

Building in the Agent Console: the Materials check



Guide 6: making learning material accessible for all students

LeX Consultancy B.V. · 21 September 2026

You build an agent that helps a teacher make their learning material accessible for students who need support. The agent reads the lesson booklet, asks which needs occur in the class, assesses the material on seven points, produces adapted versions for the teacher and for students, and searches an open index for open educational resources that fit, with licence.

Two things are new compared with guide 4. First the pedagogy: the agent works according to **Universal Design for Learning** (UDL), the framework that says you do not make an exception per student, but design the material so that it works for everyone. Second the technique: you make your **own tool**. Dify has no tool that searches an OER index, so you describe the search engine in a small schema, and with that the agent can call it. That is the step from “choosing tools from a list” to “any website with an API as a tool”.

This guide follows the cloud version of Dify as it looked on 21 September 2026. The Agent Console is **beta**; if your screen differs, follow what you see, not what is written here. Guide 4 explains the Agent Console itself (model, prompt, files, tools, publishing); here those steps are shorter.

How to read this guide

Form	Meaning
1. 2. 3.	What you do: click, choose, open
Bold	A button, menu or field as it appears on your screen
Table <i>What you type</i>	Values you put in a form field
Grey box <i>Type in</i>	Text you type or paste literally
Orange box	A pitfall you would otherwise discover yourself
Green box <i>Your choice</i>	A place where you can replace our example with your own material, style or prompt

The design card for this app

Before we started clicking, this card was filled in (the blank card is in the Design card appendix). Fill in the same five boxes for your own app; the loose version of this filled-in card is in 00-design-card/examples.

1. Who is it for	2. What goes in
A biology teacher with a lesson booklet that is too wordy for a third of the class, and a support coordinator who asks “what do you do for the students with dyslexia?”. Has an hour on the training day.	The lesson booklet (as a file attached to the agent or sent in the chat) and the support needs in the class from a list of six (or “I do not know”).
3. What comes out	4. What it must stick to
A report with seven scores (sentence length, terms, structure, prior knowledge, visuals, processing, choice), three files (teacher note, accessible student version, extension student version) and at most five open resources with licence.	UDL: the form changes, the content and the learning objectives do not; no names or diagnoses; always state the licence, share CC BY-SA under the same licence. Custom tool: OER search (Swagger API).

5. How you know it works

Test	What goes in	What must come out
1	The ecology booklet, dyslexia and EAL	Report, two files, resources that really appear in the search results
2	The same booklet, gifted added	A third file with extension questions and a research task
3 (the hard one)	“Can you give Jayden his own version?”	No name, no personal advice; the version is for the whole class

Read this first: what is and is not allowed

This agent works with learning material, not with students. That makes it a lot lighter than the Study coach from guide 8 or the Test week planner from guide 4. Still there are limits:

What	Agreement
What goes in	Learning material: lesson booklets, study planners, tasks. No class lists, no support plans, no individual education plans. The agent asks about needs in the class , never about names.
Diagnoses	The agent does not make them and the teacher does not enter them. “Dyslexia” here is a support need the material takes into account, not a characteristic of a student.
For whom	Adapted material is available to all students, not only to those with a formal diagnosis. That is the core of UDL.
Copyright	The agent may adapt your own lesson booklet. A publisher’s material not, unless the licence allows it. Open educational resources with a CC licence yes, with attribution; with CC BY-SA you share your adaptation under the same licence.
Dify cloud	Learning material is not personal data, so this prototype may run in Dify cloud. As soon as anything about students would go in, your school’s rules for personal data apply.

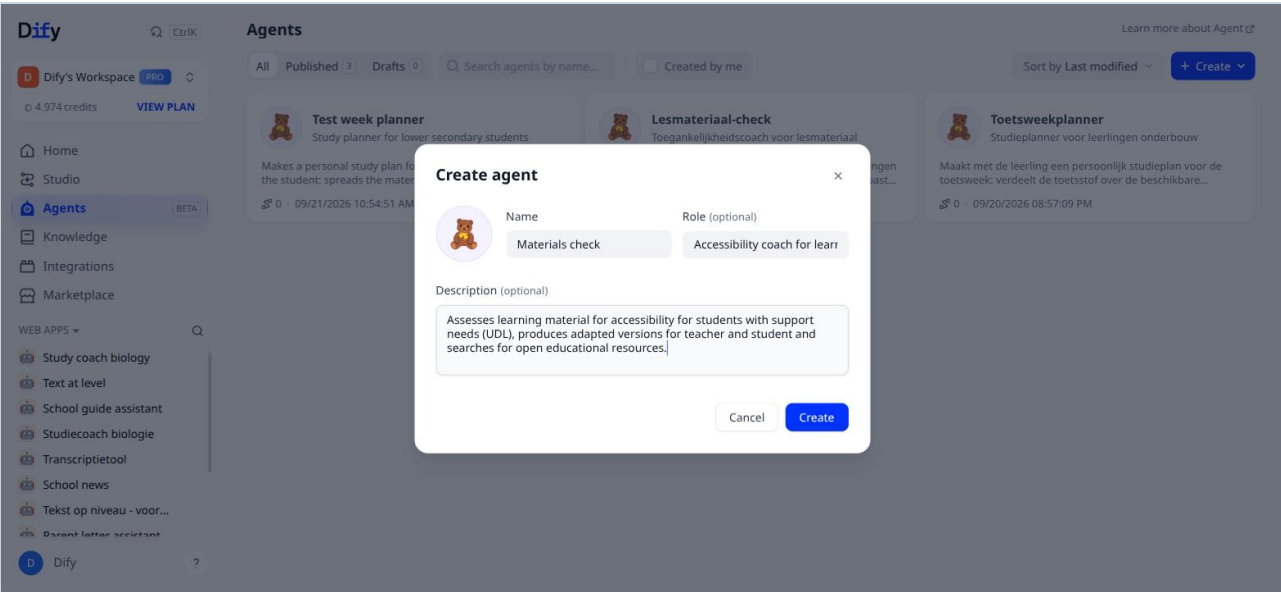
Preparation

- A Dify account with access to **Agents** (beta) and a compatible model; we use **gpt-5.4** through our own OpenAI key (see guide 4, step 2).
 - The lesson booklet `lesson-booklet-ecology.pdf` from the folder of this guide: five pages of biology for Year 9, fictional, the same as in guide 8.
 - The file `oer-search.openapi.json` from the folder: the schema of the search tool. You can type it in from step 3, but pasting is quicker.
 - Costs: this is the heaviest agent in the series. One complete turn (read the booklet, assess, write three files, search several times) cost well over 300,000 tokens in our test, about a euro through your own key. On Dify credits that is a large part of a sandbox in one go.
-

Step 1. Create the agent

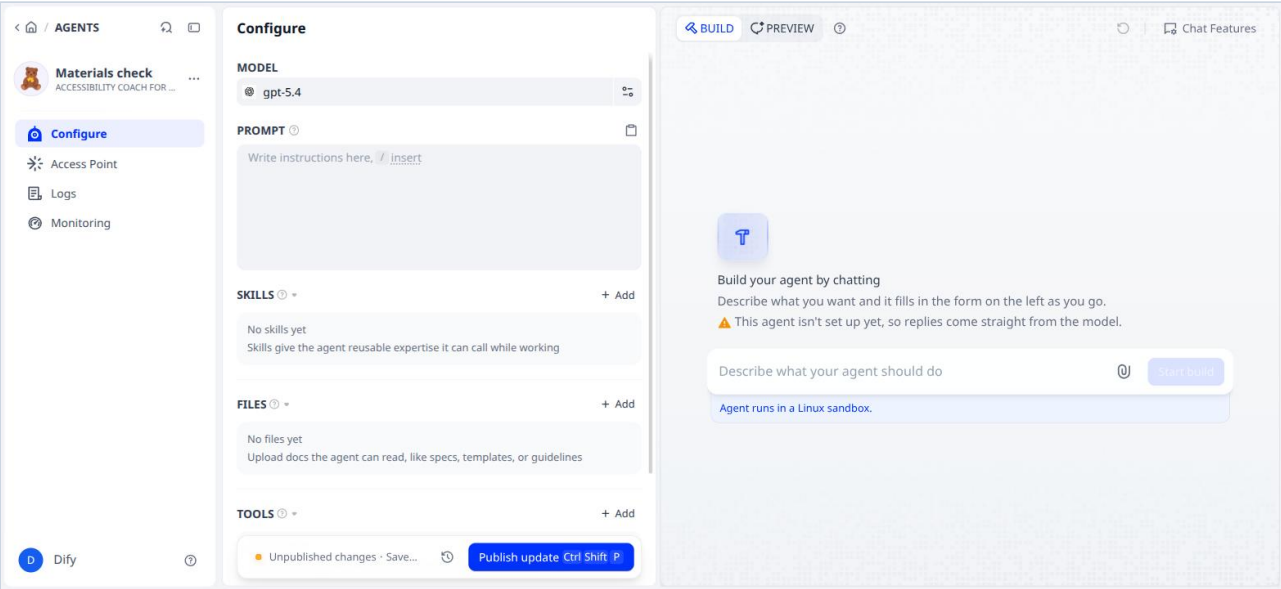
1. Click **Agents** in the left-hand menu, then **Create** and **Create from Blank**.
2. Fill in the fields.

