the model or the prompt — not the plugin. That single test resolves most "it isn't working" cases in under a minute.
3. **It is a fallback.** A small local model that fumbles a tool call can be bypassed entirely for the one action you actually needed.

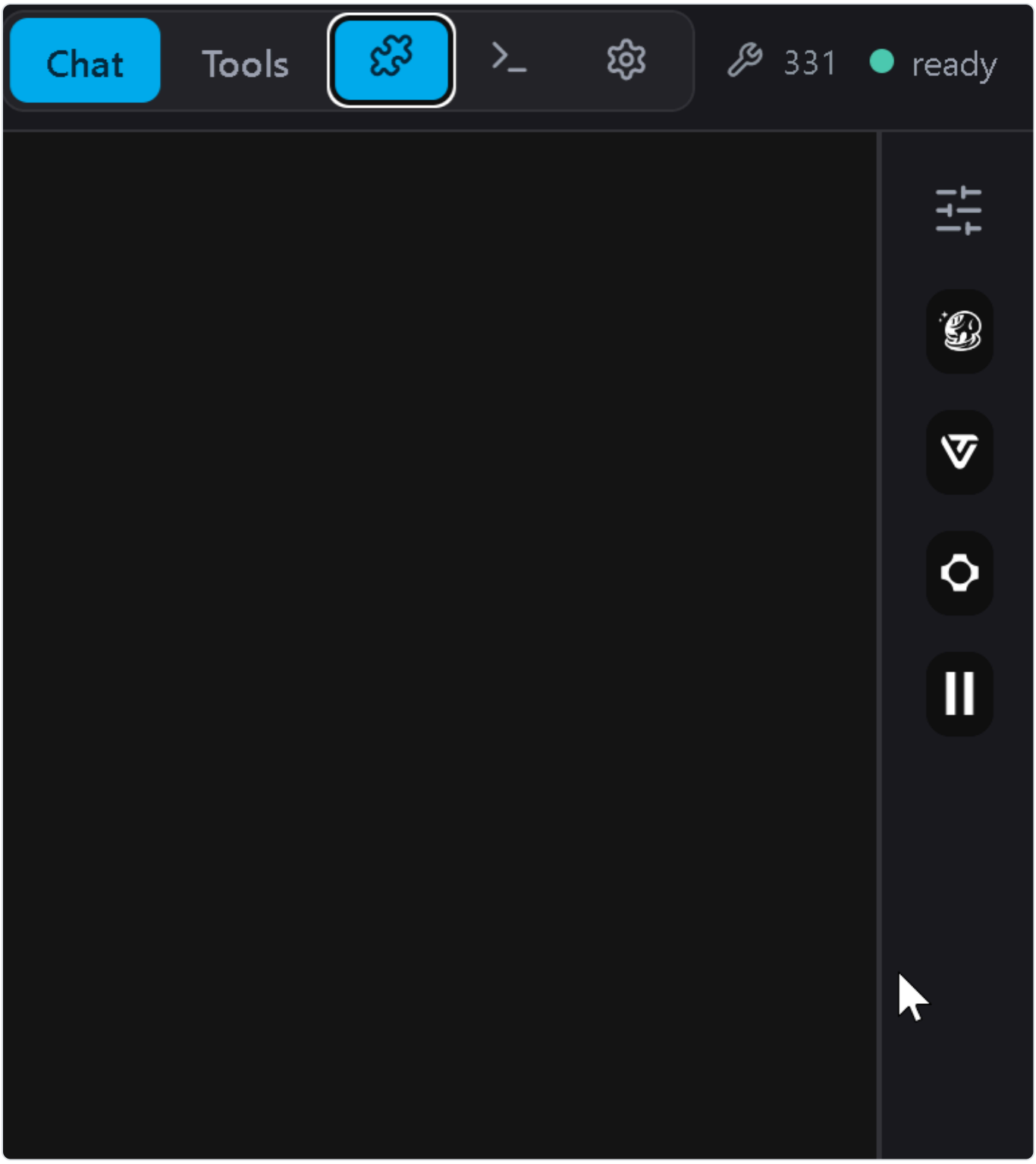
9. The right dock (provider tools)

The icons on the right open per-provider panels. When collapsed they take almost no space:


Chat
Tools
244
ready

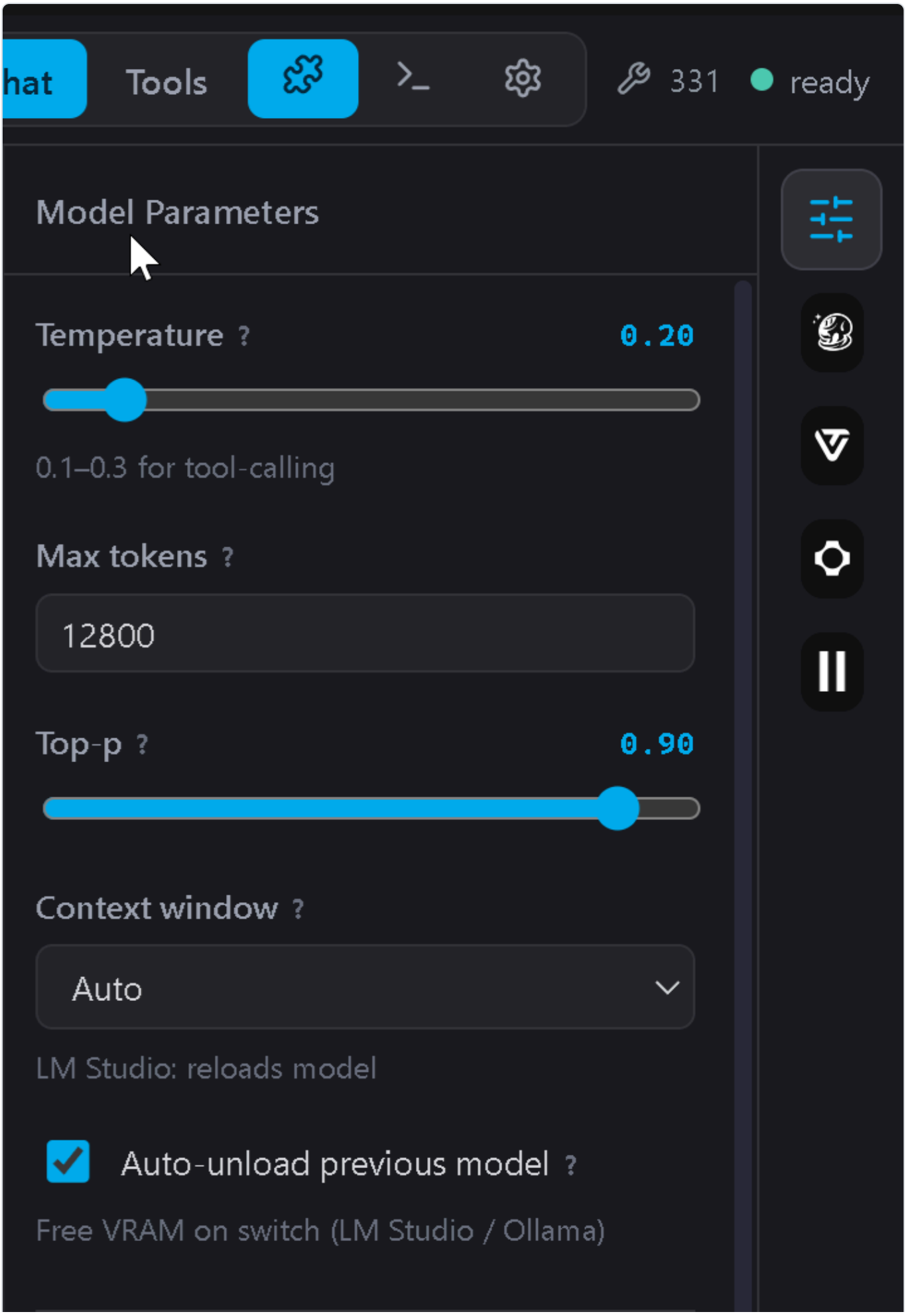


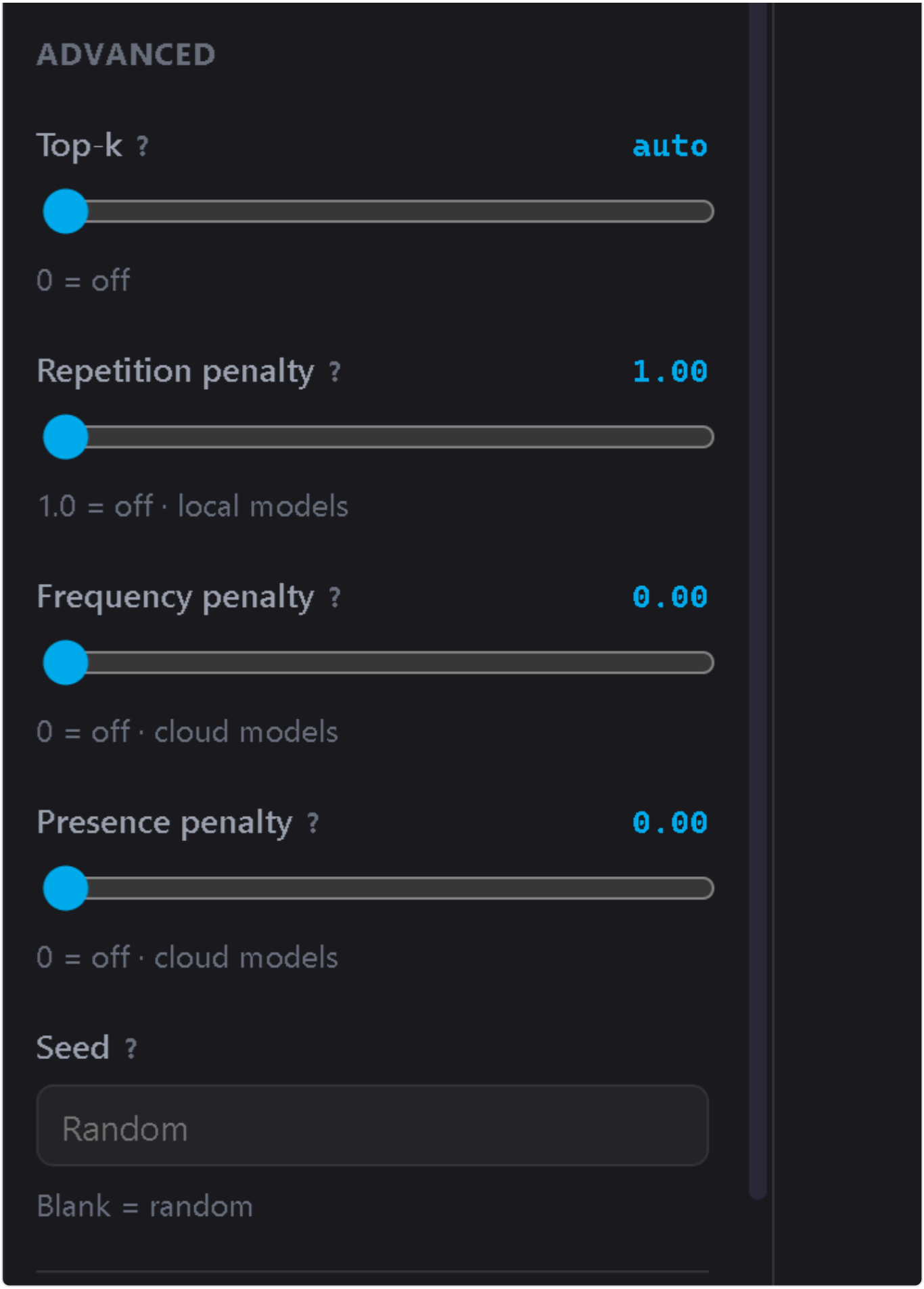
Expand a panel to work with that provider. Each one is independent, so you can keep the ones you use open and the rest collapsed:



Model Parameters

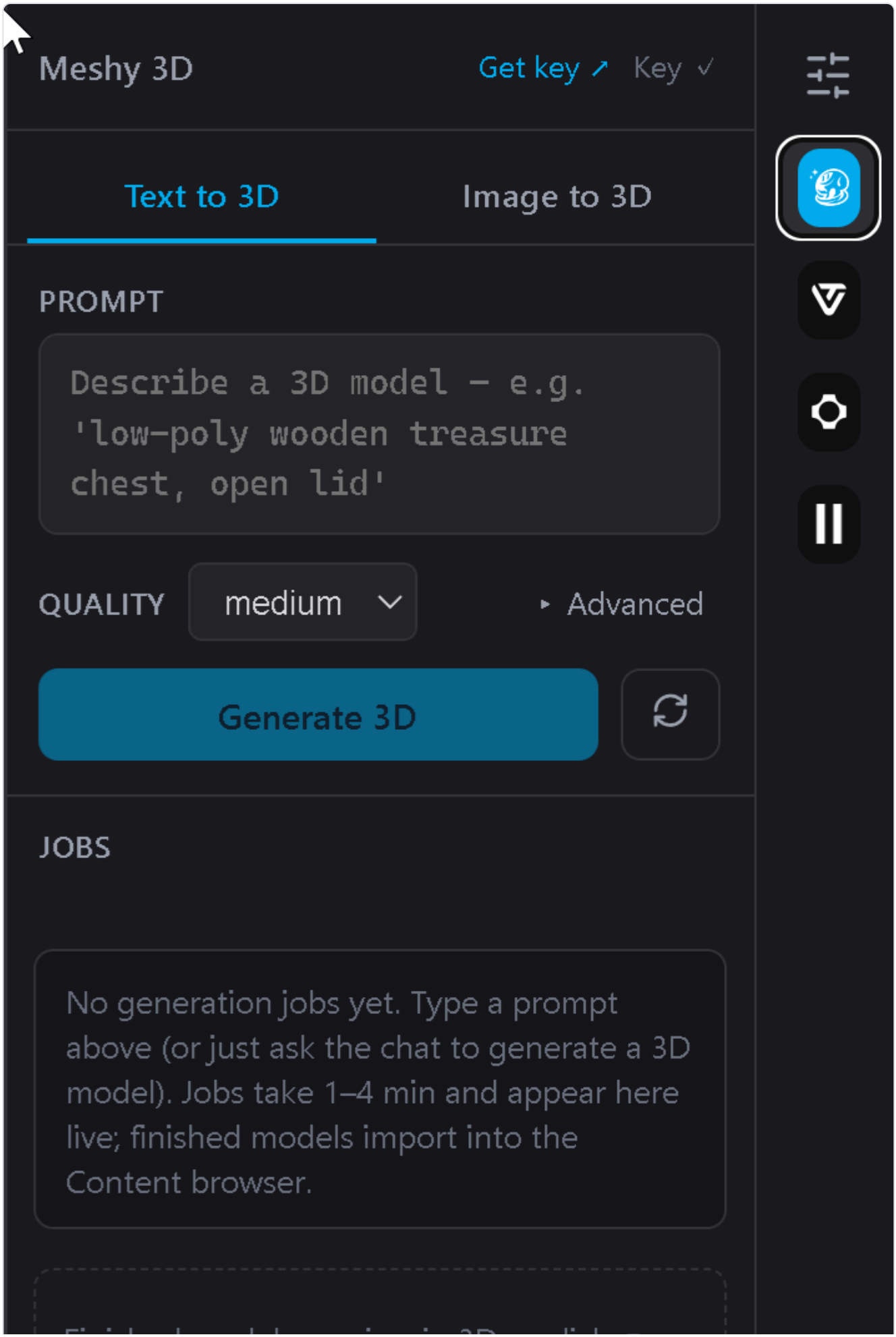
Tune sampling for the selected model — temperature, top-p, max tokens (**MaxTok**) and the other sampling knobs. Changes apply live to the next run. MaxTok is the setting Section 19 spends the most time on.

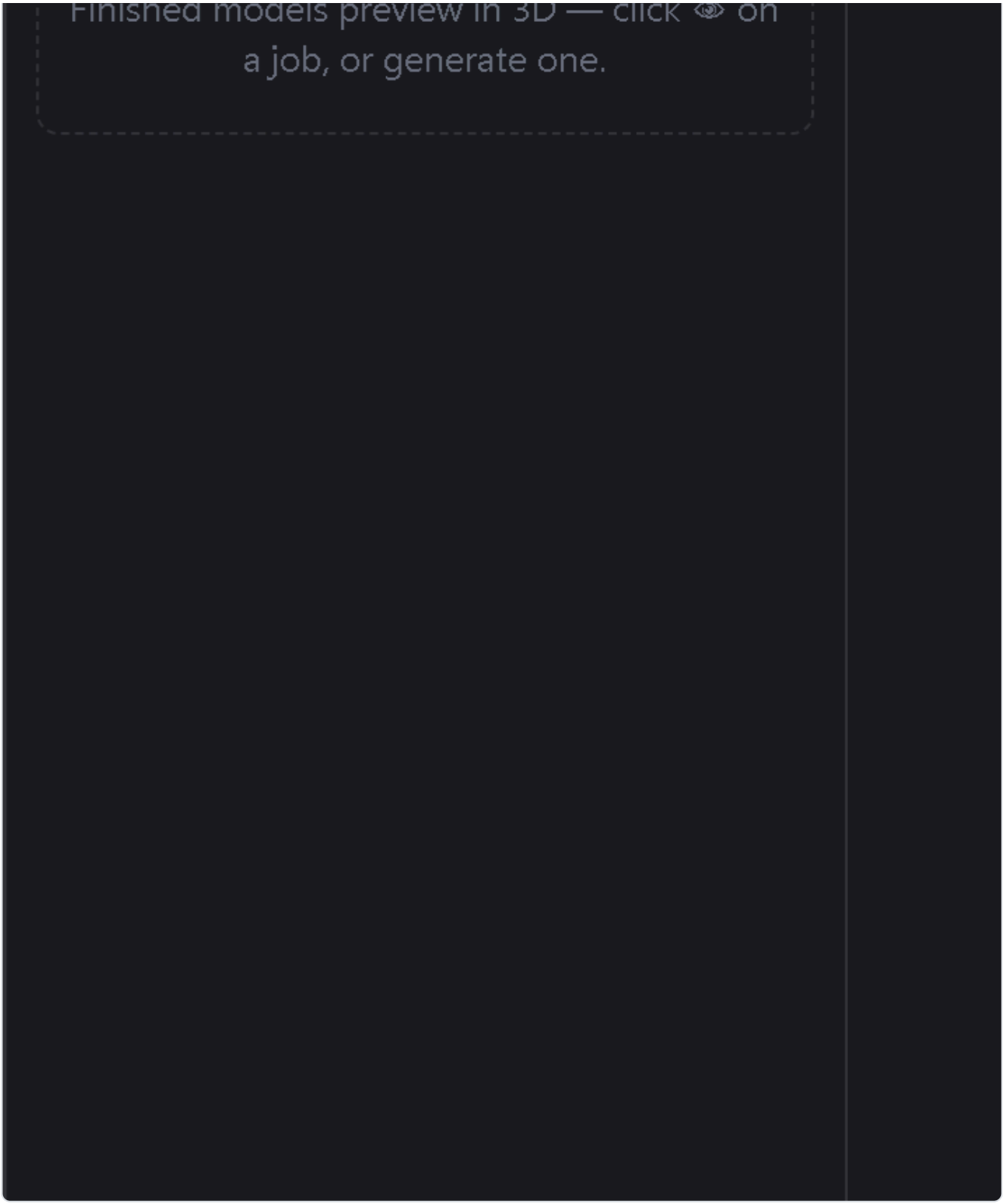

hat
Tools
331
ready
Model Parameters
Temperature ?
0.20
0.1–0.3 for tool-calling
Max tokens ?
12800
Top-p ?
0.90
Context window ?
Auto
LM Studio: reloads model
Auto-unload previous model ?
Free VRAM on switch (LM Studio / Ollama)



