

Five design patterns for AI agents, and how to build them in Dify



Appendix to the AI Agents workshop

LeX Consultancy B.V. · 20 September 2026

Anyone who wants to build “an agent” first has to answer one question: how much may the model decide for itself? The answer determines how you build, what it costs and how much can go wrong. The field distinguishes five patterns, from “decide nothing at all” to “decide, but under supervision”. You have already built all five in the guides without them being called that.

The rule of thumb: choose the simplest pattern that solves the problem. Every step up costs more time, more credits and more chance of a surprise.

Pattern	Who decides the steps	Example from the series	Guide
1. Single call	You, in advance; the model does one thing	Text at level (one LLM step)	1
2. Reason and act (ReAct)	The model, step by step, with tools	School guide assistant, Parent letter assistant (agent with knowledge base)	2, 3
3. Plan and execute	First a plan, then separate executors	School news (three channels from one message), Test week planner	3, 4
4. Self-critique	The model assesses and improves its own work	The Checker in School news with checks; the Materials check	7, 6
5. Gatekeeper	An independent check lets the result through or not	The Gatekeeper (code) in School news with checks; the human in the loop of the Study coach	7, 8

1. Single call (single-shot)

How it works. The user supplies something, the model does exactly one operation, done. No loop, no tools, no choices along the way.

```
input → [ LLM: one instruction ] → output
```

Choose this when the task is the same every time: classifying, summarising, rewriting, extracting something from a text, converting a form into a fixed layout.

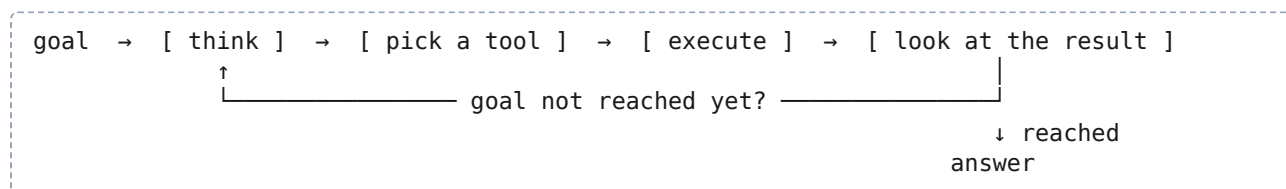
Education example. “Turn this parent complaint into a structured form: subject, class, urgency, requested action.”

In Dify. A Workflow with User Input → LLM → Output. Two LLM steps in a row (as in guide 1: rewrite and then write questions) is still this pattern: you decide the order, the model decides nothing.

Advantages. Fast, cheap, predictable, easy to test. **Disadvantages.** Cannot deal with unclear input and cannot look anything up. If the user forgets something, you get a wrong answer instead of a question.

2. Reason and act (ReAct)

How it works. The model gets a goal and tools. It thinks about the next step, picks a tool (search, calculate, consult a knowledge base, call an API), looks at the result and decides again. That repeats until the goal is reached.



Choose this when the model has to look something up or try something and you do not know in advance how many steps that takes: answering questions from documents, web research, getting something from a database, troubleshooting.

Education example. “What are the rules for leave outside the holidays, and what do I have to do as a form tutor?” The agent searches the school guide, finds two chapters, combines them.

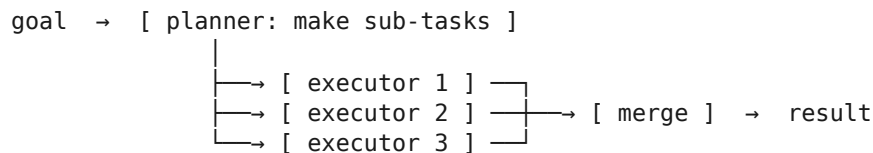
In Dify. The Agent app in Studio (guides 2 and 3): instruction plus Knowledge plus Tools; the agent decides itself when to search. And the Agent Console (guides 4 and 6), where the model can also read files, run code and search the web. The label “Used Knowledge” above an answer is this pattern in miniature: the model decided to search.

Advantages. Can handle unexpected questions, uses real information, is flexible.

Disadvantages. Slower and more expensive (every loop is a model call), more chance of a detour, and you need boundaries: a maximum number of steps, which tools yes and no, what to do on an error. In Dify those are the rules in the instruction and the choice of which Tools you switch on.

3. Plan and execute (planner-executor)

How it works. One step first makes a plan with sub-tasks. Then a separate step (or a separate agent) does the work per sub-task, in parallel where possible. At the end the results are merged.



Choose this when the task consists of recognisable parts that can stand alone: a report with fixed chapters, a message for several channels, an analysis in steps (collect, analyse, write, check).

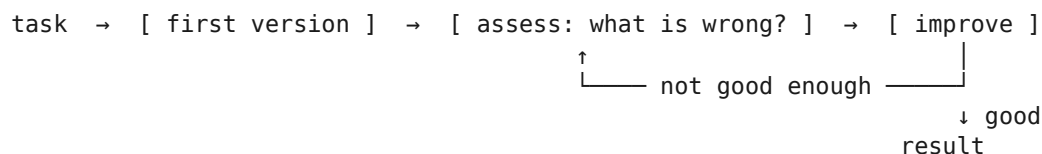
Education example. One core message becomes a website item, a parent app message and a social post, each according to the rules of that channel.

In Dify. School news (guide 3) is this pattern with a fixed plan: you wrote the plan as three LLM blocks with their own prompts, plus an Output that merges. If you want the model to plan itself, you put a first LLM block that makes a list of sub-tasks, and let an **Iteration** node work through that list. In the Agent Console the model does this by itself for larger tasks: it first writes a plan and works through it (the Test week planner in guide 4 shows that).