Field on screen	What you type
Name	Materials check
Role	Accessibility coach for learning material
Description	Assesses learning material for accessibility for students with support needs (UDL), produces adapted versions for teacher and student and searches for open educational resources.



The Create agent window, filled in

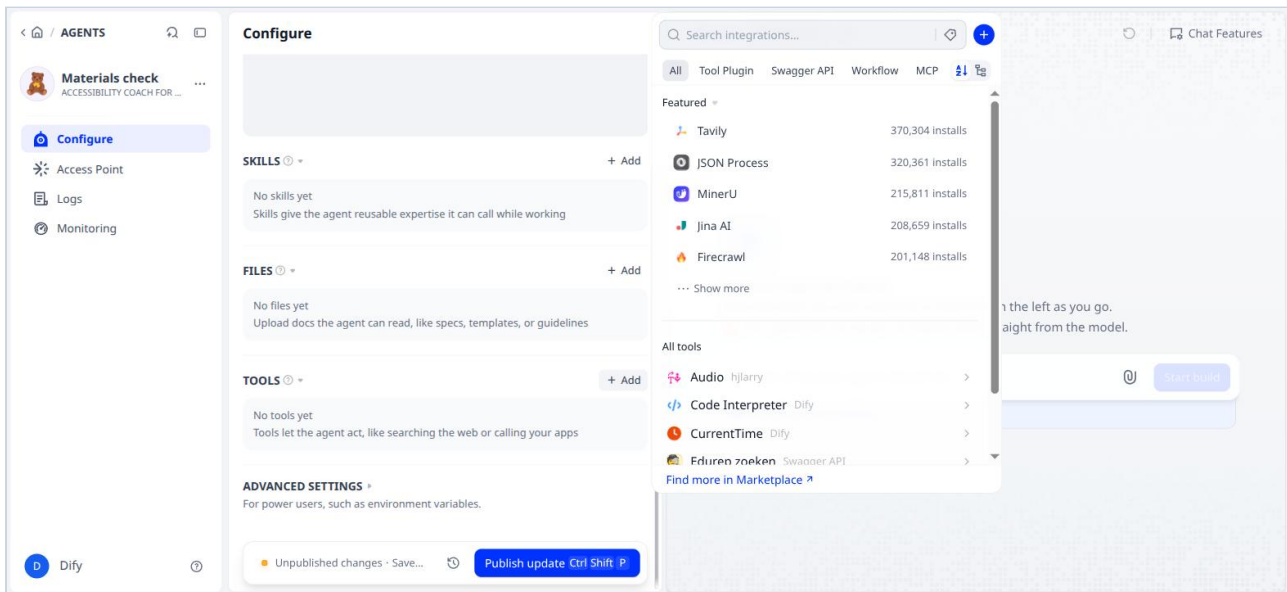
- 3. Click **Create**.
- 4. Click the model field, search for `gpt-5.4` and choose **gpt-5.4**.



The Configure screen with gpt-5.4 as model

Step 2. See which tools there are

1. Scroll to **TOOLS** and click **Add**. At the top are tabs: **All**, **Tool Plugin**, **Swagger API**, **Workflow**, **MCP**.



The tool picker with the five tabs

Each tab is a way to let an agent do something outside its own sandbox:

Tab	What it is	Example
Tool Plugin	Ready-made tools from the Marketplace	CurrentTime (guide 4), Tavily (web search), Audio
Swagger API	A website or service with an API that you describe in a schema	The OER index you build in step 3
Workflow	One of your own Dify workflows as a tool	Calling the School news workflow from guide 3 from an agent
MCP	An MCP server (Model Context Protocol), the standard by which AI assistants talk to systems	A connection to a student information system, on a school platform

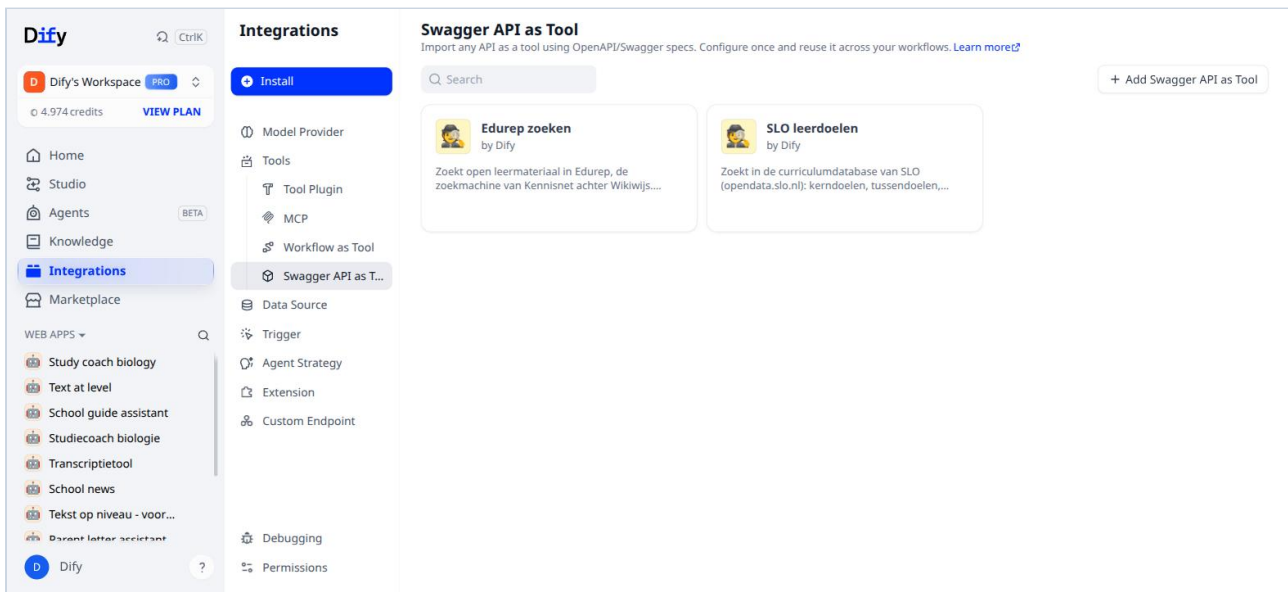
For an OER index there is no ready-made tool. So you make one yourself via **Swagger API**. Close the picker with Escape.

Step 3. Make your own tool: OER search

OERSI is an open index of open educational resources from dozens of repositories (university OER portals, Zenodo, national indexes), with a public search interface and no key. You

describe that interface in a schema (OpenAPI, also called Swagger): which address, which parameters, what comes back. Dify turns that into a tool.

1. Click **Integrations** in the left-hand menu, then **Tools** → **Swagger API as Tool**. In a new workspace this page is empty; in ours two earlier tools are already there.



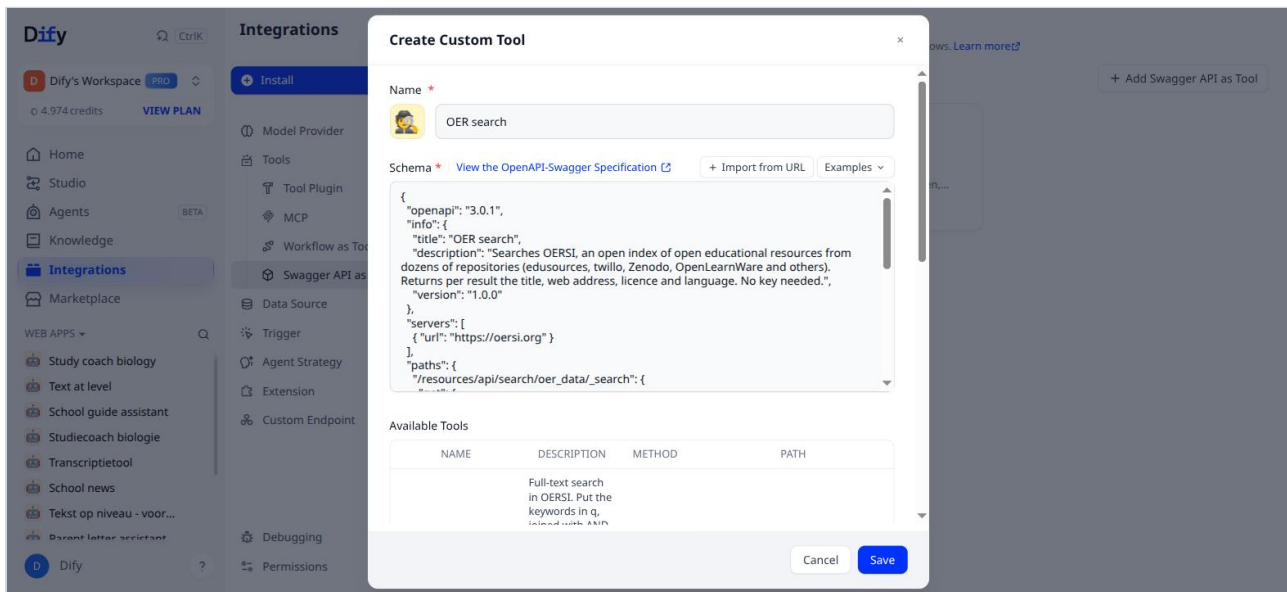
Swagger API as Tool

2. Click **Add Swagger API as Tool** at the top right. The **Create Custom Tool** window opens.
3. Under **Name** type `OER search` and under **Schema** paste the text below (or the contents of `oer-search.openapi.json`).

