

GBS SE Agent, explained in plain English

What it is, how it works, where its limits are, how it compares to the big names, and what comes next. No jargon, everyday examples, pictures.

If you remember one thing: it is a very fast, very literal junior assistant who works inside the code, shows you everything it does, and can run entirely on our own computers.

Written for	Anyone who has to understand or decide about this, technical or not
About	GBS SE Agent 0.0.13, plus MCP support from the next release
Date	26 October 2026
Online	kit.gramproindia.com/se-agent · github.com/gramproindia/agent

What is this thing, really?

A helper that works inside the code, not a chatbot that talks about it.

Imagine hiring someone on their first day. They don't know where anything is. But they are quick, they never get bored, and they have three good habits:

They look before they touch. They don't guess where the login code lives; they go and find it. The agent is not even allowed to change a file it hasn't read, and if the file changed since it looked, the change is refused.

They ask before anything risky. Running commands, changing settings, opening a file that looks like it holds a password: they stop and check with you first, showing exactly what they are about to do.

They show their receipts. When they're done, they list every file they touched, with a side-by-side comparison and a single button that puts everything back.

That is the whole product. Everything technical exists to force those three habits onto a computer program that would otherwise happily guess.

THE ONE-SENTENCE VERSION

A helper that reads and changes our code on its own, shows every step, asks permission before anything it can't undo, and can run entirely on our own computers so client code never leaves the building.

What it is not

It is not a chatbot that answers questions. A chatbot talks; this one acts: it opens files, edits them, runs the tests, reads the errors and tries again. And it is not a replacement for a developer. It is very good at the boring, repetitive half of the job and genuinely unreliable at deciding what should be built.

IT IS GOOD AT	A PERSON IS STILL NEEDED FOR
Explaining a codebase nobody on the team wrote	Deciding what should be built at all
Repetitive changes across many files	Judging whether the result is actually right
Writing the first version of tests	Anything where being confidently wrong is expensive
Narrowing down where a bug lives	Context nobody wrote down
Following conventions it was taught once	The last review before it ships

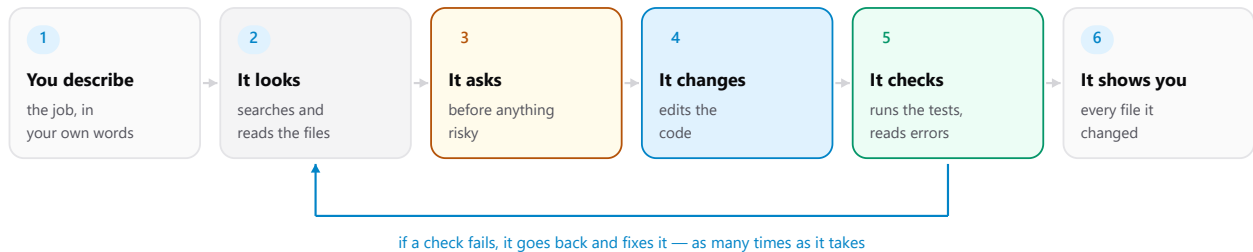
PART TWO

How it works, step by step

One instruction from you turns into a loop of small steps. The loop is the whole trick.

How it works, in plain words

One instruction turns into a loop of small steps. The loop is the whole trick.



Steps two to five repeat until the work is actually done. That is what makes it an "agent" rather than something that gives you an answer and stops.

While it works you see every step as a line in the chat: "Read src/auth.ts", "Search login", "Run npm test". You can keep typing to steer it ("actually, put it in the middleware"), and the Esc key stops it at any time.

WHY THIS MATTERS FOR COST AND SPEED

Every one of those steps is a separate question to the AI. A simple edit is three to eight steps; a big change can be twenty to sixty. So when a free plan "allows 1,000 requests a day", that is not 1,000 tasks; it is more like 30 to 100. This is the single most misunderstood number in the whole subject.

Helpers for big searches

For a big search, the agent sends a junior helper who reads a hundred files in its own room and comes back with a six-line answer. The hundred files never land on the main desk (see Part three). Up to four helpers can work at once, which is why it copes with a large codebase on a small model.

PART THREE

Three words you'll keep hearing

Model, tokens, context window. Everything else is detail.