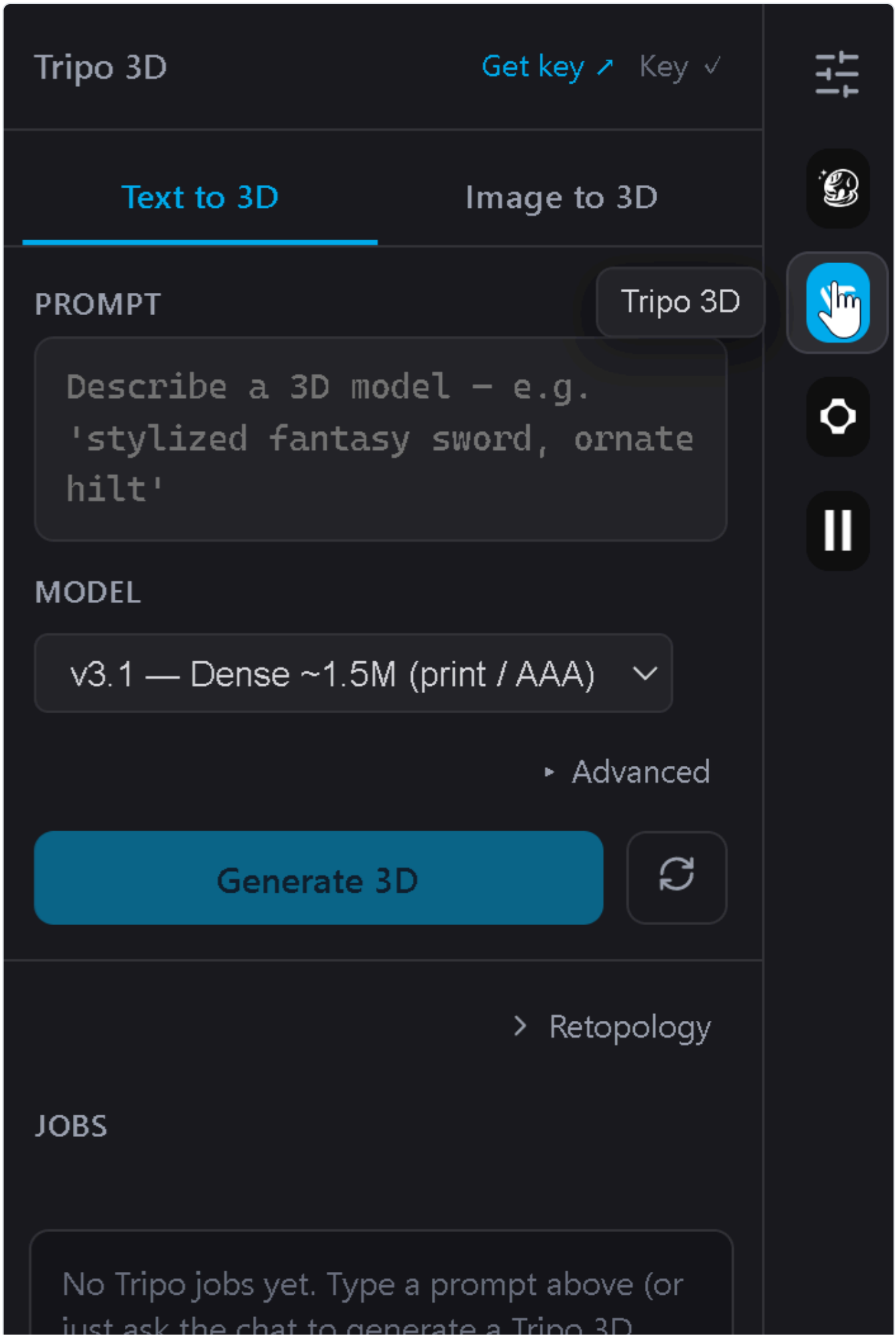
Generative providers

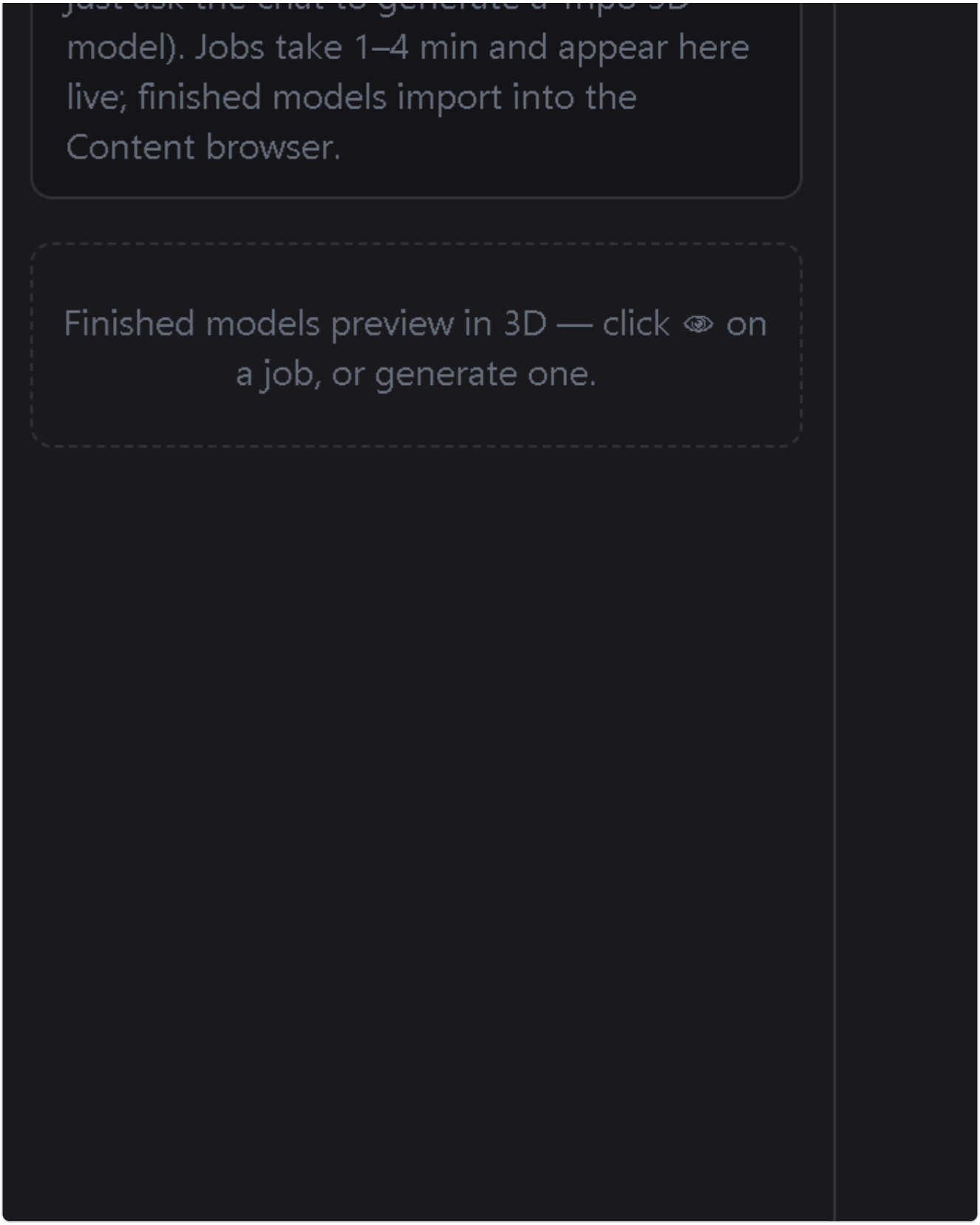
- **Meshy 3D** — text-to-3D and image-to-3D. Paste your provider key, then generate models that preview in the inline viewer before import.


Meshy 3D
Get key ↗
Key ✓
Text to 3D
Image to 3D
PROMPT
Describe a 3D model – e.g. 'low-poly wooden treasure chest, open lid'
QUALITY
medium
Advanced
Generate 3D
JOBS
No generation jobs yet. Type a prompt above (or just ask the chat to generate a 3D model). Jobs take 1–4 min and appear here live; finished models import into the Content browser.

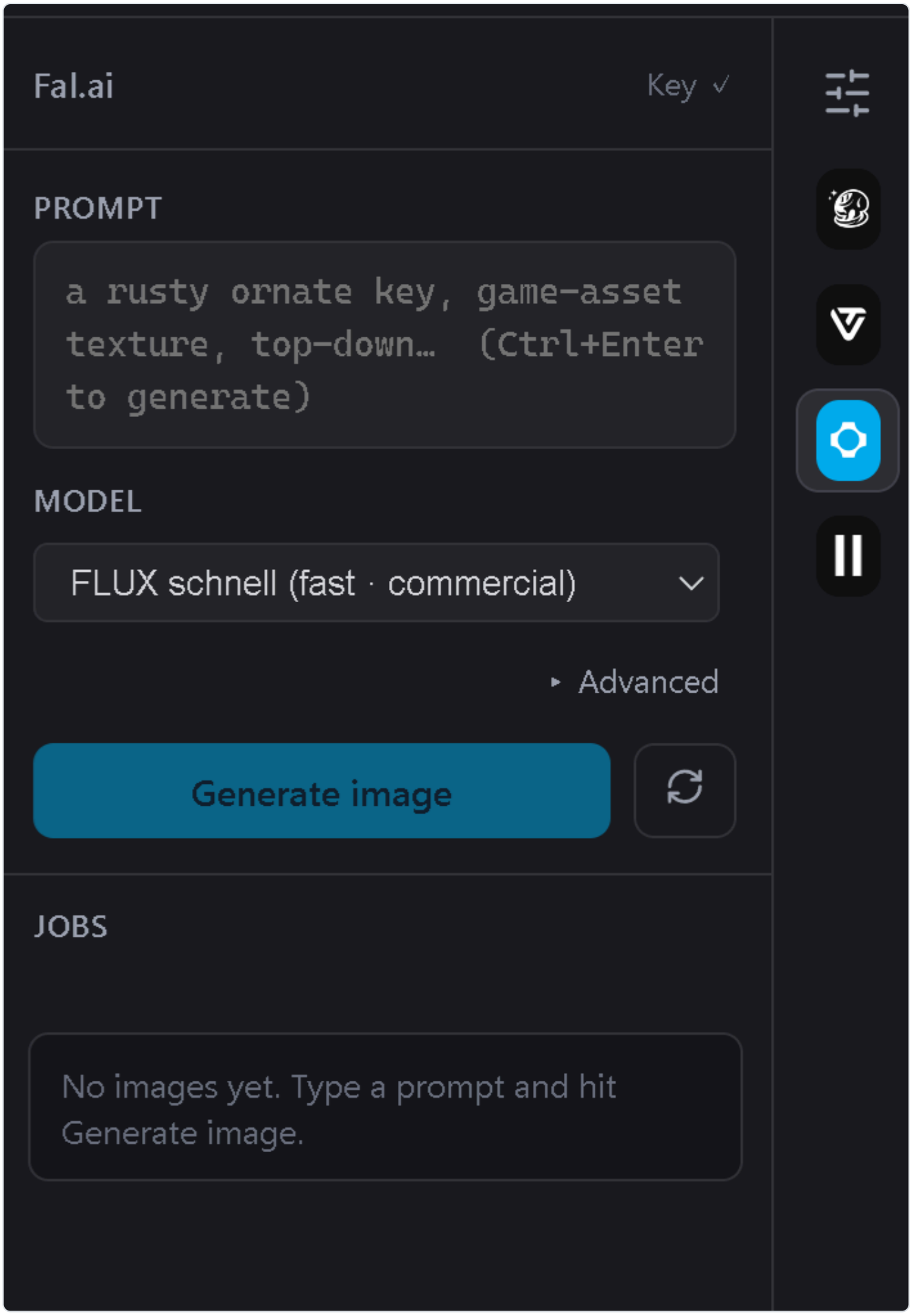


- **Tripo 3D** — an alternative text/image-to-3D backend, with its own quality and topology options.