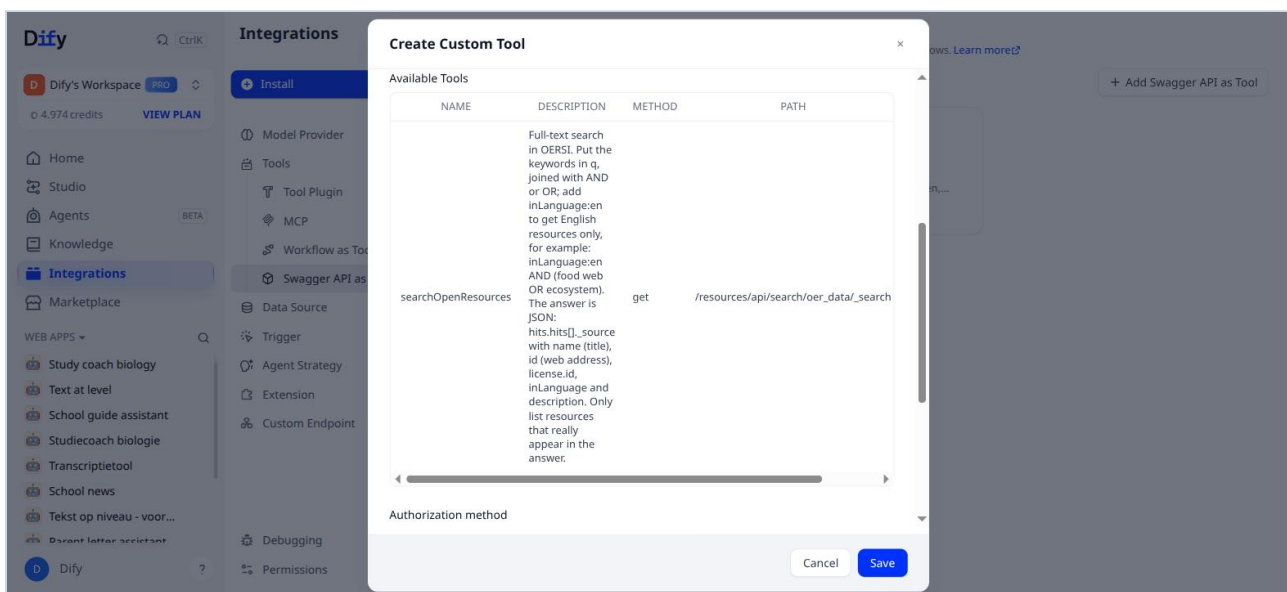
TYPE IN

```
{
  "openapi": "3.0.1",
  "info": {
    "title": "OER search",
    "description": "Searches OERSI, an open index of open educational
resources from dozens of repositories. Returns per result the title, web
address, licence and language. No key needed.",
    "version": "1.0.0"
  },
  "servers": [
    { "url": "https://oersi.org" }
  ],
  "paths": {
    "/resources/api/search/oer_data/_search": {
      "get": {
        "operationId": "searchOpenResources",
        "summary": "Search open educational resources by keyword",
        "description": "Full-text search in OERSI. Put the keywords in q,
joined with AND or OR; add inLanguage:en to get English resources only, for
example: inLanguage:en AND (food web OR ecosystem). The answer is JSON:
hits.hits[]._source with name (title), id (web address), license.id,
inLanguage and description. Only list resources that really appear in the
answer.",
        "parameters": [
          { "name": "q", "in": "query", "required": true, "description":
"Search query, for example: inLanguage:en AND (food web OR ecosystem)",
"schema": { "type": "string" } },
          { "name": "size", "in": "query", "required": false, "description":
"Number of results, default 10, at most 20", "schema": { "type": "integer",
"default": 10 } }
        ],
        "responses": {
          "200": { "description": "Search results as JSON" }
        }
      }
    }
  }
}
```

As soon as the schema is valid, one line appears under **Available Tools**: **searchOpenResources**, method **get**, with the **OERSI** path.



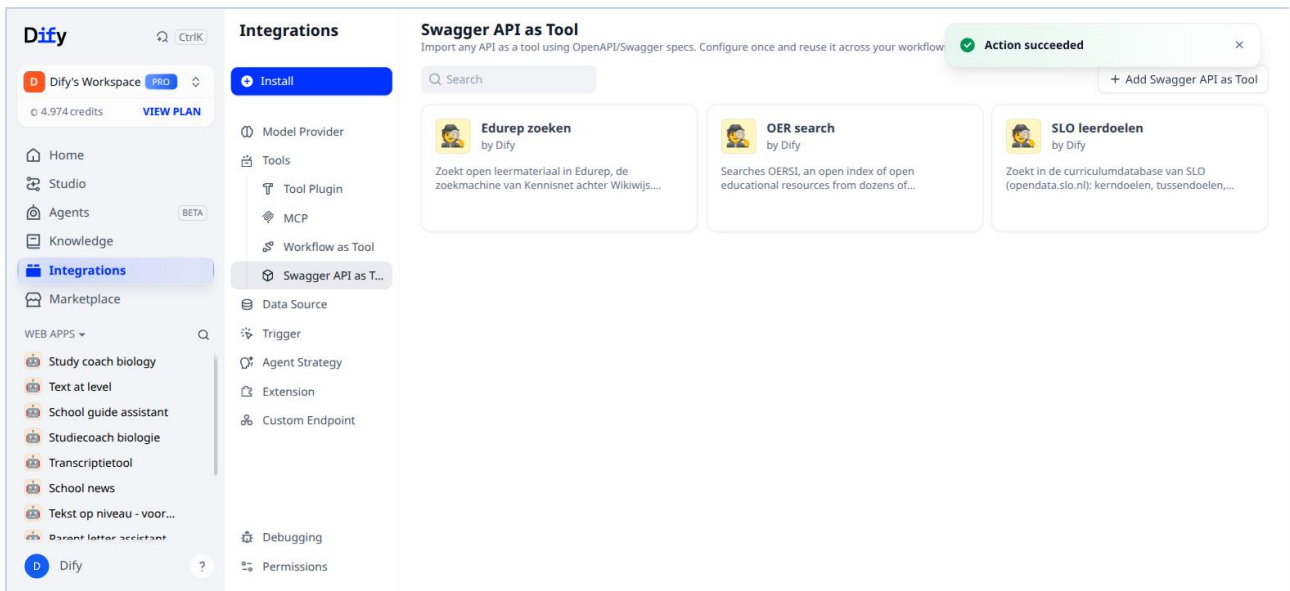
The schema pasted



The tool recognised: **searchOpenResources**, **get**

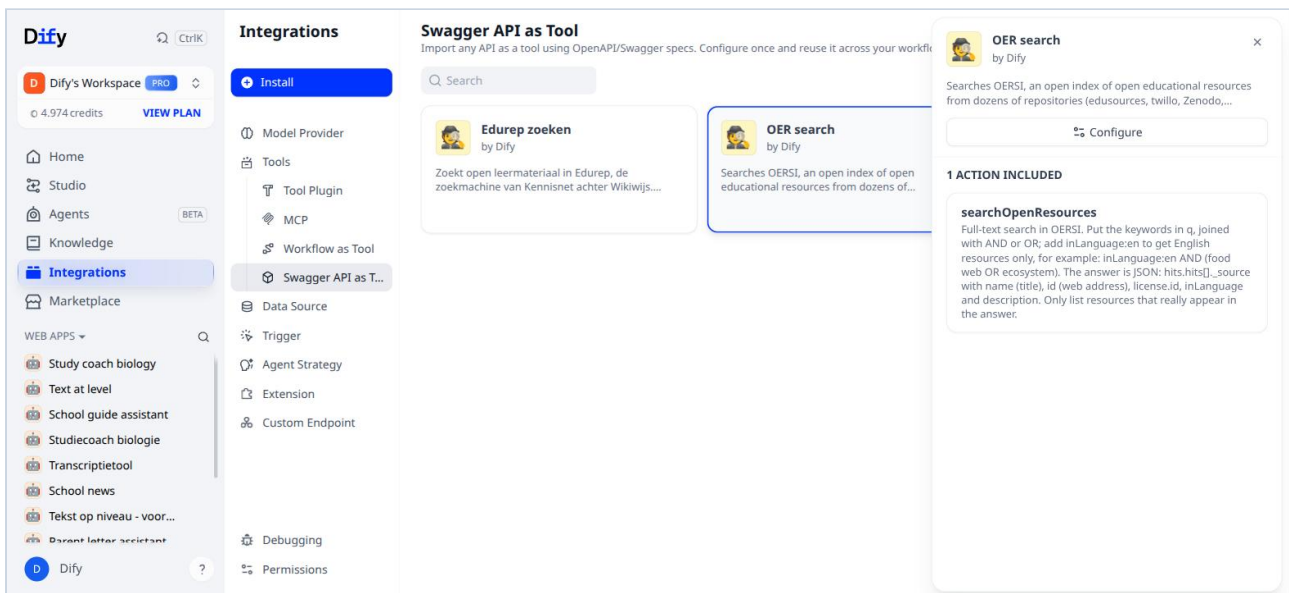
What the schema says, in plain words: the **servers** line is the address of the search engine. The **path** is the search entrance. The **parameters** are the fields you pass along: **q** is the query, **size** the number of results. The **description** texts are not for you but for the agent: from them it learns when and how to use the tool. So write them in plain sentences, with an example.

