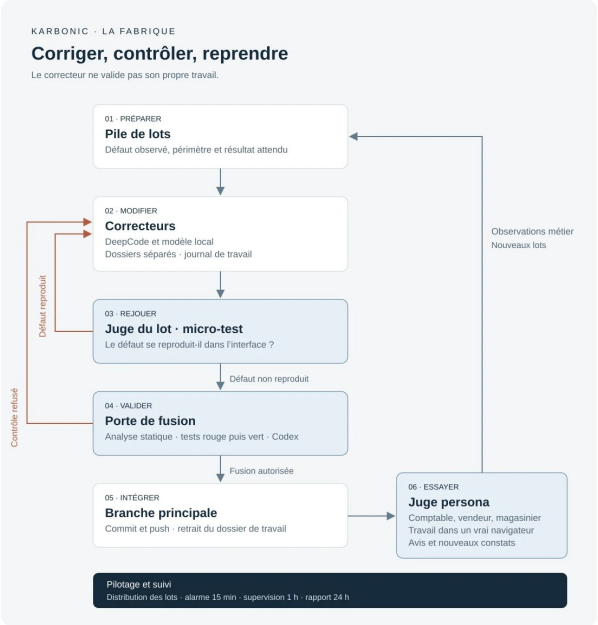
LOGICIELS ET IA

The factory: a year of development with agents

Du grand prompt à une équipe d'agents qui travaille presque seule : les essais, les modèles locaux, les juges, les erreurs, et ce qui tient après une an

Publié le 12 novembre 2026 — Imprimé le 27 novembre 2026



Published on 12 November 2026 · Karbonic

Working with agents? Nothing magical. It is like a team: you have to trust, but… check carefully. In one year, the workshop has gone from the big prompt fired off "to see what happens" to agents that move complete business management software forward almost on their own.

You see some very impressive demonstrations. The plan of a city seen from the sky, built from OpenStreetMap and a few instructions: you launch the agent in the evening, you look at the result the next morning. It really is stunning.

Business management software is another matter. Sales, purchasing, stock, accounting, payroll, hours, after-sales service: a lot of code, a lot of data, and areas that depend on one another. The functions have to hold together, the data has to stay consistent, and the user has to be able to work without knowing the developer's workarounds. You do not judge this kind of software on waking up, on an overall impression.

Agents produce code at an impressive speed. That does not yet tell you whether the changes are right, whether they solve the problem set or whether they create one in the neighbouring department. The whole organisation described here grew out of these questions.

Where we are looking from

Training as a business-IT specialist first teaches a trade: translating a client's request into software. Understanding how an accountant, a salesperson or a storekeeper works, then writing it so that a machine carries it out without betraying the need.

A career has then shown both sides of a development team. On one side, the developer, a member of teams on large systems. On the other, the IT manager: managing estates, servers and networks, leading teams of developers.

These two sides serve every day with the agents. The developer knows what a well-cut batch makes easier, and what a vague instruction produces. The manager knows that a team is worth what you check, and that an announcement that the work is finished is not an acceptance test. Agents do not change these principles. They force us to apply them with far more rigour, because they work fast and never doubt.

A year in four stages

The big prompt. Like everyone, you start with a big prompt and see what happens. We were then being sold prompt managers, and the idea that above all you had to become very good at wording. For a small application, it is perfect: “analyse this invoice and enter it” works very well. Leaving an agent to do everything alone and judging at the end no longer holds up as soon as several areas have to be handled at once. What matters is not the wording of the prompt. It is keeping a line of reasoning from one end to the other: split up, put in order, check.

We also saw repositories flourish promising almost magical reverse engineering of existing software. On a management package, with its dozens of tables and its business rules, the demonstration remains to be made. Perhaps you need to be more expert to benefit from it. The workshop’s experience points the other way: it is a good team, well led, that starts to give a usable result.