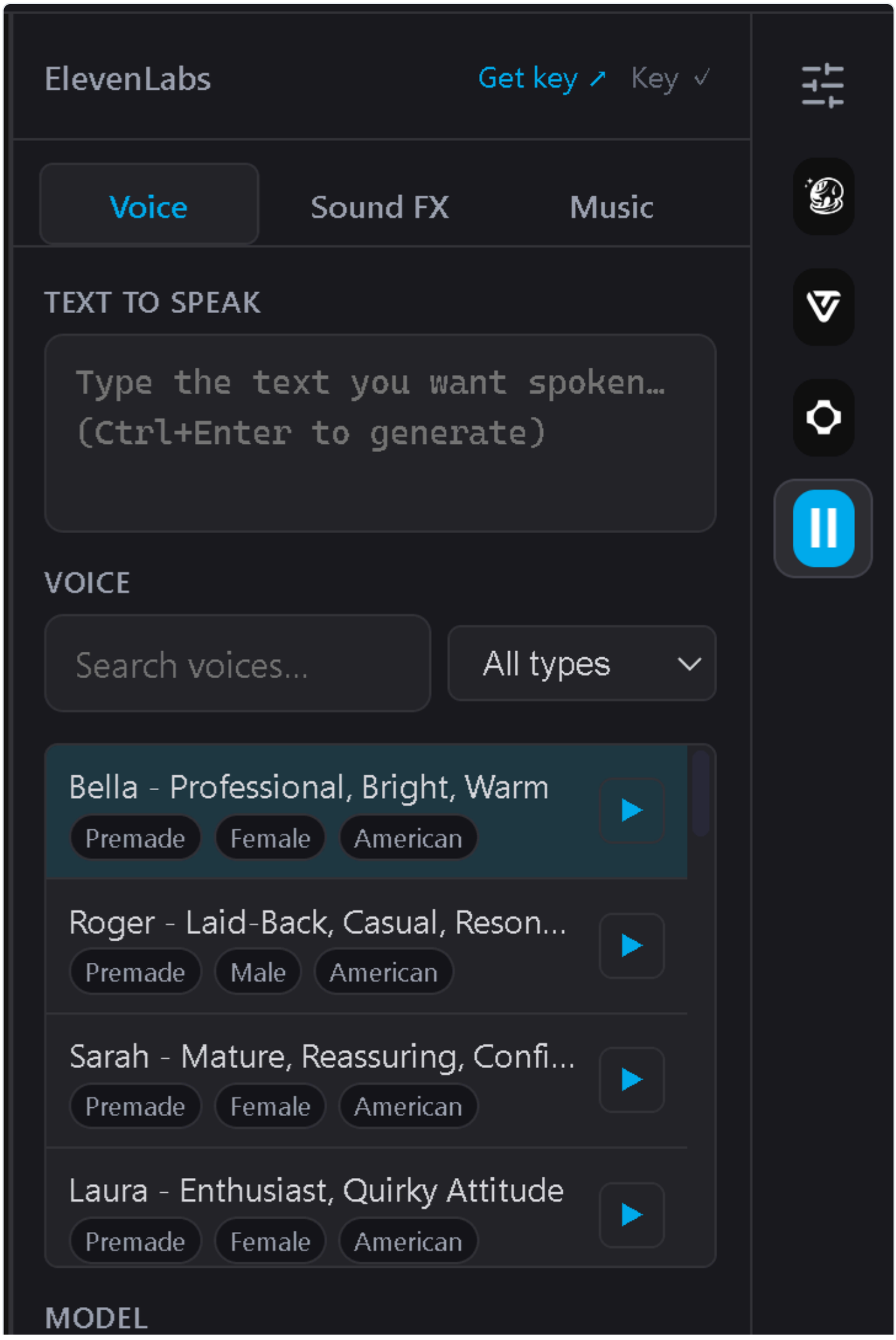

Tripo 3D
Get key ↗
Key ✓
Text to 3D
Image to 3D
PROMPT
Tripo 3D
Describe a 3D model – e.g. 'stylized fantasy sword, ornate hilt'
MODEL
v3.1 — Dense ~1.5M (print / AAA)
Advanced
Generate 3D
Retopology
JOBS
No Tripo jobs yet. Type a prompt above (or

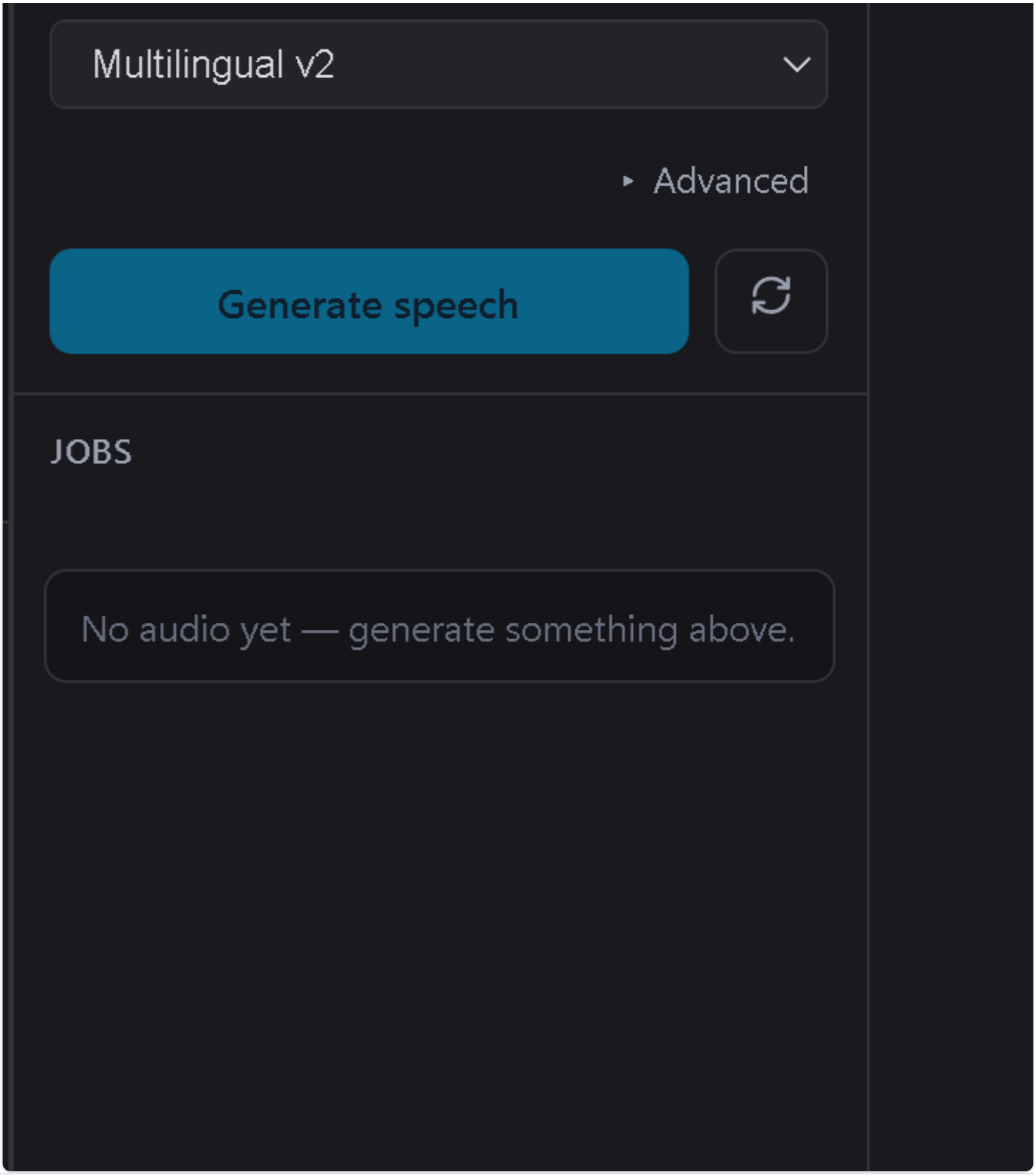


- **fal.ai** — image generation (FLUX and more).


Fal.ai
Key ✓
PROMPT
a rusty ornate key, game-asset texture, top-down… (Ctrl+Enter to generate)
MODEL
FLUX schnell (fast · commercial)
▸ Advanced
Generate image
JOBS
No images yet. Type a prompt and hit Generate image.

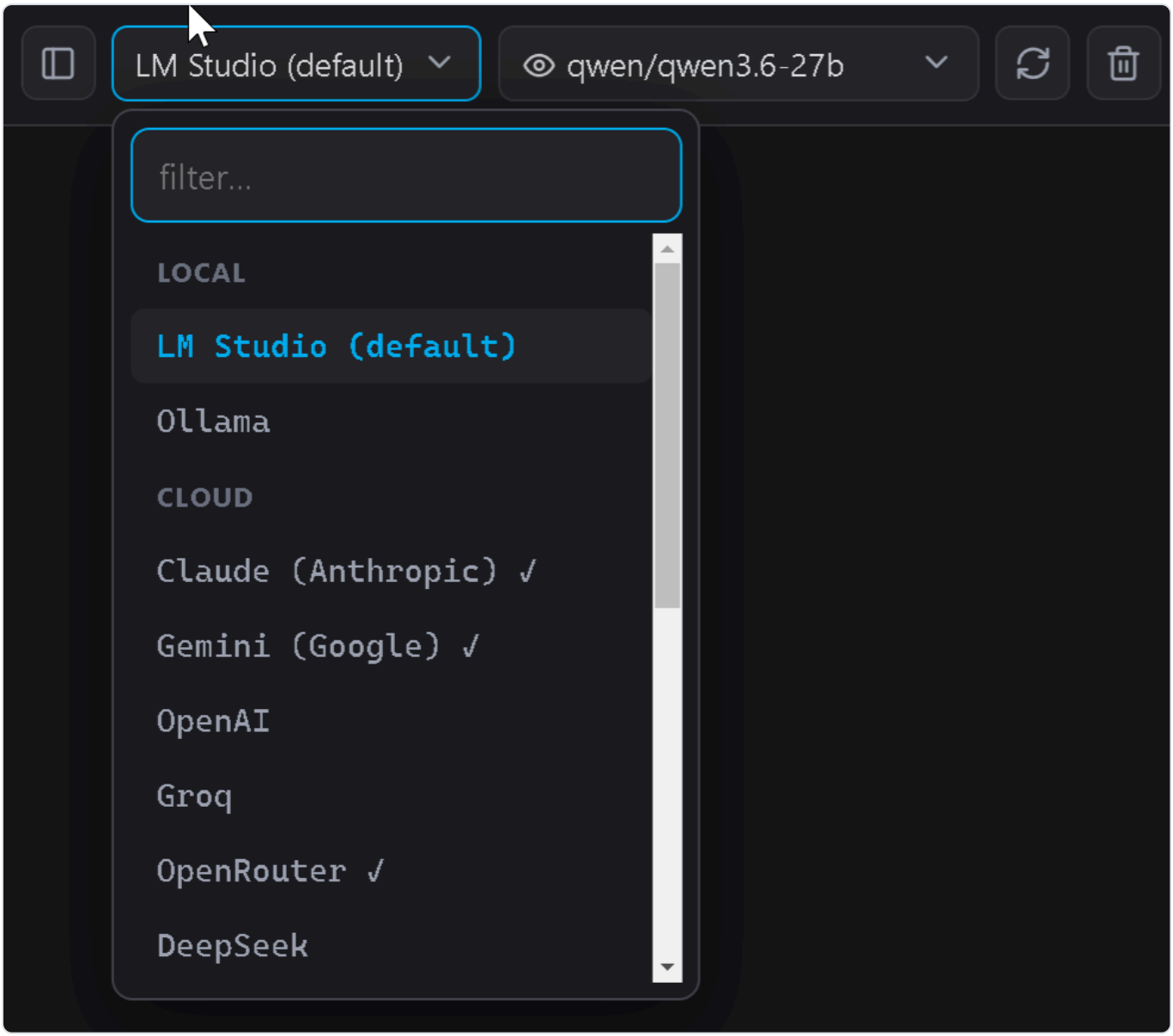
- **ElevenLabs** — voice (TTS), sound effects, and music generation, each on its own sub-tab.