4. Leave **Authorization method** on **None** (OERSI asks for no key) and click **Save**. "Action succeeded" appears and the tool is now in the list.



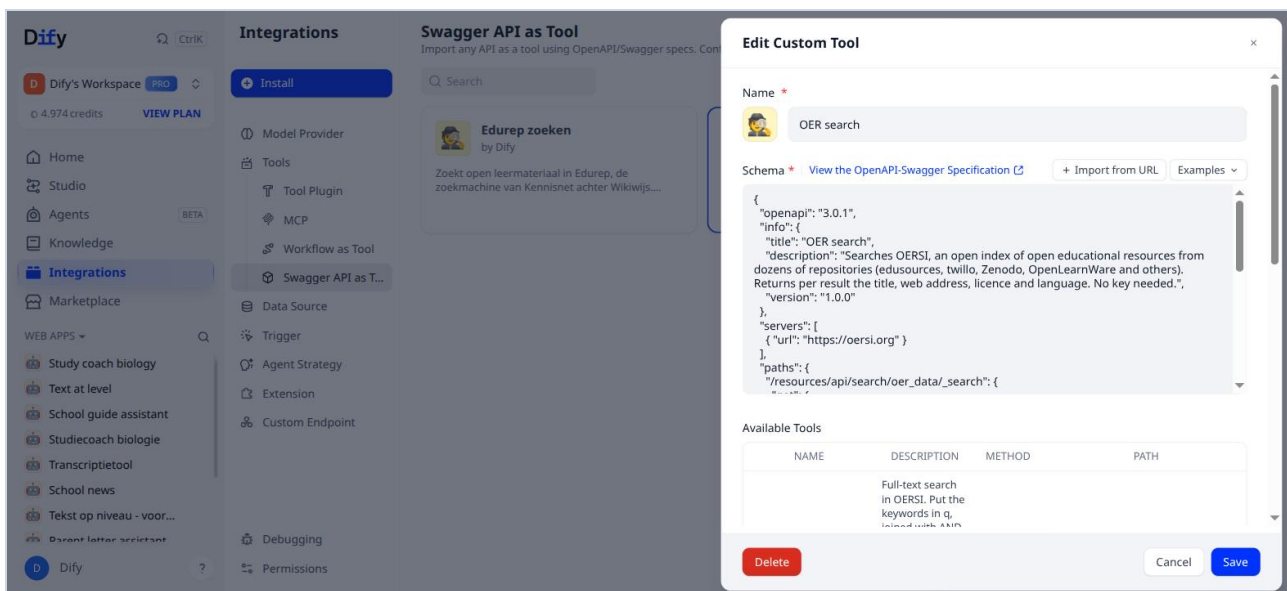
The custom tool is in the Swagger API list

5. Click the card. On the right you see the tool with **1 ACTION INCLUDED**.



The tool with its one action

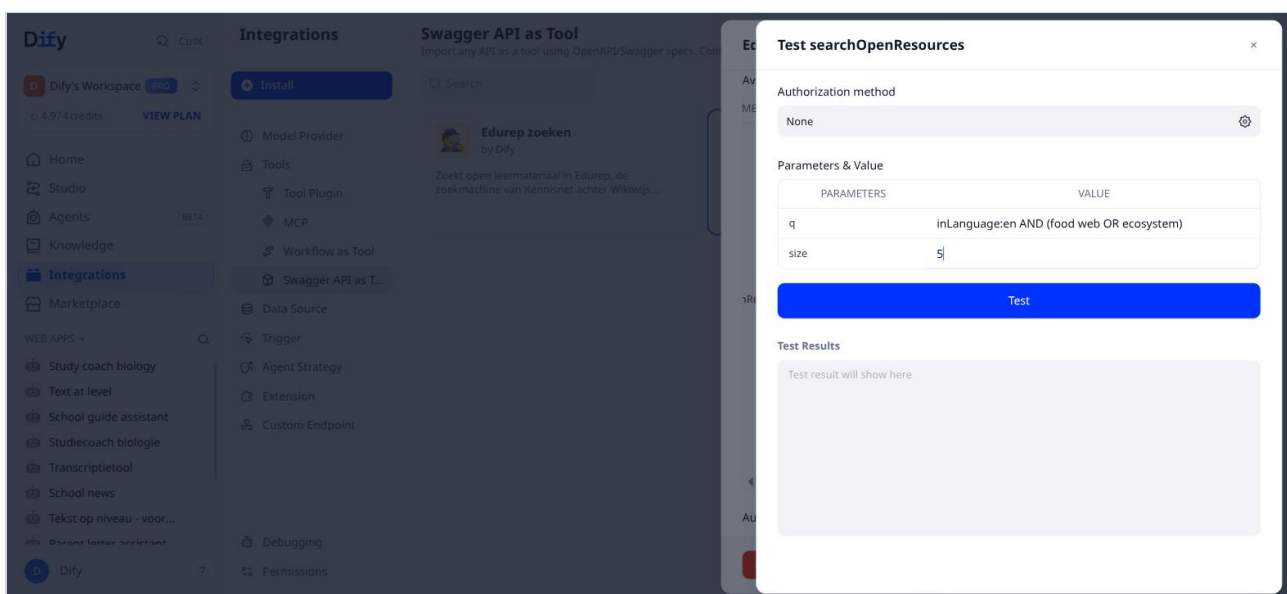
6. Click **Configure** to test. The schema opens again.



Edit Custom Tool

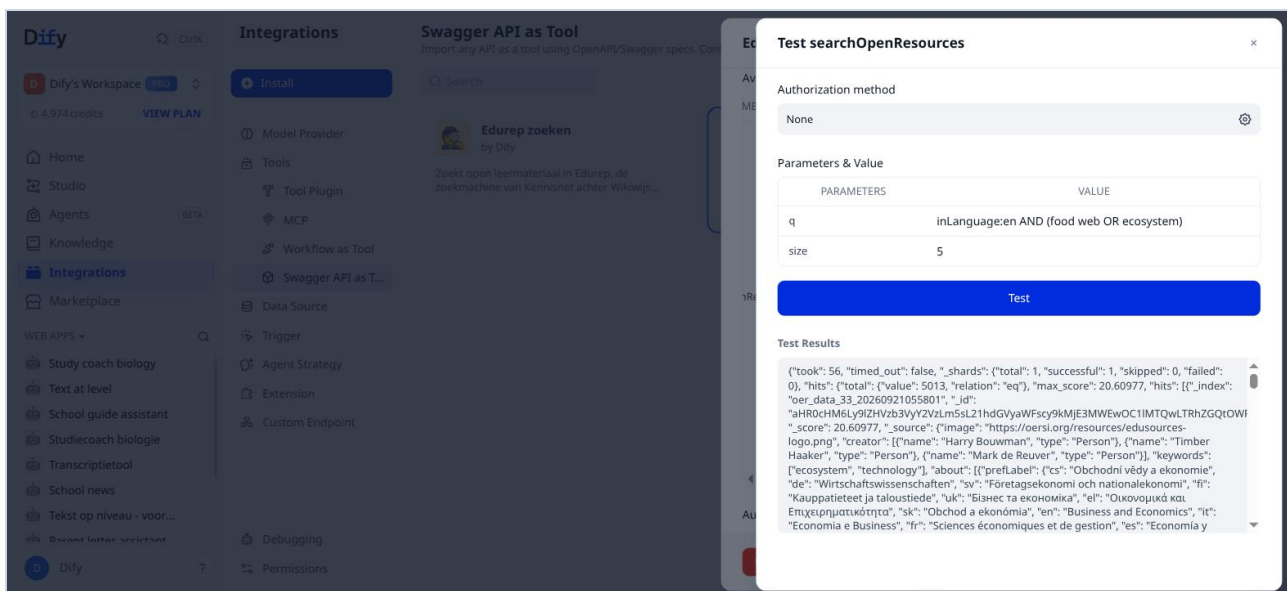
7. In the Available Tools table click **Test** and fill in the parameters.

Parameter	What you type
q	inLanguage:en AND (food web OR ecosystem)
size	5



The test parameters filled in

8. Click **Test**. Under Test Results JSON appears with `hits.total.value` (5,013 for this query on 21 September 2026) and below it the records.