Advantages. Complex tasks become manageable, every executor can be tested and improved separately, and parts can run in parallel. **Disadvantages.** More building work, and a bad plan delivers neatly executed wrong work. Keep the plan visible, so a person can correct it before the executors start.

4. Self-critique (reflexive)

How it works. The model makes a first version, then assesses it itself against criteria, and improves. Possibly a few rounds, until the quality is good enough.



Choose this when quality matters more than speed and the criteria can be named: texts for publication, code, learning material, anything where “good enough” has a list of requirements.

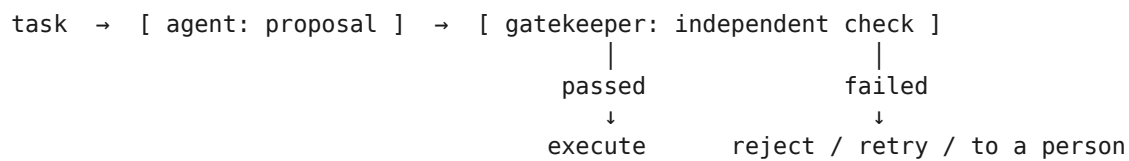
Education example. “Write a student version of this text at level B1. Then check yourself: sentences longer than fifteen words? Words from the list of words to avoid? A missing step in the explanation? Rewrite where needed.”

In Dify. A Workflow with two LLM blocks: the first writes, the second gets the text plus the criteria and the task “assess and improve”. The Checker in guide 7 is exactly that. If you want several rounds, you use a **Loop** node with a stop condition. A cheaper alternative that is often enough: one prompt with the instruction to write first, then check against the named points, and give only the improved version.

Advantages. Noticeably better texts, fewer slips, and the criteria force you to name what “good” is. **Disadvantages.** Two to three times the cost and the waiting time. And beware: the model assesses itself. A factual error it believes itself stays in. Self-critique improves form and completeness, not truth.

5. Gatekeeper (verifier-gated)

How it works. The agent’s result does not go straight out, but past an independent check: a fixed set of rules, a schema, a calculation, a second model with only the checking task, or a person. Only what passes the gate is executed or shown; what fails is rejected, retried or passed to a person.



Choose this when a mistake really does damage: money, grades, communication to parents, anything with personal data, anything covered by policy or law.

Education example. An agent drafts a message for the parent app. The gatekeeper checks: no student names, no words from the forbidden list, at most 80 words, a date in the future. If one rule fails, the message goes to the communications officer with the reason attached.

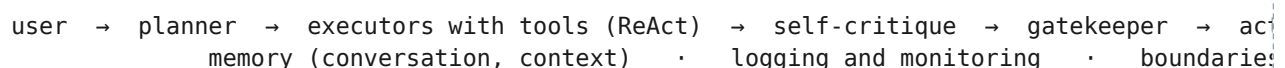
In Dify. A **Code** node or an **IF/ELSE** node after the LLM: hard rules (length, forbidden words, required fields) you check with code, not with a model. The Gatekeeper in guide 7 is six lines of Python. For softer rules a second LLM block with only the question “does this meet the following rules, answer YES or NO with the reason”. The outcome steers the IF/ELSE: on to Output, or to a branch that presents the message with the reason to a person. A **Human Input** node (guide 8) is the most honest gatekeeper there is.

Advantages. Predictable and explainable: you can say exactly why something did or did not go through. Mandatory rules are really enforced, not just “asked for” in a prompt.

Disadvantages. An extra step, and the rules have to be good: a gate that is too strict stops everything, one that is too loose does nothing. Never let the agent be its own gatekeeper.

Combining

In practice you stack patterns. A mature application often looks like this:



For a school that means:

- **Start at pattern 1.** If a fixed workflow works, an agent is unnecessary. Most tasks at a school are pattern 1 or 3.
- **Go to pattern 2 when the input varies** and something has to be searched. Set boundaries straight away: which sources, how many steps, what to do in doubt.
- **Add 4 when the output is published** and there are criteria.
- **Add 5 as soon as anything goes out** or personal data is involved. A human in the loop is a valid, often the best, gatekeeper.
- **Log everything.** In Dify every run is under **Logs**. Without a log you cannot learn from mistakes or explain what happened.

Guide 6 uses all five. The Materials check has a fixed intake (1), searches for open educational resources (2), makes a plan per support need (3), checks its own student version for sentence length and completeness (4), and states licences and refuses names before anything is delivered (5).

To remember

- The pattern determines how much the model decides for itself. Choose the lowest that works.
- You have built pattern 1 (guide 1), 2 (guides 2, 3, 4, 6), 3 (School news, Test week planner), 4 (the Checker) and 5 (the Gatekeeper, the human in the loop).
- Self-critique improves form, not truth. A gatekeeper enforces, a prompt only asks.
- No pattern guarantees a good answer. Design for the failure: what happens when it goes wrong, and who sees it?

Licence: CC BY-SA 4.0 (<https://creativecommons.org/licenses/by-sa/4.0/>). You may copy, adapt and reuse this document, including at your own school, provided you credit the source (Guido van Dijk, LeX Consultancy B.V., <https://git.lexconsultancy.nl/guido/dify-guides>) and share your adaptation under the same licence. The LeX Consultancy logo is excluded; product names are trademarks of their owners. Schools, people and documents in the examples are fictional.

Owner and contact: Guido van Dijk, LeX Consultancy B.V. · guido@l-e-x.nl