Several models in parallel. Then comes the phase where you try everything: Claude, Codex, Gemini, GLM, Kimi, Grok, DeepSeek through DeepCode. Each has its qualities. Some very capable and cheap models need tight supervision; others think at length where simple work would have been enough. Getting several agents to work on the same repository without common rules produced work that contradicted or overwrote itself.

Too many safeguards. The natural response was to put a framework around it. Guards, judges, ratchets, orchestrators, monitors: up to thirteen mechanisms coexisted, and they ended up blocking one another. One guard refused all merges because of a counting error; one orchestrator ran idle until it filled the disk. It had to be pruned.

The factory. The current organisation is simpler: small batches, interchangeable agents each within its own scope, independent judges, a pilot that distributes and a readable report. It runs on four machines, day and night. The conversation with a model is now used only for decisions.

Under way, with progress almost every day: OpenClaw and Hermes, to link the agents together and give them a shared memory.

What frontier models do very well

Writing code, of course. But one of their most useful talents lies elsewhere: putting themselves in someone’s place. “Hello, I’m Martine, an accountant for twenty years. I have to enter this month’s supplier invoices and prepare the close.” A frontier model plays this role with surprising accuracy. It looks for the functions where an accountant would look for them, gets irritated by the same detours, and notes what stops her from finishing her work.

This capability is what makes up the workshop’s persona judge today, described below. It does not replace technical tests. It measures something else: can a person in the trade work with this software?

Which model for which data

The choice of a model is not made on performance alone. It depends first on what you entrust to it. There is no reason to send confidential data to an American model, or to a Chinese one. There are plenty of reasons, though, to take advantage of their power to write code or process an image with nothing at stake.

The classification adopted comes down to three levels:

| Nature of the information | Where it is processed | Examples | | --- | --- | --- | | Confidential: legal files, accounting, payroll, a company’s images | Locally, on the workshop’s machines | Summaries of files, product imaging | | Business-related, not secret, but must stay in Switzerland | Models hosted in Switzerland (Infomaniak) | Reading invoices, assistant built into the software | | Source code, fictitious test data | Frontier models, American or Chinese | Writing and correcting code, business simulations, audits |

Each model has its advantages and drawbacks, and everything is moving very fast: the best choice of three months ago often no longer is. Hence the value of not depending on any one of them, and of open-weight

models, which you can host, examine and specialise yourself.

Local AI: a year of trials

Documents. ClearMyDocs, the workshop's document summarisation engine, served as the first test bench. In early 2026, on an 8 GB graphics card, several small models were compared on a real legal file of fifty documents. Some simply did not fit in memory. Others ran, but produced abundant and unreliable text, or read the wrong document. The smallest gained in speed what they lost in accuracy.

The bench was then completed with a high reference: a much larger model, hosted in Switzerland, to see what is possible on the same task. We also moved from Ollama to llama.cpp, to control exactly what is sent to the model, with no hidden template.

The most useful lesson did not come from the models. In one trial, a series of exceeded timeouts emptied the search axes, and the chain declared 100% coverage… on zero axes. It stopped after reading a single document, while calling itself complete. Changing model would have changed nothing. The chain had to be reworked, then a deterministic check added: each extracted fact is verified against the source document before it goes into a summary.

For about six months, ClearMyDocs v4 has been working with a modest-sized model that loads properly on ordinary hardware and handles real data without "producing jam". The same engine can also rely on a model hosted in Switzerland or on a frontier model, without changing anything else. The fidelity check depends on no model, which makes it possible to compare them honestly on the same file.

Code. Consumer graphics cards of 8 to 12 GB, spread across several machines, host reduced, quantised models, useful for short tasks. A serious code model does not fit on them. Hence the purchase of a mini PC with 128 GB of unified memory, installed under Fedora with llama.cpp.

On this type of machine, it is memory throughput that limits, not its size. A large dense model of 70 billion parameters produces a few words per second there: unusable in an agent loop. Mixture-of-experts (MoE) models change the picture: all their weights fit in memory, but only a small part works for each word produced.