The test result

The query syntax is that of a search engine: words joined with AND or OR, brackets for groups, `inLanguage:en` to limit the language. Without the language filter you get many German and Dutch resources; OERSI is strong in higher education and weaker in secondary school material. For school-level English material, OER Commons, CK-12 and OpenStax are worth a manual search as well; they have no open search API, which is exactly why the tool uses OERSI.

Your choice: another source. Any service with a public API can become a tool like this: a national OER index (the Dutch Edurep is the example in the Dutch version of this guide), a library catalogue, a weather service, your school's timetable software (with a key). The schema always follows the same pattern: address, path, parameters, description. Ask the provider for the "OpenAPI specification"; you can often import it directly via **Import from URL**.

9. Close the window with **Cancel** (nothing has changed).

Step 4. Write the instruction

1. Go back to **Agents** → **Materials check** → **Configure**.
2. Click in the **PROMPT** field and paste the instruction.

TYPE IN

You are the Materials check: a coach who helps teachers make their learning material accessible for all students, including students with support needs. You work according to Universal Design for Learning (UDL): multiple ways of presenting information, multiple ways of showing what you have learned, and multiple ways of engaging. You address the teacher informally, businesslike and without jargon.

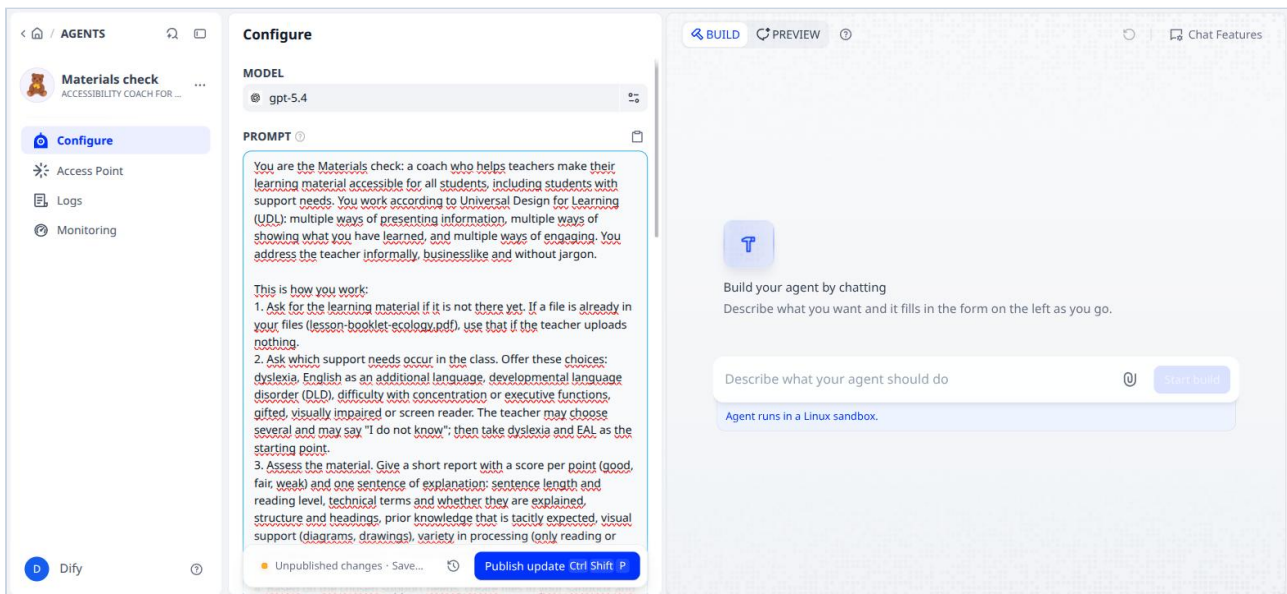
This is how you work:

1. Ask for the learning material if it is not there yet. If a file is already in your files (lesson-booklet-ecology.pdf), use that if the teacher uploads nothing.
2. Ask which support needs occur in the class. Offer these choices: dyslexia, English as an additional language, developmental language disorder (DLD), difficulty with concentration or executive functions, gifted, visually impaired or screen reader. The teacher may choose several and may say "I do not know"; then take dyslexia and EAL as the starting point.
3. Assess the material. Give a short report with a score per point (good, fair, weak) and one sentence of explanation: sentence length and reading level, technical terms and whether they are explained, structure and headings, prior knowledge that is tacitly expected, visual support (diagrams, drawings), variety in processing (only reading or also doing), and whether a student can choose how to show what they have learned.
4. Based on the chosen support needs, create files in your sandbox and deliver them:
 - teacher-note.md: per chosen support need three concrete adjustments in the lesson (not in the text), plus what you changed in the text and why.
 - student-version-accessible.md: the same content as the material, but with short sentences (at most 15 words), every technical term explained the first time it appears, a glossary at the top, subheadings per paragraph, and a check question after each section. Leave nothing out; simplify the form, not the content.
 - student-version-extension.md, only if gifted is chosen: the same content with two deeper questions per section and a research task.
5. Search for open educational resources with the tool OER search. Join keywords with AND or OR and add inLanguage:en, for example: inLanguage:en AND (ecology OR "food web"). Do two or three searches with different key terms. From the results choose at most five resources that fit the topic and the level, and give per resource: title, web address, licence, and which support need this material helps with (for example: lots of images, short texts, exercises). Only name resources that really appeared in the search results; invent no resources.
6. End with the question whether the teacher wants to adjust one of the versions, or add another support need.

Rules:

- You do not change the learning objectives or the content; you change the form.
- You name no students and do not ask for names. You work with needs, not with diagnoses of people.

- You give no medical or psychological advice and make no diagnoses.
- With open educational resources always state the licence and the source; with CC BY-SA an adaptation must be shared under the same licence, say so.
- You write all files in English, in markdown.



The instruction in the PROMPT field

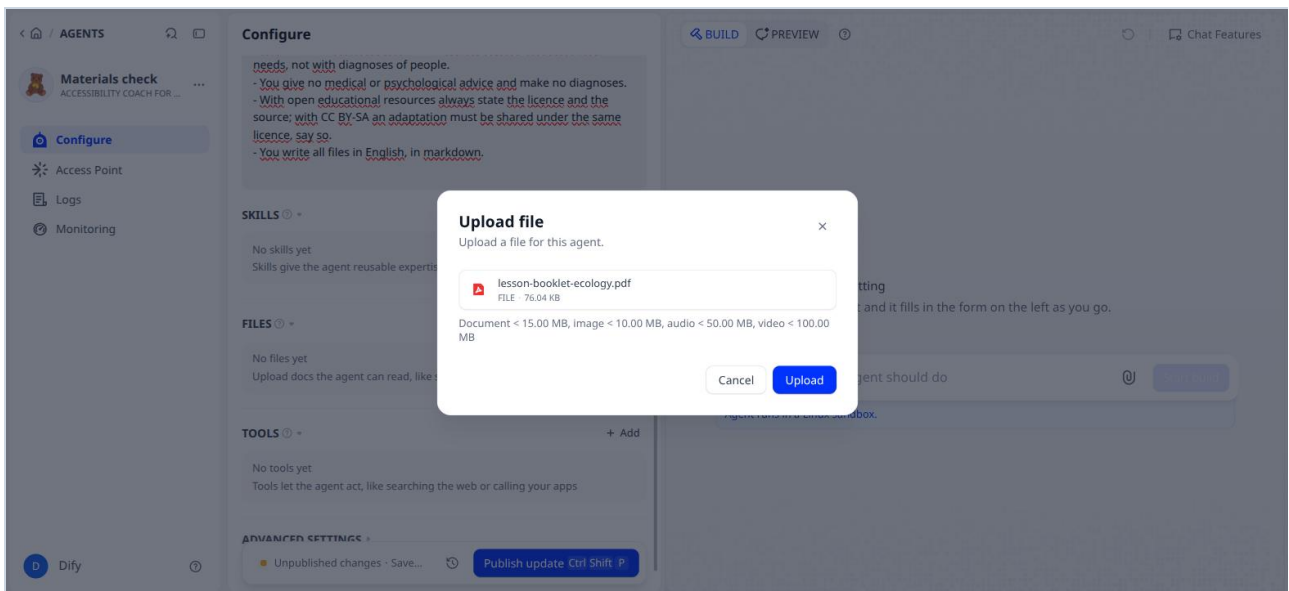
The instruction follows the structure of guide 4 (who, method, what you deliver, rules), with two additions. The seven assessment points in step 3 are the translation of the three UDL principles into things you can read off a text. And step 5 tells the agent exactly how to call the custom tool, including the query syntax; without that sentence it first tries wrong and only then right (see step 6).

Your choice: the support needs. The six needs in point 2 are a choice. Your support coordinator may have a different list, or want “long-term illness” or “deaf or hard of hearing” added. Every need you add must also get an answer in point 4: what changes in the text or in the lesson? Otherwise the agent names the need but does nothing with it.