ElevenLabs
Get key ↗
Key ✓
Voice
Sound FX
Music
TEXT TO SPEAK
Type the text you want spoken…
(Ctrl+Enter to generate)
VOICE
Search voices...
All types
Bella - Professional, Bright, Warm
Premade
Female
American
Roger - Laid-Back, Casual, Reson...
Premade
Male
American
Sarah - Mature, Reassuring, Confi...
Premade
Female
American
Laura - Enthusiast, Quirky Attitude
Premade
Female
American
MODEL



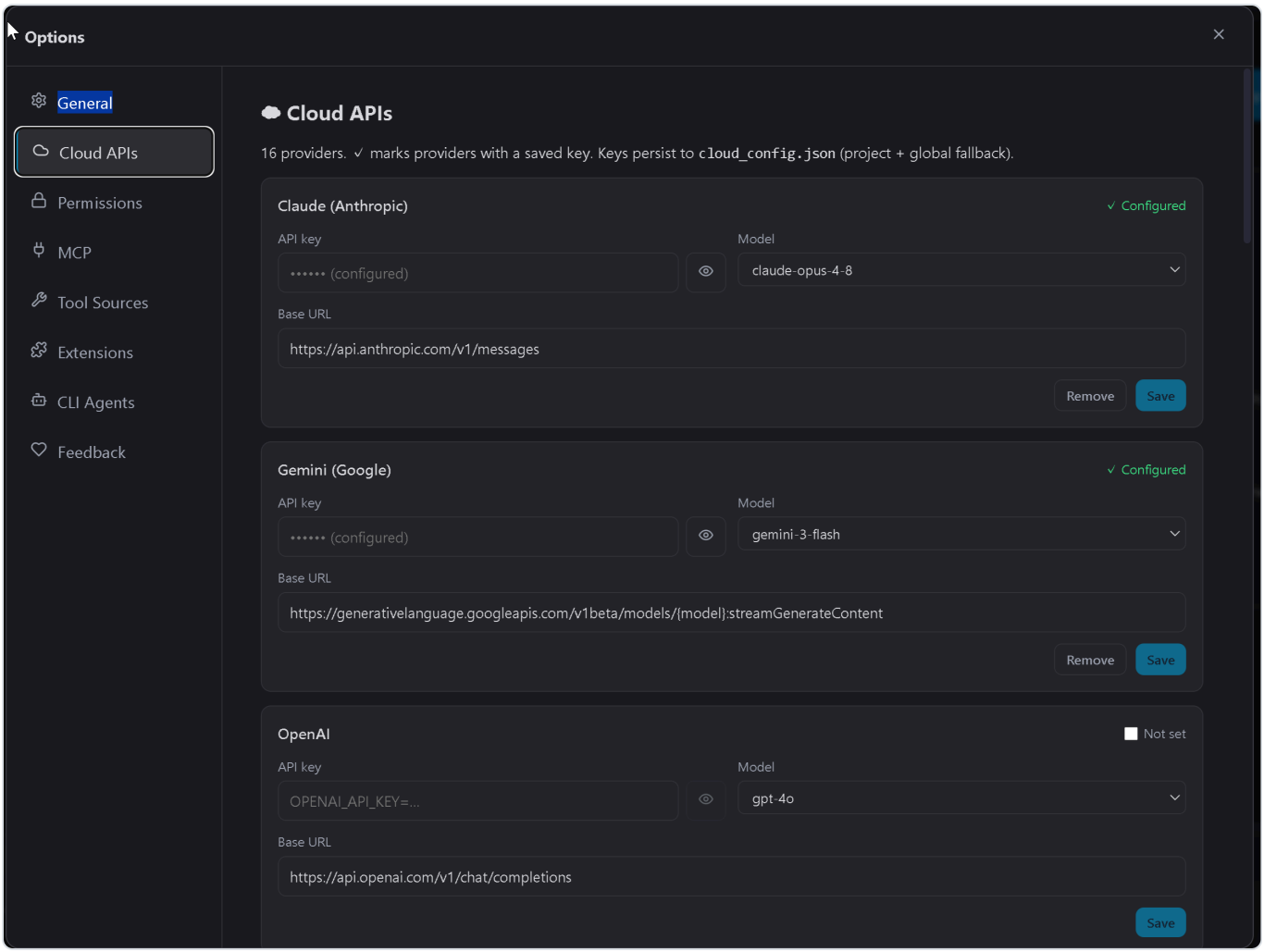
10. Options — General

Click the gear icon to open the **Options** modal. The **General** tab is where you pick the backend and model, choose the interface language, and toggle **Offline mode** to guarantee Sentari never reaches the network (useful on secured machines or planes).



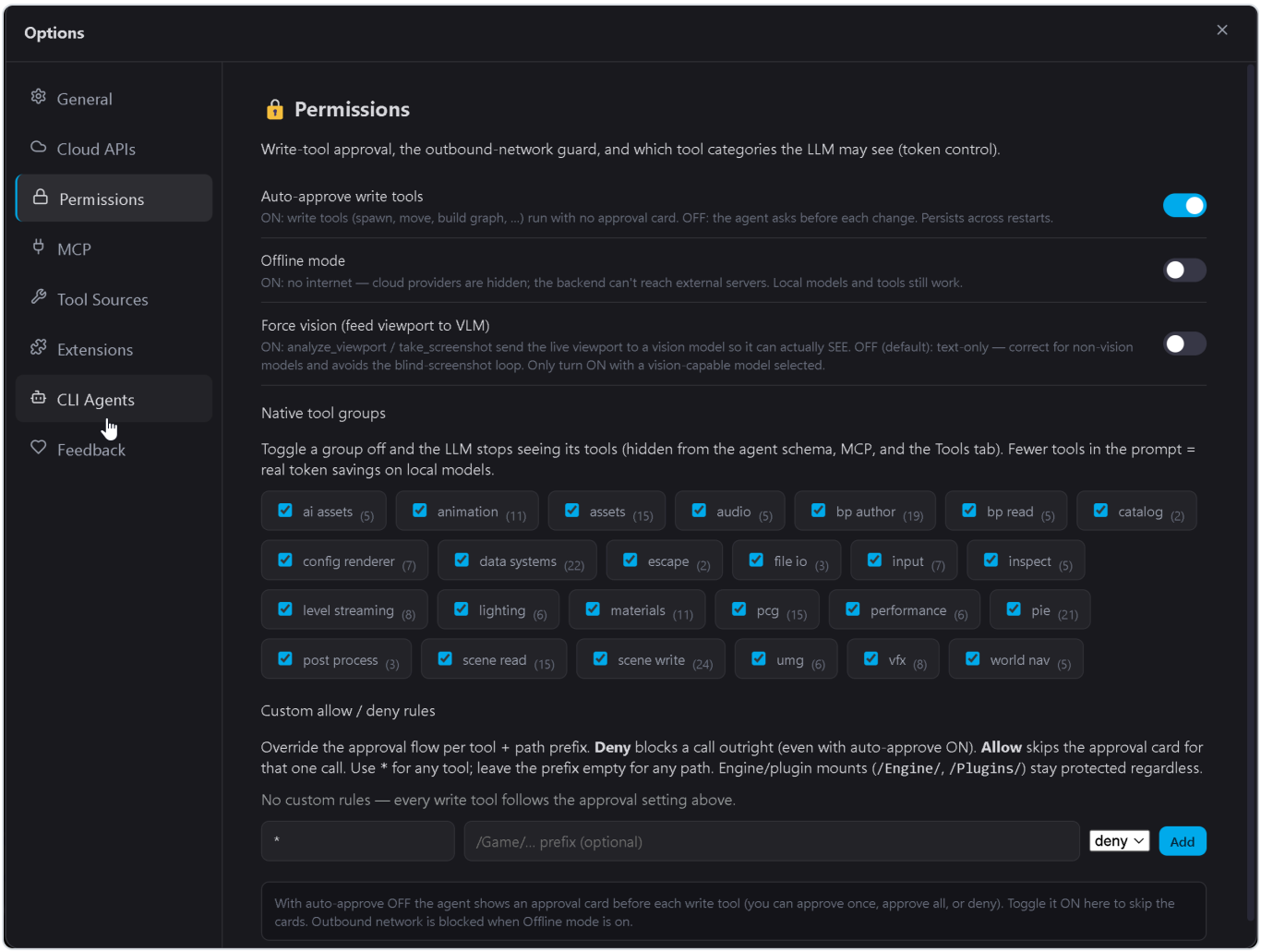
11. Options — Cloud APIs

One card per cloud provider. Expand a provider, paste your API key, and **Save**. Keys are stored locally; a ✓ appears beside the provider in the backend dropdown once saved. Prefer not to save a key to disk? Set the matching environment variable instead.



12. Options — Permissions

Fine-grained control over what Sentari may touch:



- **Auto-approve** — let tasks run without asking each time.
- **Permission rules** — allow or block specific tool families and asset folders.

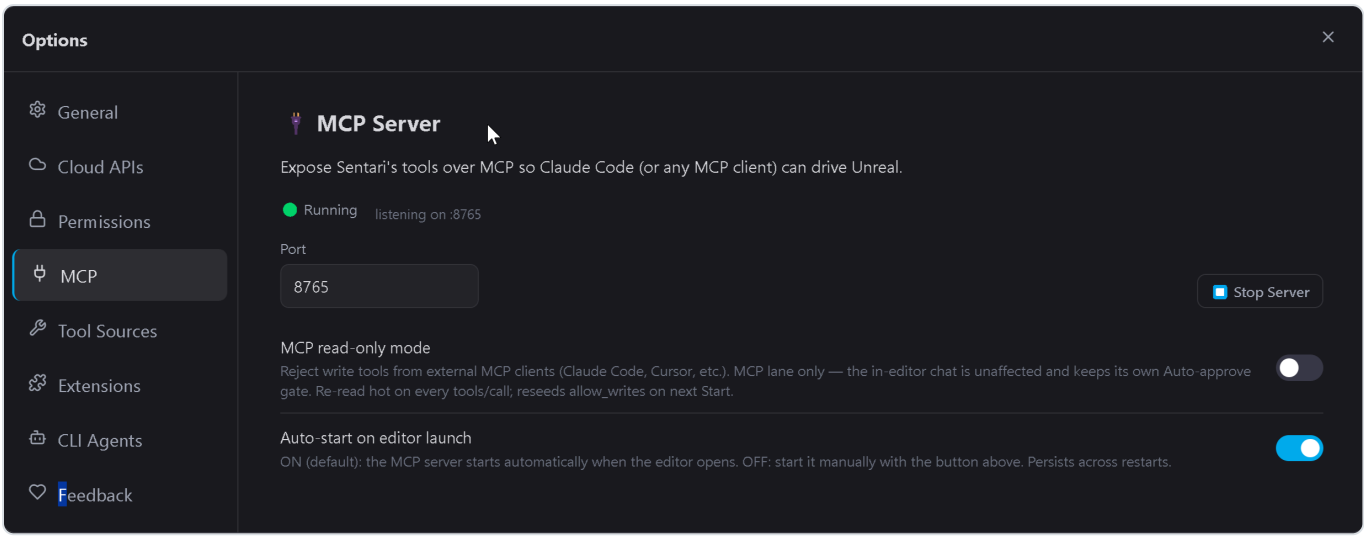
These are two independent controls, and the distinction matters. Auto-approve only removes the per-task confirmation prompt; permission rules allow or block a tool family or an asset folder *regardless* of whether anything was approved. The combination most people settle on is **auto-approve on, with the folders you care about ruled out** — freedom where you trust it, hard boundaries where you don't.

Rules apply to manual runs from the Tools view (Section 8) as well, so scoping a family off genuinely removes it as a capability rather than merely discouraging the model from choosing it.