After a series of trials, Qwen3 Coder Next was chosen, connected to OpenCode.

It handles small, short, bounded batches, and it is part of the team like the other agents: eleven batches in the last twenty-four hours. It does not replace the hosted models on the big projects, where a lot of files have to be reread very fast. It takes its share of the day-to-day work, with no subscription, and nothing leaves the house.

The next avenue: entrusting part of the business simulations to a local model, with a frontier model as a second-level judge. The criterion is set in advance: if the local model picks up at least 80% of the defects found by the frontier model on the same simulation, it takes over.

Training. LoRA trials are under way. These small adaptation layers specialise a model without retraining it entirely. The idea is to teach the local model what has already been judged good: accepted code, validated screens, a product to be represented without distortion. Putting the framework around the work upstream, rather than being turned down downstream.

An automated image workflow

The other major local project concerns product photography, in watchmaking. The aim is to automate the actions of a professional retoucher: take the official photo of a watch and place it in a scene (resting on a counter, on wood, on water), without ever distorting a dial numeral, a hand or a bracelet link. A company's images must not leave the premises: everything runs locally.

The workflow follows the order of the manual work in Photoshop:

1. **Cut out**, part by part: dial, case, lugs, crown, links.
2. **One layer per object**, with contiguous pixels, as a retoucher would do.
3. **Position, then deform**: angle, perspective, reducing the number of links.
4. **Reprocess the material** only where it is not clean, a poorly detected brushed finish for example.
5. **Place in the scene**, at the right size.
6. **Shadows and relighting.**

Each step has its tool. For cutting out, CLIPSeg turned out to be too coarse, but remains useful for locating areas. SAM 2.1 and BiRefNet give a fine cut-out, metal

edges included. Background generation and editing go through ComfyUI, driven without an interface through its API, with several families of models tried in turn: SDXL with ControlNet, FLUX.2, Qwen-Image. Blender is automated by script. It does not model the watch: it calculates the light, shadows, reflections and the geometry of the positioning. OpenCV applies the deformations.

The judges are fixed before any tuning. A pixel-by-pixel comparison checks that the dial has not moved compared with the original. A vision model judges whether the scene is credible. If the numerals cannot be read, or the watch does not look real, the image does not pass.

The errors were instructive. A generative model left in charge of the geometry invents the shape: a plausible case, but a wrong one. Pasting the real pixels back line by line produces the opposite: tears and a “pixel soup”. The right unit of work is the object, as for a retoucher. Another lesson: licences change from one version to the next. A model published under a free licence can return, in its next version, under a non-commercial research licence. You check before integrating.

This workflow is still a prototype. It already achieves realistic scenes, but not yet the consistency of a production workflow. The method, however, is the same as for code: documented steps, judges fixed in advance and a log of what works and what does not.

The current circuit

The stack of batches. Each area of the software has a control matrix. A box that fails becomes a batch: an objective described from the user’s side, a measured finding, and the neighbouring functions that must remain operational. Technical leads are noted as hypotheses, which observations may rule out. The batch stays small, even when the context supplied to the agent is broad. “Redo the module” leaves too much room for interpretation and makes the result impossible to check.

The correctors. DeepCode handles most of the corrections on two dedicated machines; the local model takes the small batches. Each agent works in its own folder and within a defined scope of files. It records what it does during the work, not only at the end: if the session stops, the attempts and intermediate results remain available. A shared mailbox, which everyone reads before starting, replaces repeated instructions.

The micro-test. An agent that announces it has finished supplies a report, not a validation. The corrector therefore never closes its own batch. Another agent first replays the defect in the interface: about five minutes, against forty-five for a full run. If the problem recurs, the batch goes back to the corrector.