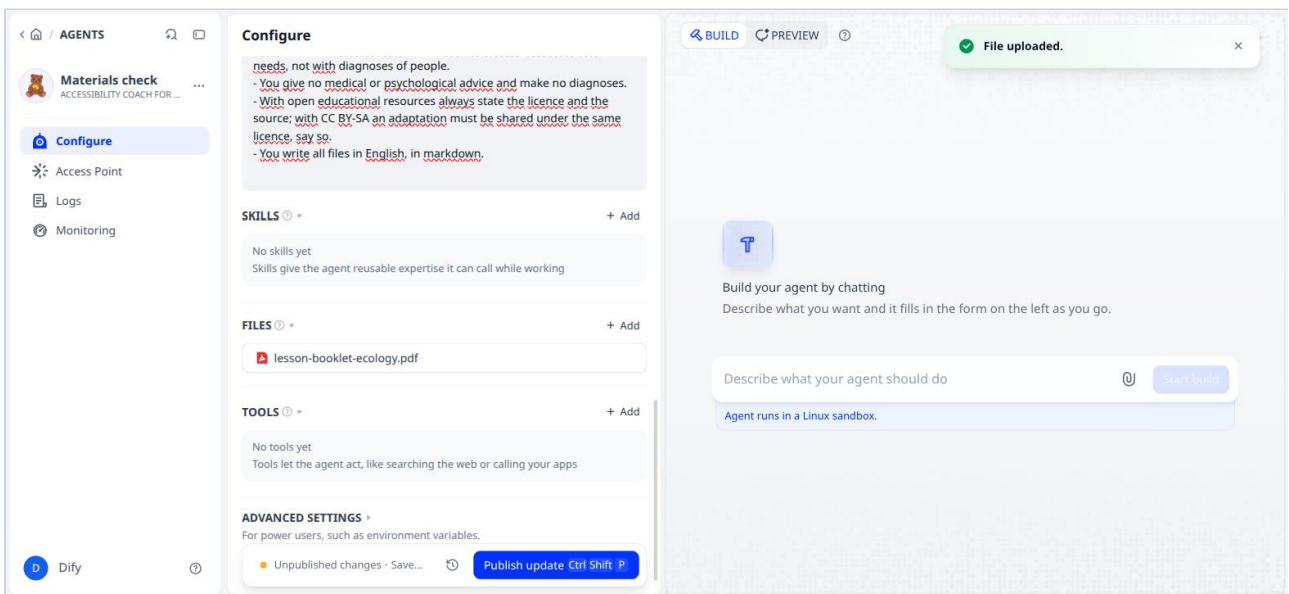
Your choice: what the student version does. “At most 15 words per sentence, glossary at the top, check question per section” are concrete rules you can verify. If you would rather have a version with pictograms, with a summary per section, or in another language for a newcomer: put it in just as concretely.

Step 5. Add the booklet and the tool

1. Under **FILES** click **Add**, choose `lesson-booklet-ecology.pdf` and click **Upload**.

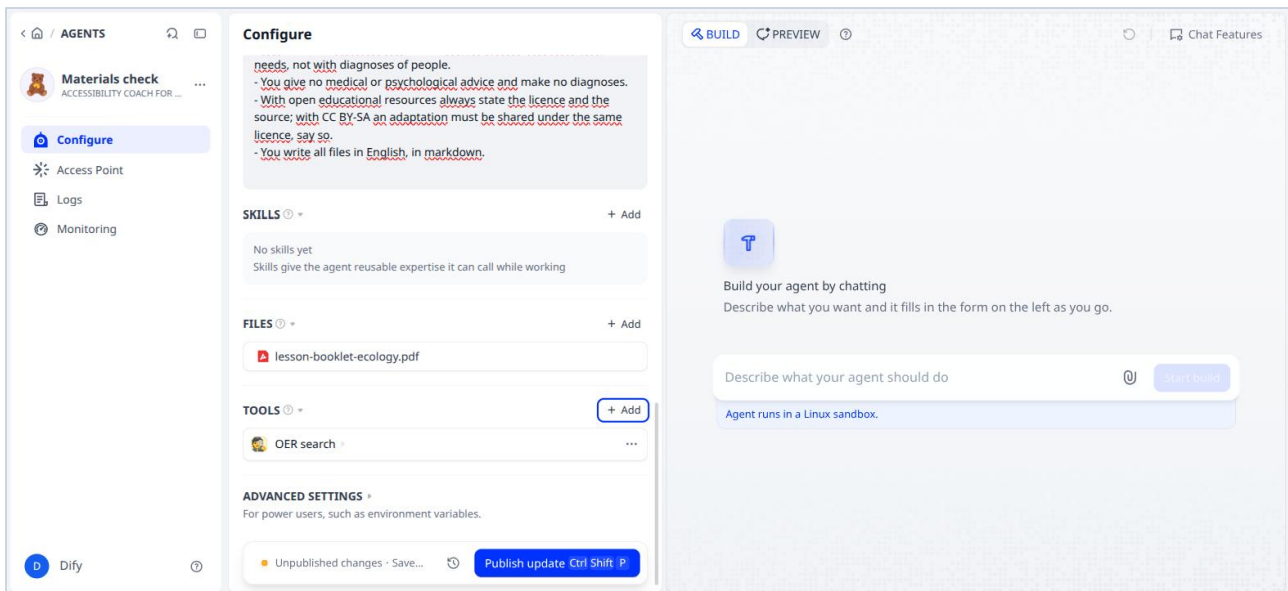


The booklet is ready to upload



The booklet is among the files

- Under **TOOLS** click **Add**, click the **Swagger API** tab, click **OER search** and then **Add all**. Close the picker with Escape. OER search is now under TOOLS.

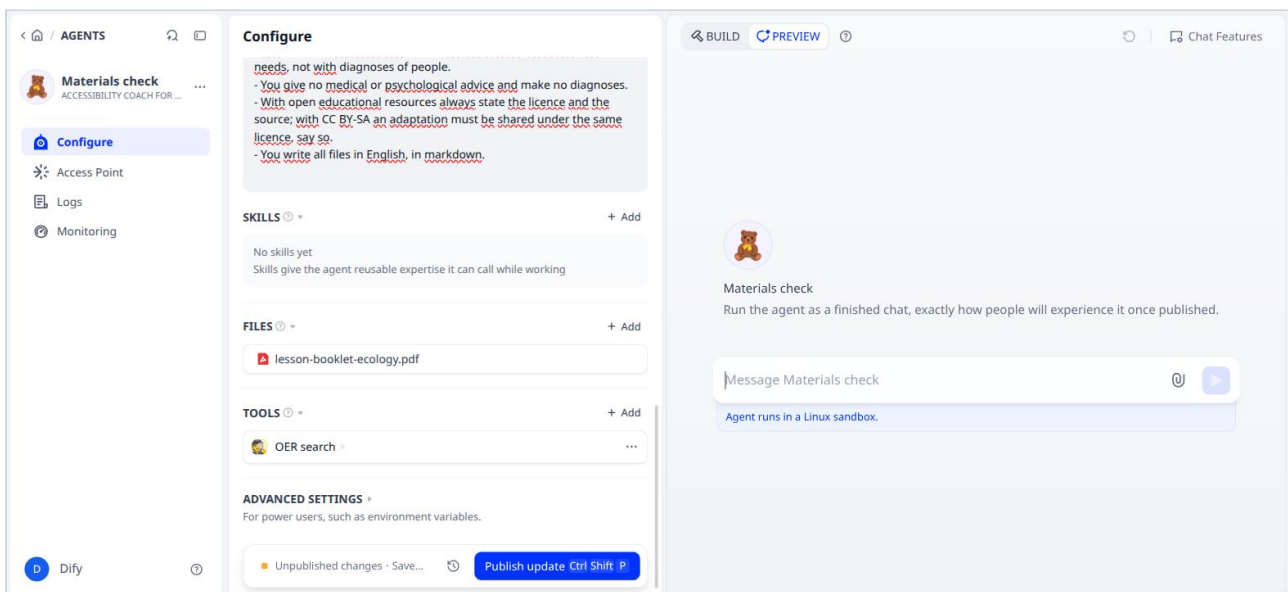


OER search under TOOLS

The booklet in FILES is the example for the test. In the webapp a teacher can later send their own booklet via the paperclip in the chat; the agent then uses that file and not the example (that is what point 1 of the instruction says).