The model is the brain

The agent itself has no intelligence. It is the hands, the eyes and the rulebook. The thinking is done by a separate thing called a model, and we can plug in different brains: one running on your own laptop, one on a company server, or one rented over the internet. Same agent, different brain, very different speed and cost.

Tokens are pieces of words

Models read tokens: chunks of about four characters. A rough rule: 750 words is about 1,000 tokens. You pay for tokens, and the brain can only hold so many at once.

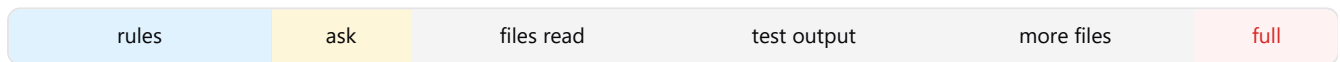
The context window is the desk

Picture a desk that fits a fixed number of pages. Everything the assistant is thinking about has to be on it: the house rules, your request, every file it has read, every test result. When the desk fills up, something has to go.

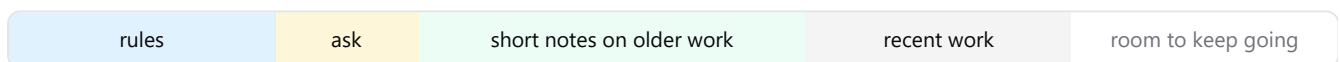
AT THE START: PLENTY OF ROOM



AFTER TWENTY STEPS: NEARLY FULL



WHAT OUR AGENT DOES: TIDIES THE DESK



It clears space in a fixed order: first old tool output becomes a one-line note, then, only if needed, it asks the AI to write a summary. The house rules never move, because keeping them identical is what lets the server answer faster on every step after the first.