The merge gate. It brings together static analysis, tests and a verdict delivered by Codex. A test supposed to cover the defect must have been seen to fail before the fix, then pass after it. Without this counter-check, it can pass without detecting anything. After validation, the change joins the main branch and the working folder is removed.

The persona judge. A series of validated fixes does not prove that software is usable. Regularly, a model takes the role of an accountant, a salesperson or a storekeeper and uses the software in a real browser. It is given a job to accomplish, without being asked to be lenient. Its opinion concerns what it was able to do, and what prevented it. Its observations become batches again. From time to time, a fresh judge who knows nothing of the history goes back over the work of the judges in place: these sometimes end up accepting a difficulty simply because they have already seen it.

The pilot. On the main workstation, a pilot distributes the batches. An alarm without AI checks every quarter of an hour that no process has silently fallen over; a supervisor runs every hour; a report brings together the last twenty-four hours. The Git repositories, the branches and the results of the checks give access to the detail: you must be able to find out why a change was accepted or rejected, without rereading dozens of conversations.

What cost time

The team described in YAML. The first trials described the whole organisation in YAML files: prompt manager, areas, managers, developers, judges. On paper, everything was there. It did not work.

The chief agent. An agent in charge of thinking for all the others and allocating their work never gave a satisfactory result. A pilot that distributes precise batches is useful; entrusting it with all the understanding of the project is not.

The scores. “1,980 tests out of 1,983” does not say whether the three failures prevent work, or whether the other

tests check what matters. The list of blockers, with their cause, is far more useful for deciding what to do next.

Changes of orchestrator. A handover to another model in the middle of a project consumed most of a quota for minimal progress. Rebuilding the understanding of a project under way costs more than you would imagine.

Tests that do not click. A test that queries the API can pass while the screen is broken. It keeps its value for processing, but a user journey is validated in the interface.

What holds up

- Small batches, each within its own scope, with an input and output contract.
- Measuring instruments first, fixes afterwards: a doubtful report generates doubtful batches.
- After two unsuccessful attempts at the same defect, the code is instrumented so that it points out for itself where it breaks.
- The judges and their thresholds are fixed before anything is tuned.
- Validation is done on a fresh installation, not on a database repaired by hand over the weeks.
- We commit and push often; no working folder is left orphaned.
- No machine is left idle while there are batches ready.

Twenty-four hours of work

From the morning of 11 November to the morning of 12 November, four machines launched 110 batches and produced 86 merges. Eighty-five of these batches came directly from the persona judge’s observations.

The judge went through fourteen times, across eight areas. At every pass, its opinion remained: not yet usable. That is exactly its role.

This measurement covers one day of activity, not a productivity gain compared with a human team. It shows the volume of work possible, and the gap that can remain between this volume and a product ready for use.

For anyone starting out

- **Be wary of the big prompt.** It suits a small application. Beyond that, it is the breaking down that makes the result, not the wording.
- **Start with the breakdown, not the model.** A good batch with an average model gives better results than a vague instruction with the best model.
- **Never let an agent validate its own work.** A second agent, or a human, makes the finding.
- **Test like a user.** Click, read the screen, do the work requested; a frontier model playing the accountant does this very well.
- **Have the state written down.** An agent remembers nothing; it only knows what has been written.
- **Choose the model according to the data,** then according to the task and the cost.
- **Measure local before believing in it.** The size of the memory and its throughput decide what really runs.
- **Lighten the checks regularly.** A safeguard that blocks for no reason costs as much as a bug.

Let’s share our experiences

After a year, code production has changed scale. The work of supervision has changed with it: preparing batches, maintaining checks, examining feedback from the business, deciding priorities. That is, at heart, the trade of the business-IT specialist: translating a need into software, and checking that the software really meets the need.

This set-up is not fixed, and it was built in large part on mistakes. Many teams and freelancers are running the same trials today, each in their own corner. Comparing our methods would save everyone time: how you organise your agents, how you judge their work, which models you keep local and why, what has worked and what has failed. Feedback is welcome.