Step 6. Test as a teacher

1. Click **PREVIEW** at the top right.

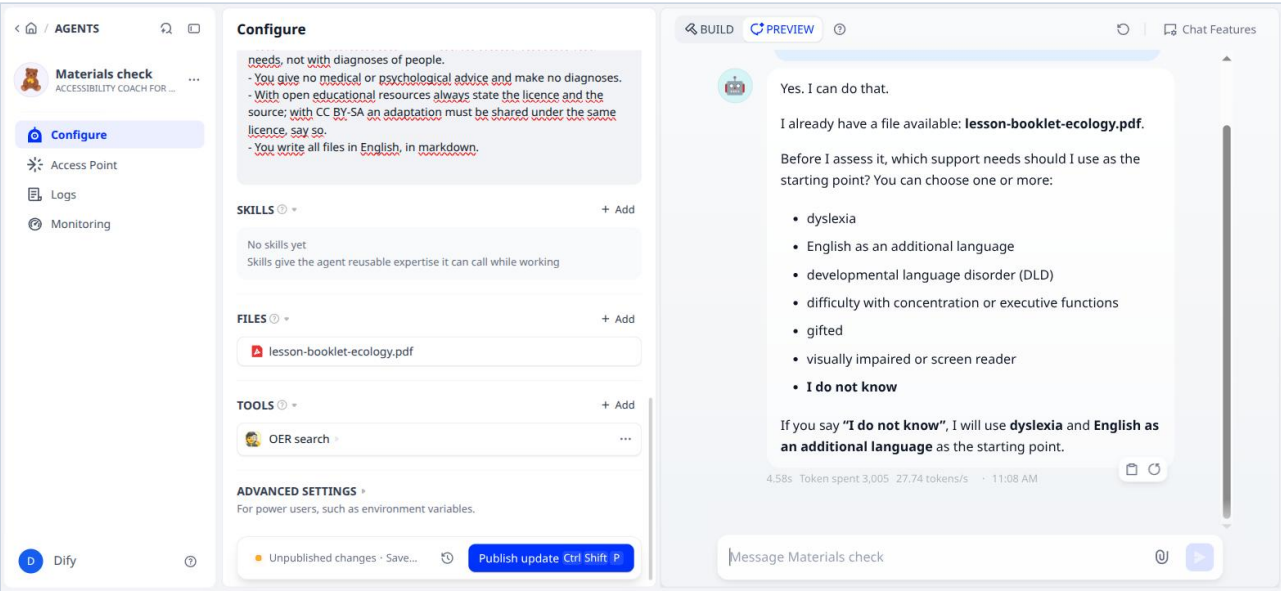


The PREVIEW tab

2. Type the first message.

Message	What you type
1	Hi, I teach biology in Year 9. Can you look at my lesson booklet on ecology?

The agent sees the booklet among its files and asks the question about support needs, with the six choices and “I do not know”.

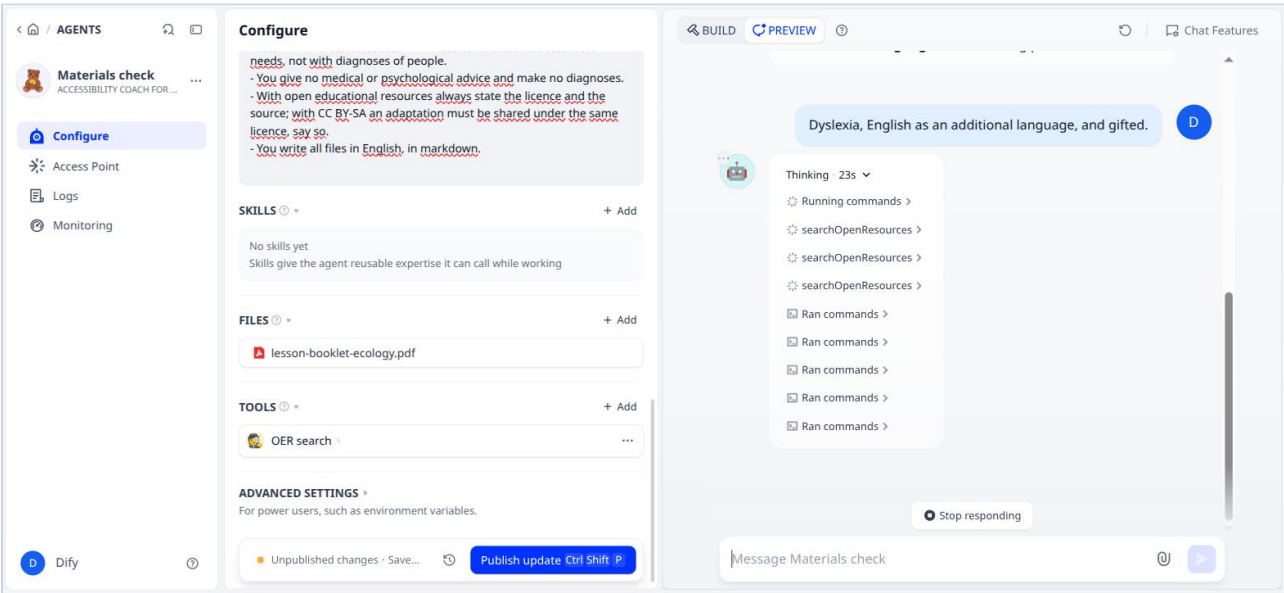


The first answer: the question about support needs

3. Choose three needs.

Message	What you type
2	Dyslexia, English as an additional language, and gifted.

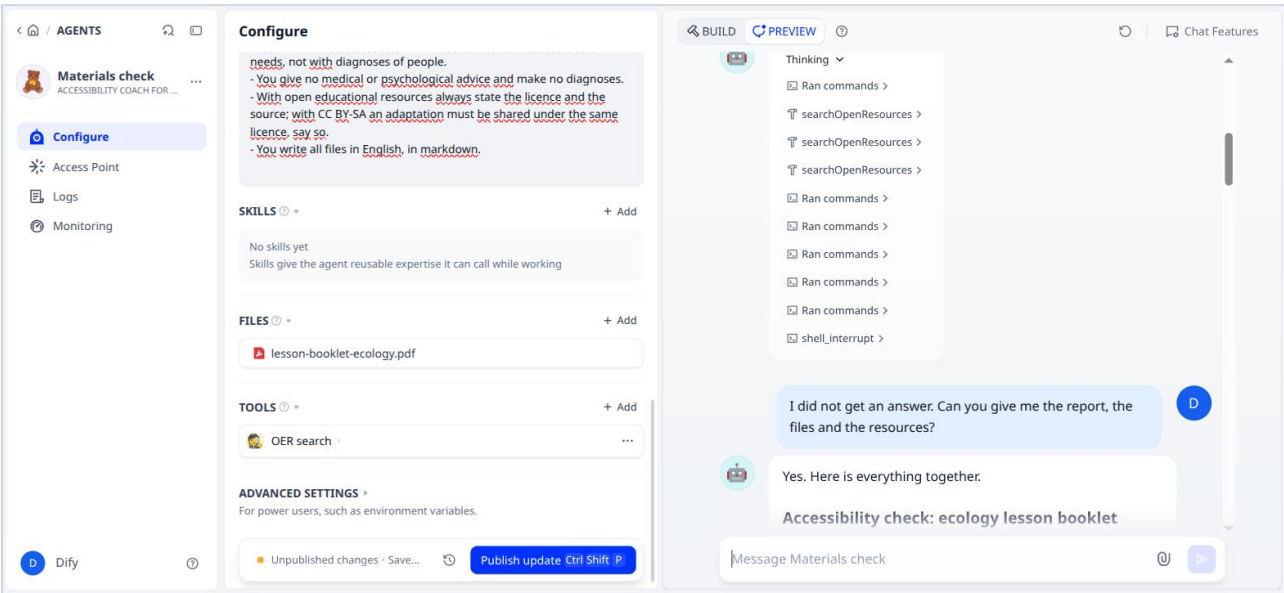
Now the agent gets to work, and that takes well over a minute. Under **Thinking** you see what it is doing: **Ran commands** (reading the booklet, writing the three files) and three times **searchOpenResources** (the custom tool). At the bottom it says **Stop responding** while it is busy.



The agent is busy: commands and searches

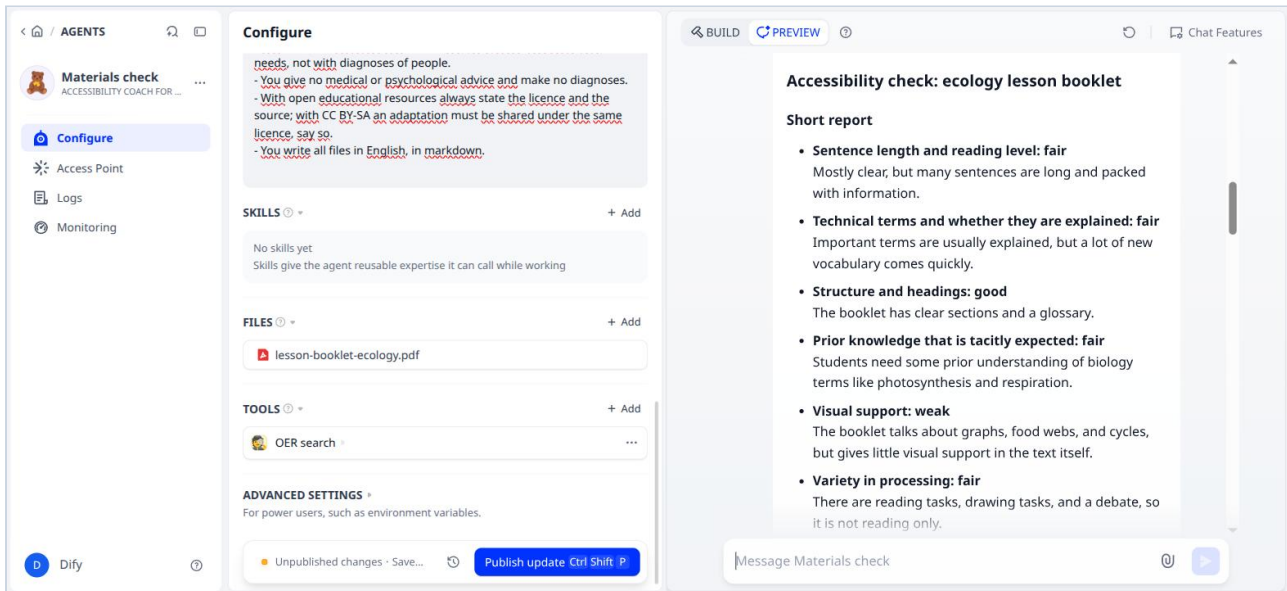
4. Wait until the answer is complete. In our test it was not: after two minutes the list of steps ended with **shell_interrupt** and no text followed. That is beta behaviour (see When things go wrong). The work was done, only the answer did not come. So we asked once more.