---

13. Options — MCP

Start/stop the local **Model Context Protocol** server on `localhost:8765`. Any MCP-capable client (Claude Desktop, Cursor, Windsurf) can then call the same tools Sentari uses — useful for driving the editor from an external workflow.

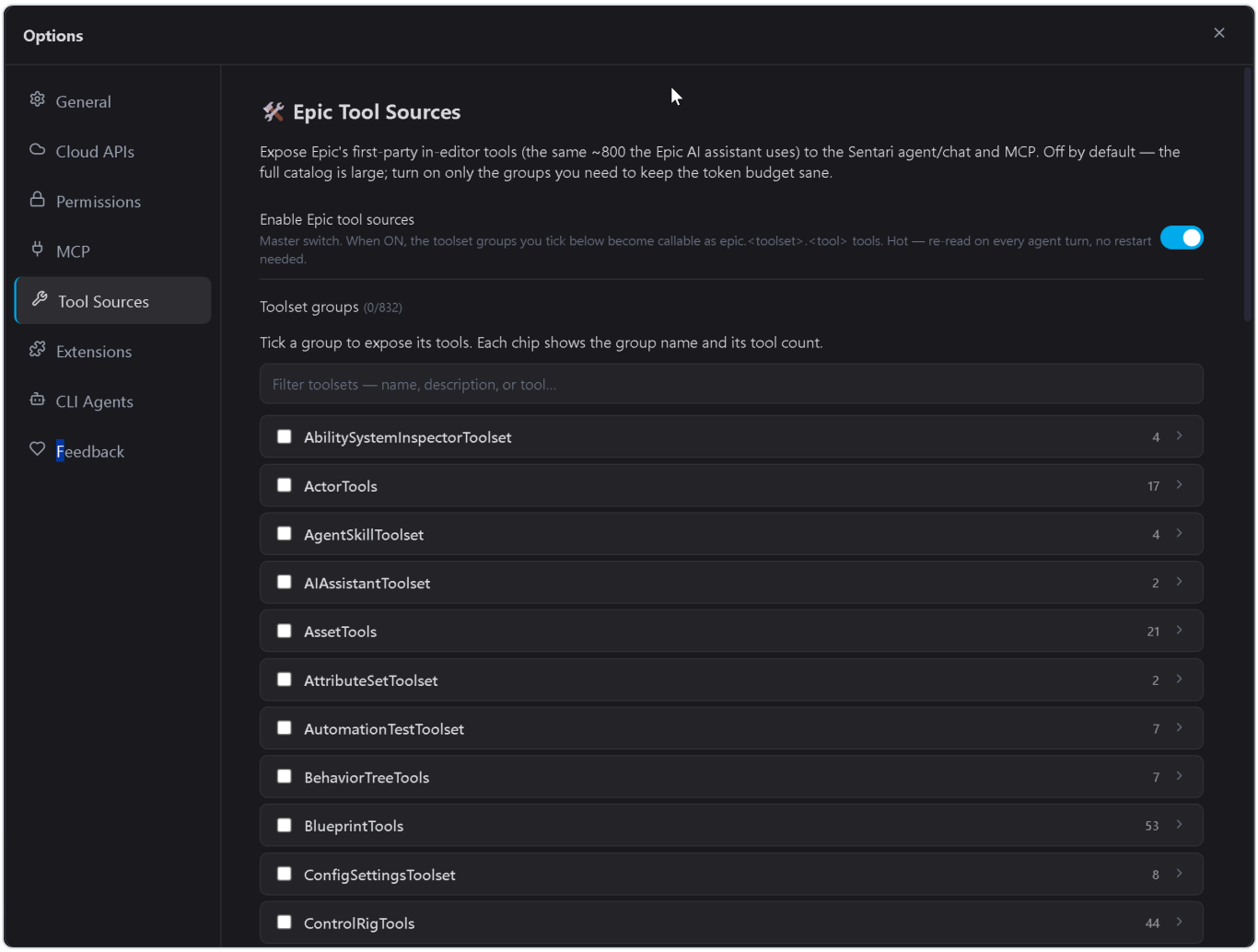


Read-only mode scopes external MCP clients so they can inspect the project but not mutate it. It applies to the MCP surface, not to the panel's own chat.

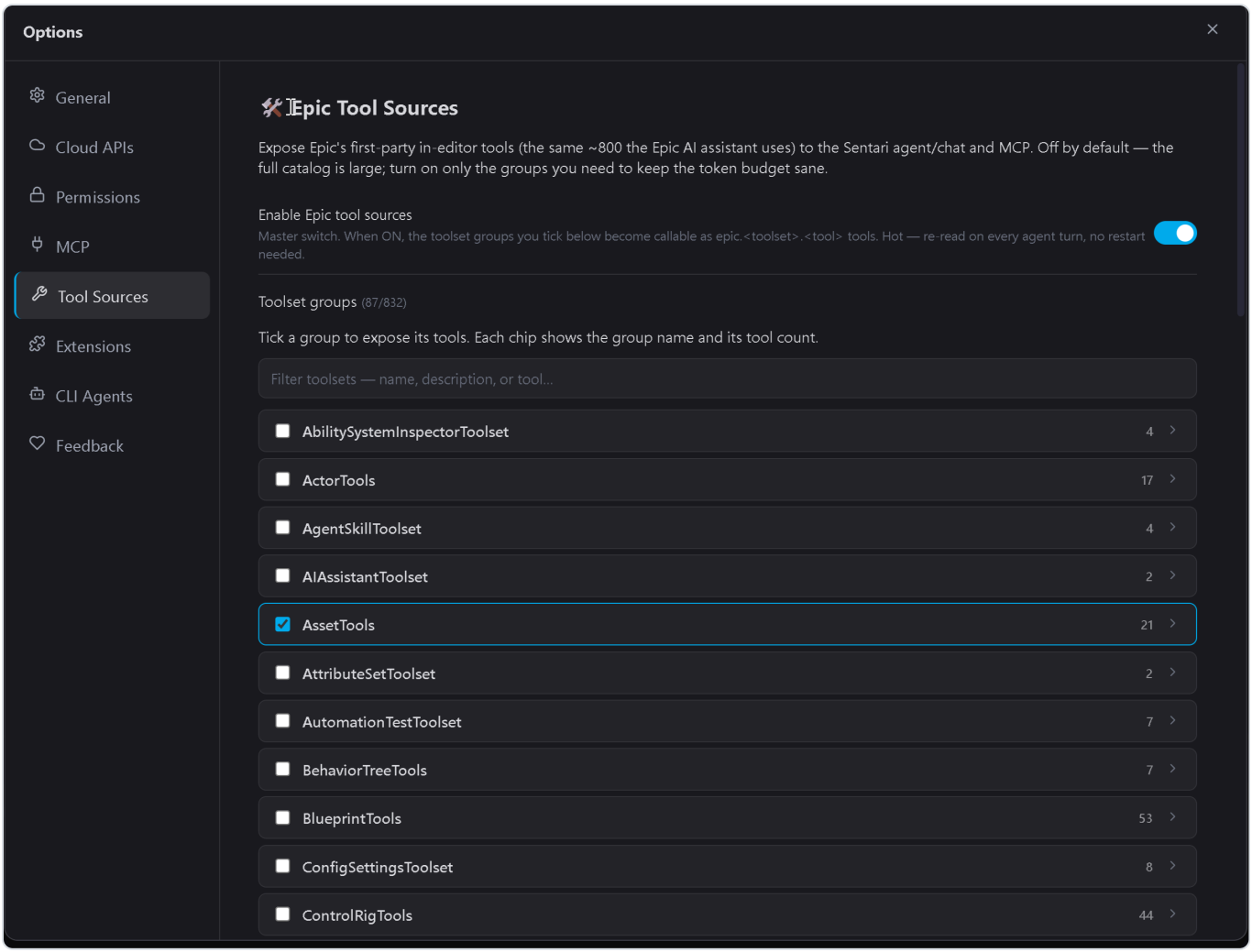
---

14. Options — Epic Tool Sources

Toggle the hundreds of built-in Epic editor tools on or off. When enabled, they appear in the tool catalogue and the model can call them; when off, they're hidden from the model entirely. This is what moves the toolbar counter well past 244.



Use the **Tool Sources** view to enable or disable entire groups at once, scoping the model down to exactly the capabilities you trust:

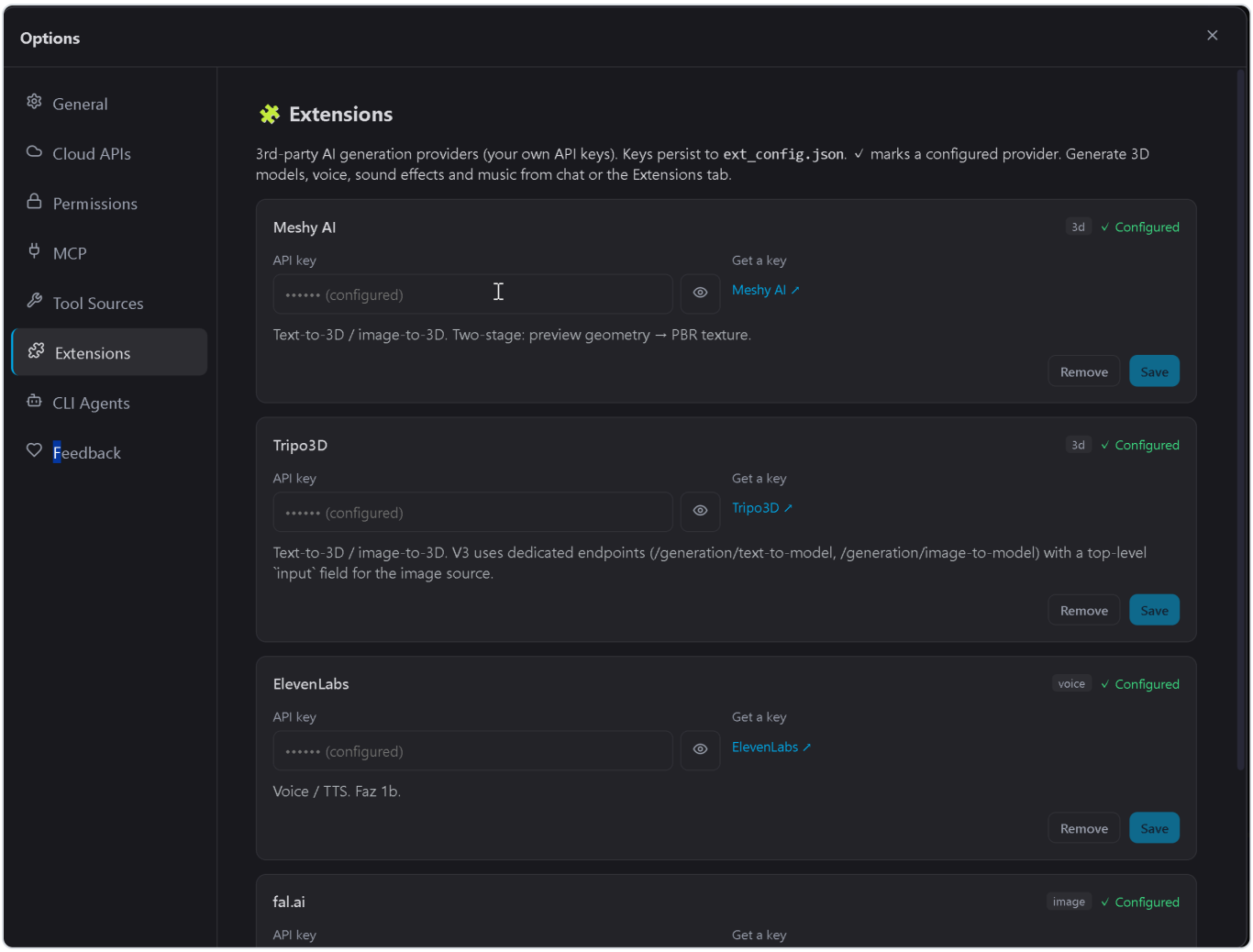


> Epic Tool Sources require Unreal Engine 5.8. On earlier versions the tab hides itself and the core toolset is unaffected.

