Tips for Using the Namazu Agent

Namazu Agent · https://namazustudios.com/docs/namazu-agent/namazu-agent-tips/

The Namazu Agent works well out of the box, but a little technique goes a long way. This page collects the habits that get the best results out of every session: how to steer the agent before it touches your code, what to do when something goes wrong, and how to pick the right model for the job. For the day-to-day workflow itself, see Common Agent Tasks.

Namazu Cloud is currently in closed beta. You can sign up for Namazu Cloud today, and once you have, a member of our team can activate the agent in your instance.

Plan Before You Code

Before letting the agent write code, make sure it is in **plan mode**. In plan mode the agent reads your workspace, researches the problem, and proposes an approach for your approval before it changes anything. You review the plan, correct its course while it is still cheap, and only then let it start editing.

This matters because the most expensive mistakes an agent makes are the ones it makes confidently, across many files, before you have said a word. A few minutes of plan review is far cheaper than a large refactor in the wrong direction. It also gives you a natural place to add constraints the agent could not have guessed: naming conventions, parts of the codebase to leave alone, or an approach you already know you want.

For small, well-defined tasks (a one-line fix, a quick script), plan mode is overkill and you can just let the agent work. For anything that touches multiple files or that you could not describe fully in a sentence, plan first.

When You Hit a Problem, Ask the Agent

If you run into a problem, ask the agent. Generally it will help. Unlike a plain chat assistant, the agent is sitting in an environment with real access: your project's code, the logs of your running instance, its database, and the instance's own API. Describe the symptom in plain language ("the login screen hangs after submitting", "matches stop being created after about a minute") and let it investigate.

This is one of the agent's real strengths. It can correlate what it finds in the code with what it sees in a live instance, which is exactly the kind of cross-referencing that is tedious to do by hand. The same applies to administrative work: if something about your instance seems off, ask the agent to look at it rather than clicking through the admin panel on your own.

Still Stuck? Ask It to File an Issue

If you get stuck using the agent even after asking the agent, ask it to file an issue with us. The agent can open a well-formed issue that captures what you were trying to do, what failed, what you already tried, and how to reproduce it, which is exactly the report a human would need to jump in. This is the escalation path for problems that look like they are in Namazu itself rather than in your game's code: the agent hands the failure to the people who can actually fix it.

Ask Us on Discord

You can always join Discord and ask us for help. Whether it is a question the agent could not answer, feedback on the workflow, or just wanting to compare notes with other developers building on Elements, the Discord is the fastest way to reach the team.

Pick the Right Model

The agent can run on many different models, and the difference matters. A small, fast model is great for administering an instance, troubleshooting, or working on a budget. A frontier model is the better choice for deep, complex coding tasks, where its higher quality more than pays for the higher price. You can bring your own agent (opencode, Codex, or Claude Code) and supply your own API key or log in to your existing accounts, or use the bundled local models that run on GPU instances for budget-

friendly work.

The table below breaks down the major models: what each is good at, and what it is not. Model generations move quickly, so treat this as a snapshot of the current landscape rather than a permanent reference.

Model	Provider	Good At	Not Great At
Claude Fable 5	Anthropic	The hardest long-horizon coding: overnight refactors, large migrations, deep debugging. Tops coding and reasoning benchmarks.	Price. It is the most expensive Claude and overkill for everyday work.
Claude Opus 4.8	Anthropic	Quality-first, multi-file refactoring, architecture, code review, and long autonomous runs that stay coherent across dozens of steps.	The priciest mainstream option and slower to respond than smaller models.
Claude Sonnet 5	Anthropic	The daily driver: near-flagship coding quality at a much lower price, strong on code generation and reviews.	The very hardest, longest-horizon problems, where Opus and Fable still pull ahead.
Claude Haiku 4.5	Anthropic	Quick edits, lint fixes, simple well-specified tasks, and fast sub-agent work at the lowest Claude price.	Complex multi-step reasoning and anything that needs sustained context.
GPT-6 Sol	OpenAI	The newest OpenAI flagship tier for demanding coding, science, and security work, with efficient token use.	Premium pricing and a very new release; most daily work does not need it.
GPT-5.6 Sol	OpenAI	Frontier coding and cybersecurity-adjacent work; strong scores on coding-agent benchmarks with fewer tokens.	Overkill for routine edits; costs well above mid-tier models.
GPT-5.5	OpenAI	Terminal-native and DevOps-style work: shell, git, CI/CD debugging. The strongest pick for command-line-heavy sessions.	Large, repo-scale refactoring, where Claude's long-context coherence wins.
GPT-5.4	OpenAI	High-volume workhorse coding with a good speed/price balance; quick at finding edge cases and bugs.	Consistency on subtle, multi-step logic, where it trails Claude's quality.

Model	Provider	Good At	Not Great At
GPT-5.4 Mini	OpenAI	Cheap, fast handling of simple tasks, classification, and summarization.	Hard reasoning and complex code; it is a utility model, not a builder.
Gemini 3.1 Pro	Google	Value flagship: near-frontier coding at roughly half the price, a 1M-token context window, and native image/video/PDF input.	Consistency on complex stateful logic; it needs more human oversight on the tricky cases.
Gemini 3.5 Flash	Google	Extremely fast (several times quicker than typical frontier models) and competitive on coding benchmarks; ideal for iterative loops and CI.	The deepest long-horizon work, where flagship reasoning models still win.
Gemini 3.5 Flash Lite	Google	The budget end: high volumes of simple, well-specified calls for very little money.	Anything complex; keep it away from real feature work.
Grok 4.7	xAI	Affordable long-context reasoning and general coding assistance at a competitive price.	Long autonomous multi-file coding sessions, where it is less proven than Claude or GPT flagships.
DeepSeek V4 Pro	DeepSeek	Open-weight frontier-class coding (MIT licensed): near-flagship benchmark scores you can self-host for data sovereignty.	Getting the most from it takes infrastructure; as a managed API its edge shrinks.
DeepSeek V4 Flash	DeepSeek	The price floor: extremely cheap output with a 1M-token context, ideal for batch jobs and CI bots.	Hard agentic work; it is built for volume, not depth.
Qwen3.7 Max	Alibaba	Open-weight frontier coding at low cost, competitive with much pricier proprietary models.	The absolute hardest tasks and the most polished agent ecosystems.
Kimi K2.7 Code	Moonshot AI	Coding-focused work and web-browsing agents, with cheap long-context support.	A smaller ecosystem and less mature tooling than the major labs' models.

Model	Provider	Good At	Not Great At
GLM-5.2	Zhipu	Budget-friendly agentic coding with a 1M-token context and self-hosting options.	Name recognition and the frontier ceiling; it trades peak quality for price.
MiniMax M3	MiniMax	Some of the cheapest frontier-class output available, with a 1M-token context.	Ecosystem maturity and peak quality on the hardest problems.
Qwen 3.8B (bundled)	Local, on GPU	Budget-friendly local model included with the agent: private, no API key needed, good for instance administration, troubleshooting, and quick questions.	Deep, complex coding tasks; it is far below frontier models and is not meant to replace them.

As a rule of thumb: use the bundled local model (or a small, cheap model) for administration, troubleshooting, and budget-friendly work, and reserve frontier models, used through your own opencode, Codex, or Claude Code account or API key, for deep and complex coding tasks.

Get Started

Namazu Cloud is currently in closed beta. Sign up for Namazu Cloud, and a member of our team can activate the agent in your instance so you can start building.

Related Pages

- Namazu Agent: what the agent is, the agent variants, model options, and how sessions run
- Common Agent Tasks: the end-to-end workflow, from starting a session to moving your game to production