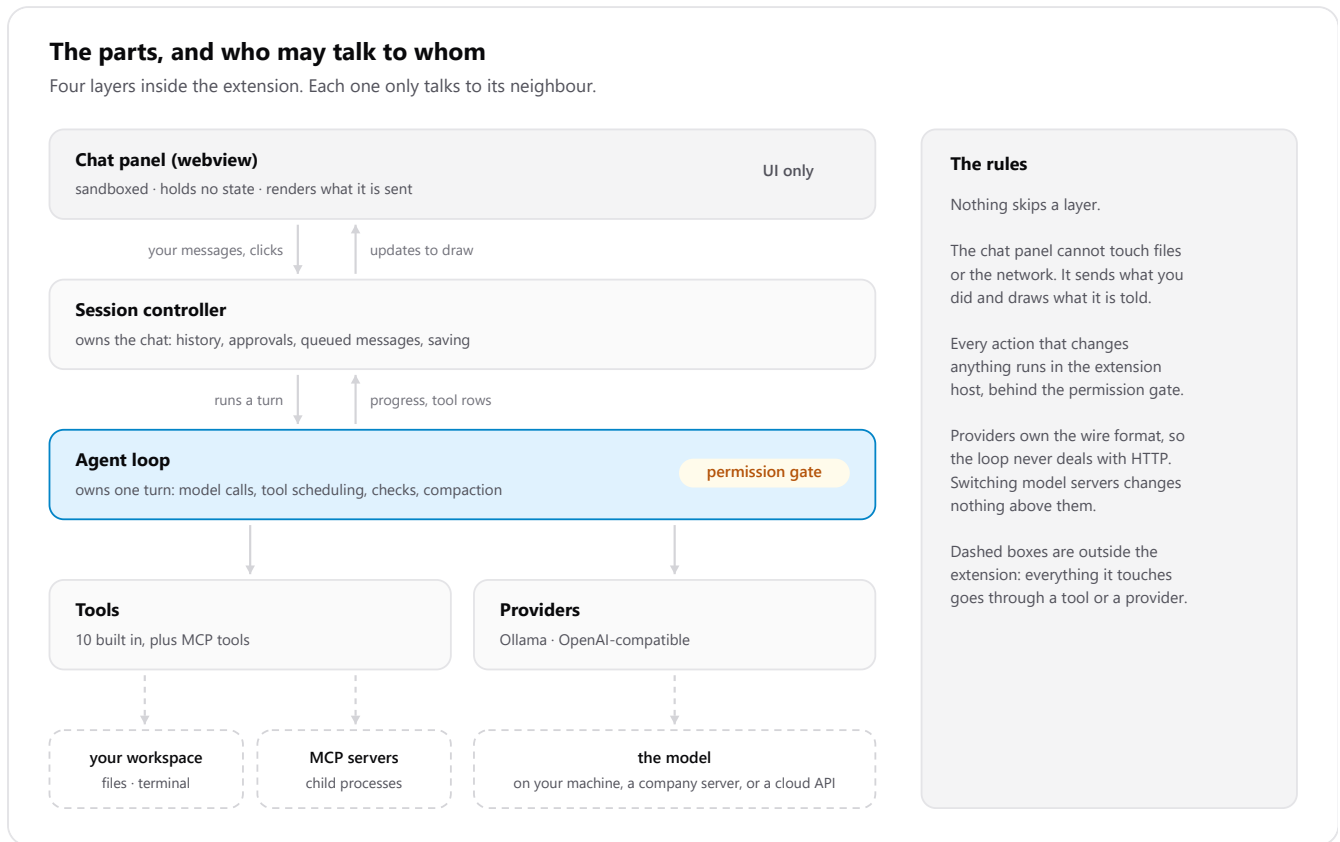
IN NORMAL NUMBERS

Our default desk is 32,768 tokens: roughly 24,000 words, or about 50 pages. A big model on a good server can have a desk four times that size. But bigger is not automatically better: models get noticeably worse at finding things in the middle of a very full desk, which is why we tidy it even when there is room.

PART FOUR

What's inside our agent

Four layers, each with one job. Nothing skips a layer, which is why it's safe.



The chat panel is deliberately powerless: it can only show things and pass on what you type. Anything that touches a real file goes through the safety guard first.

Three ways to teach it

A rules file. Short, always-true facts ("run tests with npm test", "we use tabs") go in a file called AGENTS.md that it reads every time.

Skills. A folder where we write down bigger house rules: "always use our DataGrid, never a plain table". The assistant only reads them when the job actually involves that topic, so they cost almost nothing the rest of the time. Like a recipe book on the shelf rather than read aloud at every meal.

Plugs (MCP servers). Small programs that give it new abilities, such as asking our component library what a component's real options are. Part eight explains them.

Where the brain lives, and what that costs

Three options. They trade privacy, speed and money against each other.

1. On your laptop

Like cooking at home.

- + Free, works on a plane, nothing ever leaves.
- Slower, and the brain must fit your machine.

2. On a company server

Like a staff canteen.

- + A big brain, fast, shared, code stays with us.
- Costs money up front and needs setup.

3. Rented online

Like a takeaway.

- + Strongest brains, no hardware, free tiers to try.
- Our code goes to someone else's computer.

The agent works with all three, and you switch with two clicks. Option 1 is the one nobody else makes easy, and our distinguishing feature. Before anything is sent to an outside service the first time, the agent asks, and a "cloud" badge stays on screen from then on.

Why the rented one sometimes says "server overload"

When you rent, you share. A free plan is a busy shop at lunchtime: sometimes the answer is "too many people right now, come back in a minute". That is the "429" error. It is a queue, not a bug. The agent waits a second and tries again, then two seconds, then four, and tells you it is waiting instead of looking frozen.

If we run our own server

This is buying the pizza oven instead of ordering delivery. It makes sense at volume, and the sums are about memory on the graphics card, not speed. Each conversation the brain is holding takes memory, and a bigger desk takes proportionally more: a mid-size brain with a very large desk can use 30–40 GB of card memory per person using it at that moment.

THE ONE FREE WIN

Our agent sends the same house rules at the start of every step. Server software can be told to remember that repeated opening (it's called prefix caching), and it makes every step after the first much faster for the price of one setting. Turn it on from day one.

How it keeps you in control

The safety guard, the receipts, and the dials.

The safety guard

There are three settings, chosen with one click: **Ask first** (it asks about everything), **Auto-edit** (the default: ordinary edits go ahead, commands ask) and **Full auto** (it never asks; for trusted work only). Whatever the setting, outside Full auto it always asks before it changes files that control tools, scripts or settings, opens a file that looks like it holds a password, or runs a command that chains several programs together.

"Always allow" is a narrow yes: it remembers "npm test" for the rest of the session, not "run anything".