---

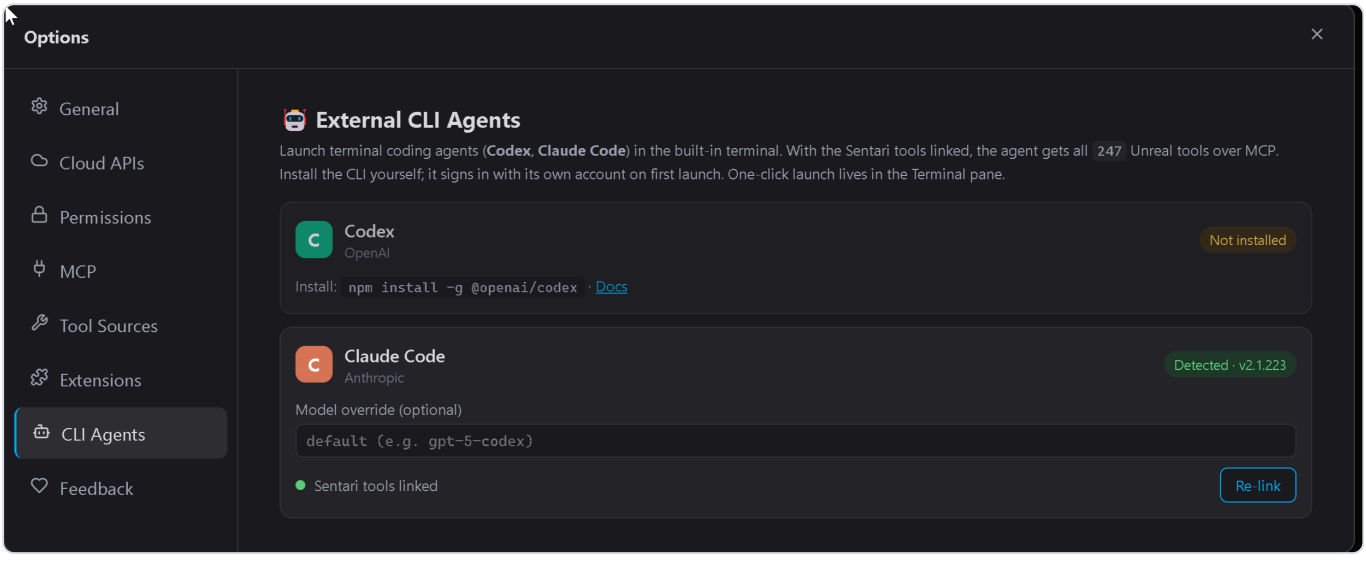
15. Options — Extensions

Settings for the generative providers in the right dock — paste your Meshy, Tripo, fal.ai, and ElevenLabs keys here.

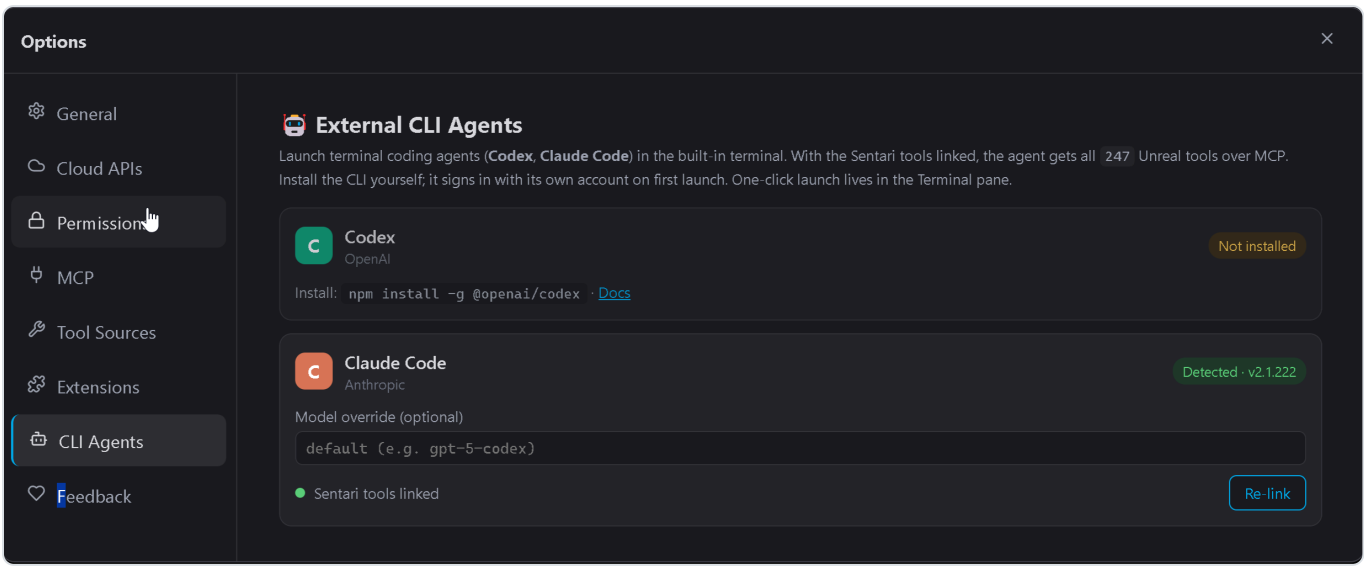


16. Options — CLI Agents

Launch **Codex** or **Claude Code** from the panel, pre-configured to work alongside Sentari. A status chip shows whether each agent is running.

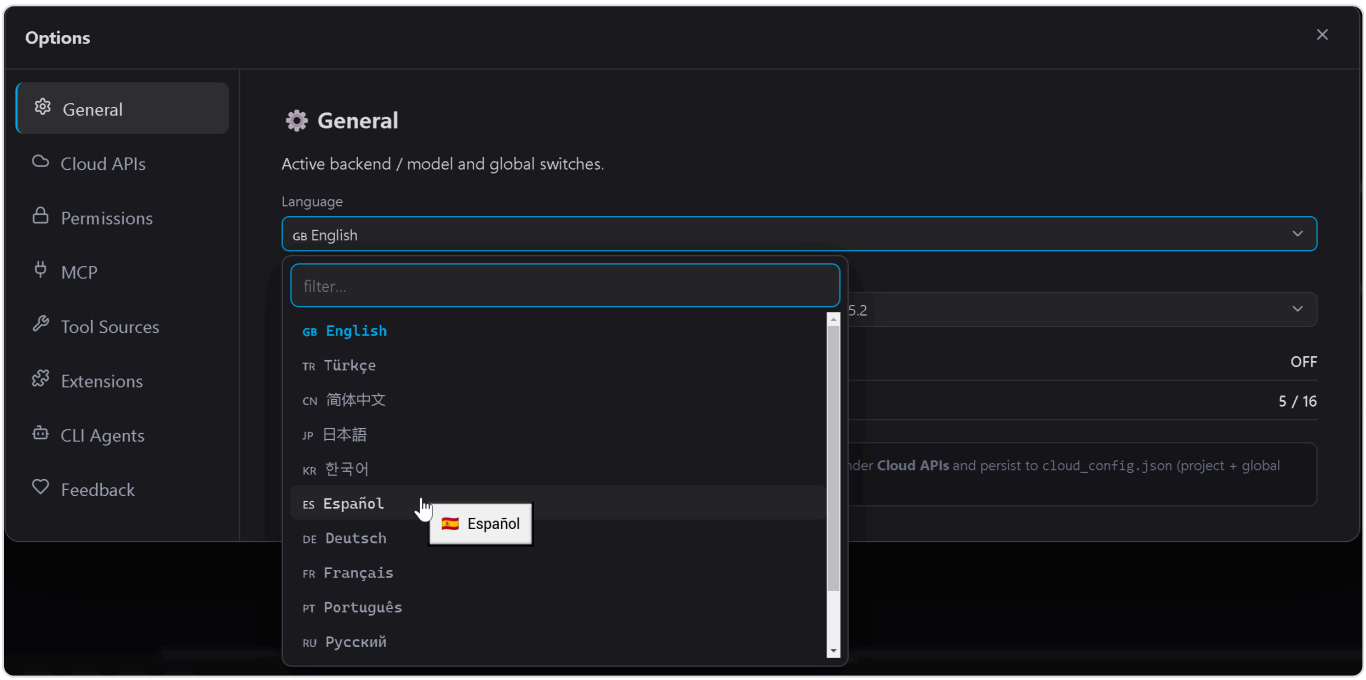


Sentari can also surface other external agents you've connected:



17. Options — Language

Sentari ships with a multilingual interface — 12 languages. Pick yours here; the whole panel — menus, buttons, and placeholders — switches immediately, with no restart.



The panel's language and the model's language are two different things

The setting above translates the **interface** — menus, buttons, placeholders.

Asking your questions in that same language is a separate matter, and it depends entirely on **the model you picked**, not on Sentari. You can write to it in essentially any language you like, but how well it answers — and how reliably it fills in tool arguments — is the model's ability, not the panel's.

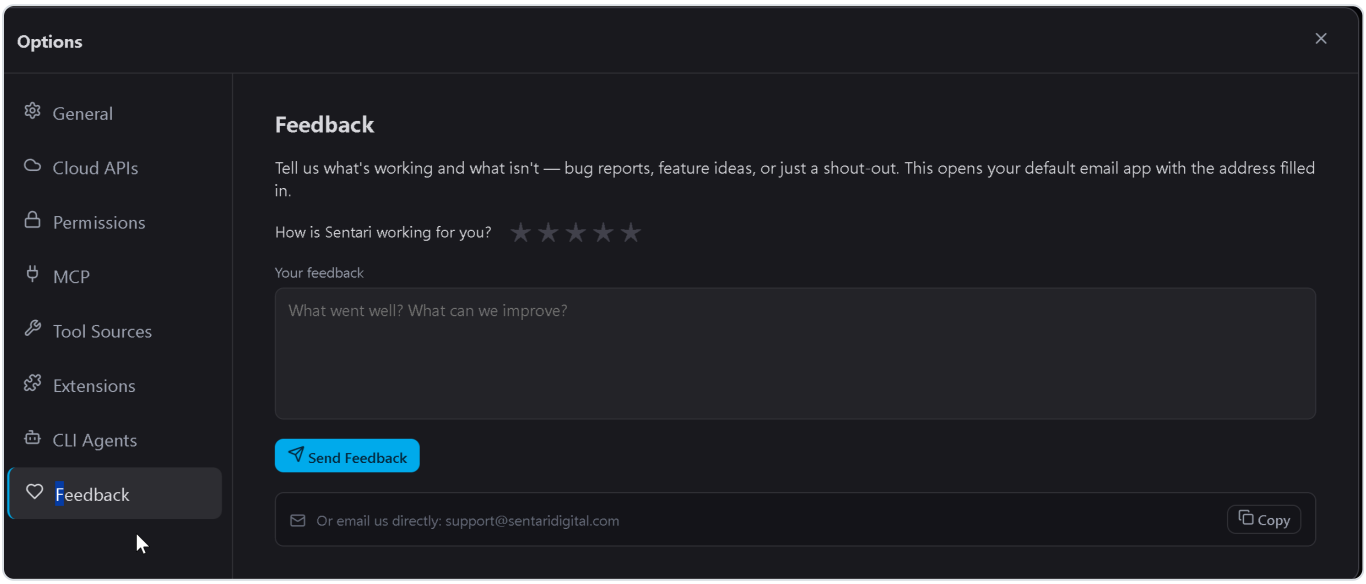
- **Frontier cloud models and larger open-weight models** handle non-English prompts comfortably, tool calls included.
- **Small local models** often degrade outside English: vaguer answers, and occasionally malformed tool arguments even where the same model handles the English version of the prompt fine.

So set the interface to your language and write however you prefer. If a small local model starts giving woolly answers, try the same prompt once in English before blaming the plugin — that one test tells you which of the two you are actually fighting. The model replies in whichever language you write in.

---

18. Options — Feedback

Send a star rating or a written note straight to the team — it opens your mail client pre-filled.



---

19. Getting the best out of Sentari — a tuning guide

Sentari is designed so the defaults work. This section is for the moments when they don't: a task that stops halfway, a local model that feels slow, or a message about a token limit. It explains what the plugin is doing behind the scenes and which two settings actually matter.

Everything here applies to local and cloud models alike, but the trade-offs bite hardest locally, because a local model has a much smaller context window than a cloud one.

19.1 MaxTok — how much the model may write in one turn

MaxTok (*Right dock → Model Parameters*) is the ceiling on a single reply. It has to cover **everything the model produces in that turn**: its reasoning, the arguments it passes to tools, and the answer you read.

Symptom of it being too low:

> ⚠ The model hit the max_tokens (MaxTok) limit in a single turn and could not recover.

— or an answer that simply stops mid-sentence.

Why it bites reasoning models hardest. A "thinking" model (Qwen3, DeepSeek-R1, QwQ, GPT-5, Claude with extended thinking, GLM) spends part of every turn reasoning before it writes anything, and that reasoning is charged against MaxTok. A model can burn its entire budget thinking and produce nothing at all — Sentari detects that case and retries with thinking disabled, but it is better not to reach it.

Situation	Suggested MaxTok
Simple, single-step edits	4 096
A reasoning model, ordinary work	8 192 — Sentari raises the floor to this automatically
Multi-step scene work, image matching, long tool chains	12 000 – 16 000
Long code or document generation	16 000+

Raising MaxTok does **not** make the model slower on short answers — it is a ceiling, not a target. The cost is that a big reply leaves less room in the context window, so raise it deliberately rather than to the maximum.

19.2 Context size — how much the model can hold at once

The context window holds the system prompt, the tool descriptions, your message, any attached image, the conversation so far, and the space reserved for the reply. When it overflows you get a refusal from the backend rather than a graceful failure.

For local models the context is set **when you load the model** in LM Studio or Ollama, not by Sentari. A useful rule:

Context	What it comfortably supports
8k	Short questions. Too small for real scene work
16k	Simple tasks, few tools
32k	**A good default** — multi-step tasks, images, a healthy toolset
64k+	Large tool chains, long conversations, big code files

Bigger is not free: a larger context means a slower first response, because the model has to process ("prefill") everything before it emits a token. On a large local vision model that prefill can take minutes, which is why Sentari stretches its stall timeout for local vision turns instead of assuming the model has hung.

19.3 What Sentari already does for you

You do not have to manage the toolset by hand. Two mechanisms run on every request.

A task router picks the tools. Sending all 244 tools (plus Epic's, on UE 5.8) on every request would be wasteful and would confuse smaller models. Instead a router reads your request and selects the relevant families — lighting tools for a lighting task, PCG tools for scattering vegetation, GAS tools for ability work. You typically see 40–160 tools offered instead of everything.

A context budget trims what still doesn't fit. Tool descriptions are prompt text, and they are not small. On a 32k local model a large selection can genuinely exceed the window. When that happens Sentari measures the real cost and drops tools until the request fits, removing the broad Epic toolsets first and keeping the curated tools the router chose for your task. You can see this in the Output Log:

```
context budget: tools 241→91 (~49907→15869 tok; ctx=32768) — Epic tools dropped first
```

That line is not an error. It means the request was made to fit rather than rejected.

19.4 Local versus cloud — an honest comparison

Sentari is BYOK: you bring your own model, local or cloud. Neither is simply better.

Local (LM Studio / Ollama)

- Nothing leaves your machine. No API cost, no rate limits, works offline.
- Smaller context, so the toolset gets trimmed more often.
- Slow first response on image tasks (prefill), then steady.
- Best with a 32k+ context and MaxTok around 8–12k.

Cloud (Claude, GPT, Gemini, GLM, and others)

- Large context, fast prefill, generally stronger multi-step reasoning.

- Your prompts and any attached screenshots go to that provider.
- Costs money per request, and rate limits apply.

A practical pattern many users settle on: local models for iteration and anything sensitive, a cloud model for the hardest one-off tasks. The sunset match in Section 6 was done entirely on a local model — capable local weights go further than most people expect.

19.5 Symptoms and fixes

What you see	What it usually means	Fix
"hit the max_tokens (MaxTok) limit"	The turn needed more room than MaxTok allows	Raise MaxTok (8k → 12k), or split the task
"The answer was cut off"	Same cause, but output had already started	Raise MaxTok and ask it to continue
"did not respond for N seconds"	The model is prefilling a large prompt, or is genuinely stuck	Normal on local vision turns; if it repeats, lower the context or use a smaller image
"stopped inside its reasoning"	A thinking model spent its whole budget reasoning	Raise MaxTok, or rephrase the request more concretely
The backend rejects the request outright	The prompt exceeds the context window	Load the model with a larger context, or start a new chat to drop history
It writes Python instead of calling tools	The model is not doing native tool calling	Check the 🔧 badge next to the model; some local builds ship without tool support

---

20. Working with Sentari (examples)

A few prompts that show the range. Sentari chooses the right tools itself — you just describe the goal.

- **"Create a weathered copper material and apply it to the selected static mesh."**
- **"Scatter pine trees and rocks on the landscape using PCG, low-poly style."**
- **"Add a Jump gameplay ability to the player character with a 2-second cooldown."**
- **"Generate a low-poly treasure chest and import it into the Content Browser."** *(Meshy / Tripo)*
- **"Set the atmosphere and lighting like in the picture I gave you."** *(📎 attach a reference)*
- **"What's causing the red compile warning? Fix it."** *(📷 send a viewport capture)*

Sentari is just as useful when it *doesn't* change anything. Asked why a map started empty, it replied with a one-line summary, the root cause — the open map was an empty 12 KB template, not the 135 KB city — and the evidence behind it: a PIE copy time of 4.5 ms against 1096 ms for the real level, plus Mass Traffic "no zone graphs" warnings. Diagnosis first, edits only when you ask for them.

If a local model ever stops responding mid-task, Sentari shows a stall notice (e.g. *"Model didn't respond for 150 s"*) and you can stop the run. That usually means the turn is too large for the model — see Section 19.

---

21. Approvals & safety

By default, Sentari **asks for approval before changing your project**. If you prefer a faster flow, enable **Auto-approve** in Options (or per-task). You can also add **Permission rules** to lock down specific asset folders or tool families.

Two more guarantees are built in rather than optional:

- **Verify, then claim.** A change is reported as done only after it compiles or passes a check. An honest "that did not apply" is the plugin working as intended, not failing.
- **Reflection pre-flight.** Tool calls resolve names against the engine's real API before acting, so a guessed name never creates a broken asset.

Always keep your project under source control (Git / Perforce / Unreal's SVN) so any change is reversible.

---

22. Beyond the panel

- **MCP server** — *Options → MCP* starts a local server on `localhost:8765`. Connect an external AI client to drive the same tools from outside the editor.
- **Epic Tool Sources** — toggle hundreds of built-in Epic editor tools on/off so the model can call exactly the capabilities you trust (UE 5.8).
- **CLI Agents** — *Options → CLI Agents* launches Codex or Claude Code pre-configured to work alongside Sentari.

---

23. Privacy

Sentari collects **no telemetry and no personal data**. Your prompts and project content go only to the AI provider you choose (or nowhere, with a local model). API keys stay on your machine — stored locally with an environment-variable fallback for those who prefer never to write a key to disk. Add `/Sentari/` to your project's `.gitignore` so per-project settings and keys are never committed. Full details: see **Data & Privacy**.

---

24. FAQ

It's not responding / "no model". Make sure a backend is selected (*Options → General*). For local models, confirm LM Studio / Ollama is running and a model is loaded.

The model isn't using tools / writes code instead of acting. Some smaller local models don't support tool-calling; Sentari falls back to a guided path. The 🔧 badge tells you upfront whether the selected model runs the tool loop.

Do I need to install Python? No. Unreal ships its own embedded Python and Sentari runs inside it via the stock PythonScriptPlugin. There is no pip step, no `requirements.txt`, and no third-party package — the plugin's Python layer is standard library only. See *Section 1*.

Can I ask questions in my own language? The interface ships in 12 languages, and you can write prompts in whatever language you like — but the quality of the answer is the *model's* ability, not the panel's. Large cloud and open-weight models cope well; small local models are often noticeably better in English. See *Section 17*.

Do I need a powerful GPU? Only for large local models. Sentari runs happily on a modest local model for routine work, or entirely on a cloud model with no local GPU at all.

Is it safe to let it edit my project? Changes are reversible if you use source control. Start with approvals on, review a few tasks, then relax to Auto-approve once you trust it.

Does it need the internet? No. With a local model and assets on disk, it runs fully offline (enable **Offline mode** in Options).

Which Unreal versions? The core toolset targets UE 5.7 and 5.8. Epic Tool Sources are 5.8-only and hide themselves elsewhere.

---

25. Tips

- **Be specific about *where*:** "the selected actor", "everything in `/Game/Env`", "the player character".
- **Capture or attach first for visual tasks:** 📷 or 📎 before asking about lighting, layout, or anything you can see.
- **Iterate:** if a result is close but not right, reply with the tweak ("make it 20% smaller", "darker, more saturated").
- **Start a new chat for a new task** — history is context, and context is a budget.
- **Match the model to the task:** a big cloud model for complex multi-tool jobs; a local model for privacy, offline work, and volume.
- **Keep source control on** — it's your undo for any AI edit.