The receipts

Every task ends with a list of changed files, a side-by-side comparison for each one, and an **Undo** button that puts every file back the way it was before the task started. It warns you first if you have edited one of those files yourself since.

The dials

Safety caps stop a runaway task burning money, one giant file flooding the desk, or an endless loop. They were set for small brains; with a better brain, some should go up.

THE LIMIT	TODAY	ADJUSTABLE?	WITH A BETTER BRAIN
Desk size	32,768 tokens	Yes	Raise it to what the model supports
Steps per task	60	Yes	120 or more for big jobs
Answer length	8,192 tokens	Yes	Double it so long files are not cut off
Tool output kept	20,000 characters	Yes	Raise it so test failures are not trimmed
Command time limit	120 seconds	Yes	Depends on the project's test suite
Lines read at once	1,000	No	A big brain could read whole files
Helpers at once	4	No	Our own server could run more
Patience with a silent server	5 minutes	No	A large model waking up can take longer

The rule we follow: if the right value depends on which brain you use, it becomes a setting. If it is an internal balancing act nobody can reason about, it stays ours to tune.

How we compare to the big names

We win on privacy and control. The gaps we listed a month ago are mostly closed.

CAN IT...	OURS	CLAUDE CODE / CODEX / CURSOR	WHAT IT MEANS DAY TO DAY
Work fully offline, for free	Yes	Partly or no	Client code never leaves the building: our biggest advantage
Ask before risky actions	Yes, strictly	Yes	Ours is stricter than most, which suits regulated clients
Learn our house rules	Yes (skills)	Yes	Level with the leaders, and it reads the same skill folders
Undo everything it did	Yes	Yes	New: one button restores every file the task changed
Point at a file: "use @this"	Yes	Yes	New: attach any file or folder by typing @
Understand a screenshot	Yes	Yes	New: paste a picture of a broken screen, with models that see images
Plug into other tools (MCP)	Next release	Yes	New: reads the same configuration file Claude Code does
Run commands in a sandbox	Not yet	Yes	Today the guard is the approval prompt; see Part nine
Finish your line as you type	No	Cursor, Copilot	A different product; we are the "do the task" tool

The honest gap: the brain

Most of what makes Claude Code and Codex feel so capable is the brain behind them, trained specifically for this kind of work. On a small local model our agent is less capable on hard tasks, however good the rest of it is. That is why moving to our own, larger self-hosted model matters more than any feature.

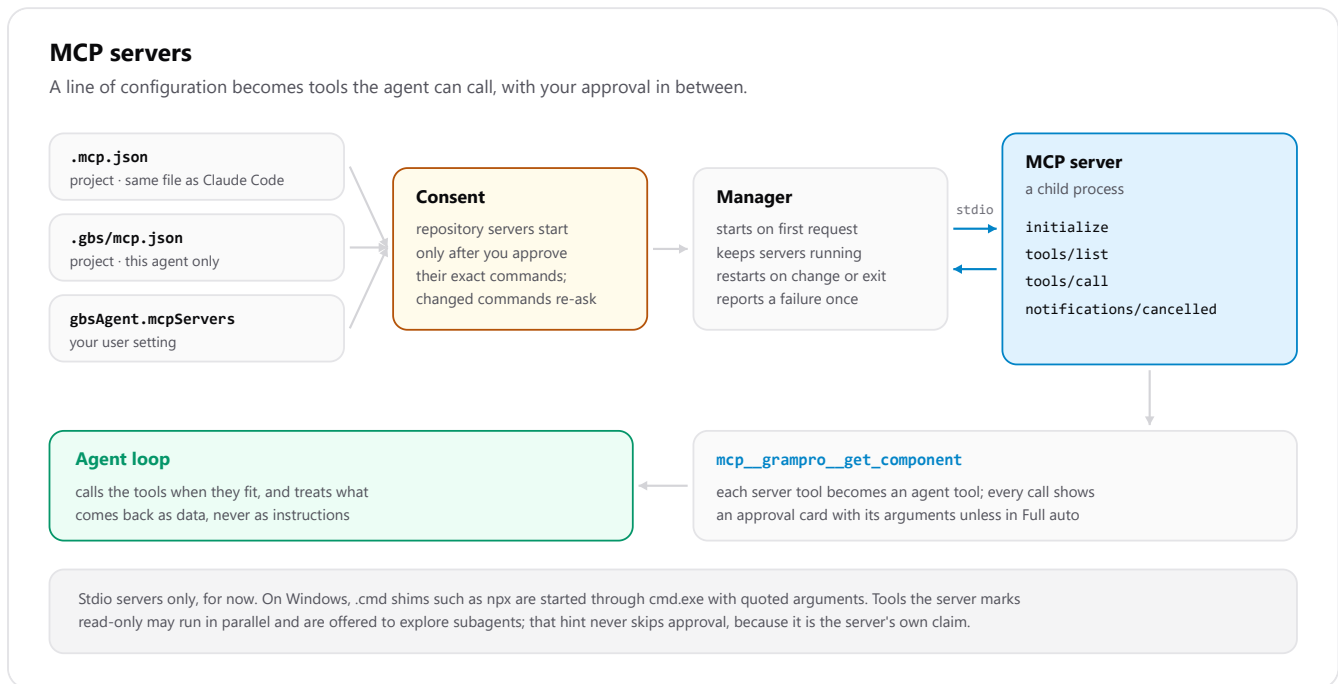
THE FAIR SUMMARY

For a company that cannot send client code to the cloud, ours is a genuinely good answer and the others are not. For a developer who just wants the strongest results and doesn't mind where the code goes, the others are still ahead on hard tasks. A stronger self-hosted model is what closes that gap.

The universal plug is built

MCP is a standard socket. Build it once, and the assistant can talk to anything that has the matching plug.

Before, the assistant could do exactly ten things, all about files and commands. Reading a design from Figma would have meant Figma-specific code, then Jira-specific code, then database-specific code, forever. MCP is the agreed shape of the plug, the USB of AI tools. We have now built the socket, and every future connection is a line of configuration instead of a development project.



What it gives us today

Our own component library, GramproKit, ships a plug. With it, the assistant asks the library what a component's real options are instead of guessing, installs the pieces it needs, and then has its own code checked against the real definitions. Combined with our house rules (skills), you get code that matches both the design and our standards, something none of the big names offer a Grampro client out of the box.

Why it is safe

A plug defined by a project we open is, in effect, permission to run a program on our machine. So the agent asks once, clearly, showing the exact command, and asks again if that command ever changes. Each time a plug is used, it asks again unless you said "always allow" for that one action.

What comes next

The list from a month ago is done. This is the new one, in order.

Last month's plan: adjustable limits (partly done), one-click undo (done), "use @this file" (done), the universal plug (done, in the next release). The repository has also moved to the company's GitHub, with automatic tests on Windows, macOS and Linux and a proper release process.

ORDER	WHAT	ROUGH SIZE	WHY NOW
1	Move to our own larger model	Depends on the model work	The brain is the biggest gap to the big names (Part seven)
2	A test set of real tasks from our own projects	About a week	So we can prove a new model or change is better before anyone switches
3	A sandbox for commands	Weeks	Limit what a command can touch, so safety does not rest on approvals alone
4	Read Claude Code and Codex settings	Days	People switching to us keep their setup on day one
5	Opt-in usage numbers	Days	Know where it struggles, without ever collecting code

A glossary, if someone quotes jargon at you

Model	The brain. We rent one or run our own; the agent is the body around it.
Token	A chunk of about four characters. 750 words is roughly 1,000 tokens.
Context window	The desk. How much the brain can hold at once.
Tool calling	The brain being able to say "read this file" instead of only talking.
Skill	A folder of house rules the agent reads only when the task needs it.
MCP	The standard plug shape that lets the agent use other programs as tools.
Subagent	A helper that does a big search in its own room and reports back briefly.
Rate limit / 429	"Too many people right now, try again shortly."
Local model	A brain running on our own hardware. Nothing leaves the building.
Prefix caching	The server remembering the repeated opening of each request, so it is faster.

GBS SE Agent 0.0.13 plus MCP support from the next release · 45 automated tests on Windows, macOS and Linux · runs on Ollama, LM Studio, llama.cpp, vLLM, Hugging Face, Groq, Google AI Studio or any OpenAI-compatible endpoint. Written 26 October 2026.
Full documentation: kit.groq.com/se-agent.