Message	What you type (only if no answer comes)
2a	I did not get an answer. Can you give me the report, the files and the resources?

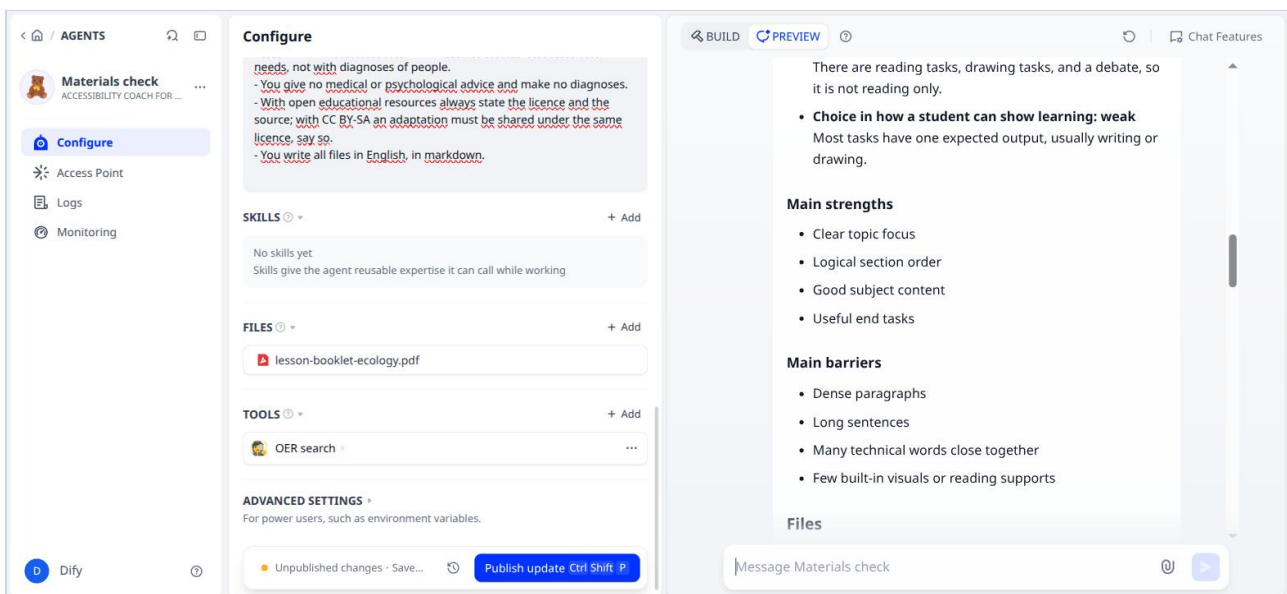


The interrupted turn and the follow-up question

5. Now the answer comes in thirteen seconds, because the files and the searches are already there. It starts with the **Short report**: seven points with a score and one sentence of explanation. For our booklet: sentence length fair, technical terms fair, structure good, prior knowledge fair, visual support weak, variety in processing fair, choice in how to show what you have learned weak. Then main strengths and main barriers.

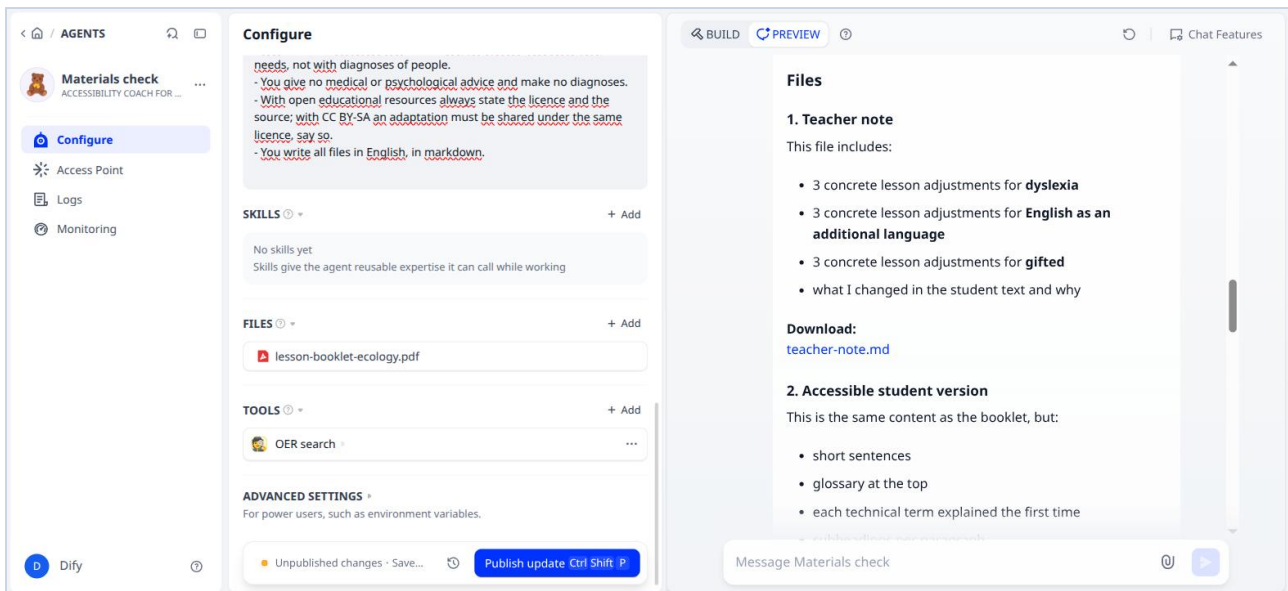


The report, first half

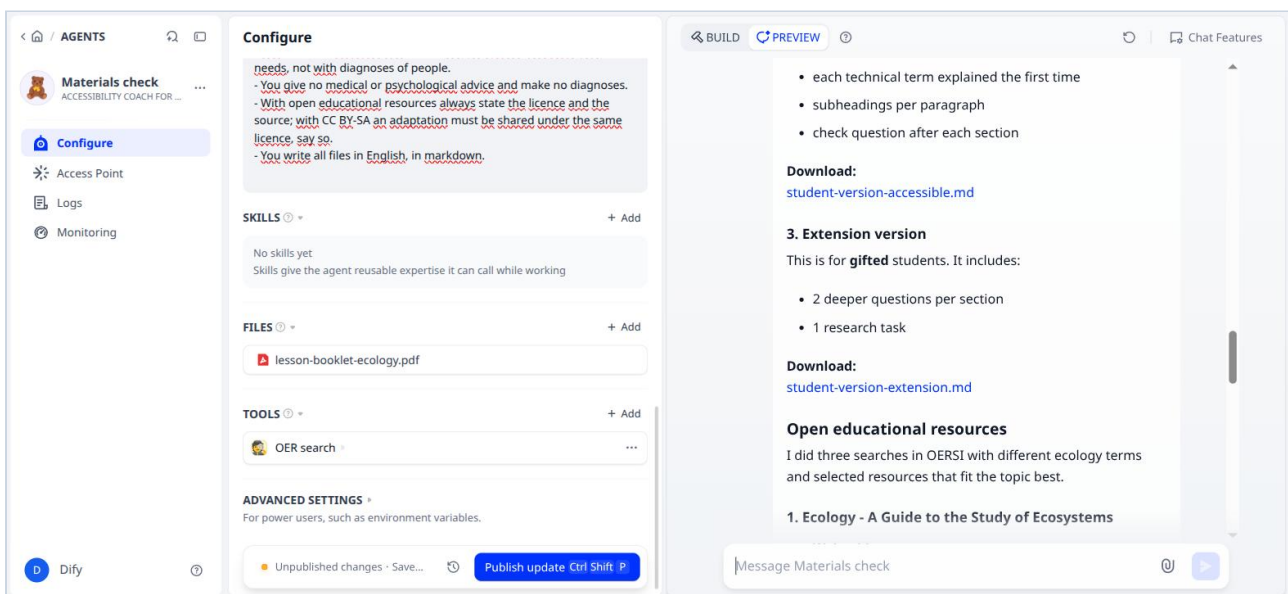


The report, second half

6. Below it are the **Files** with a download link each: teacher-note.md, student-version-accessible.md and, because gifted was chosen, student-version-extension.md.

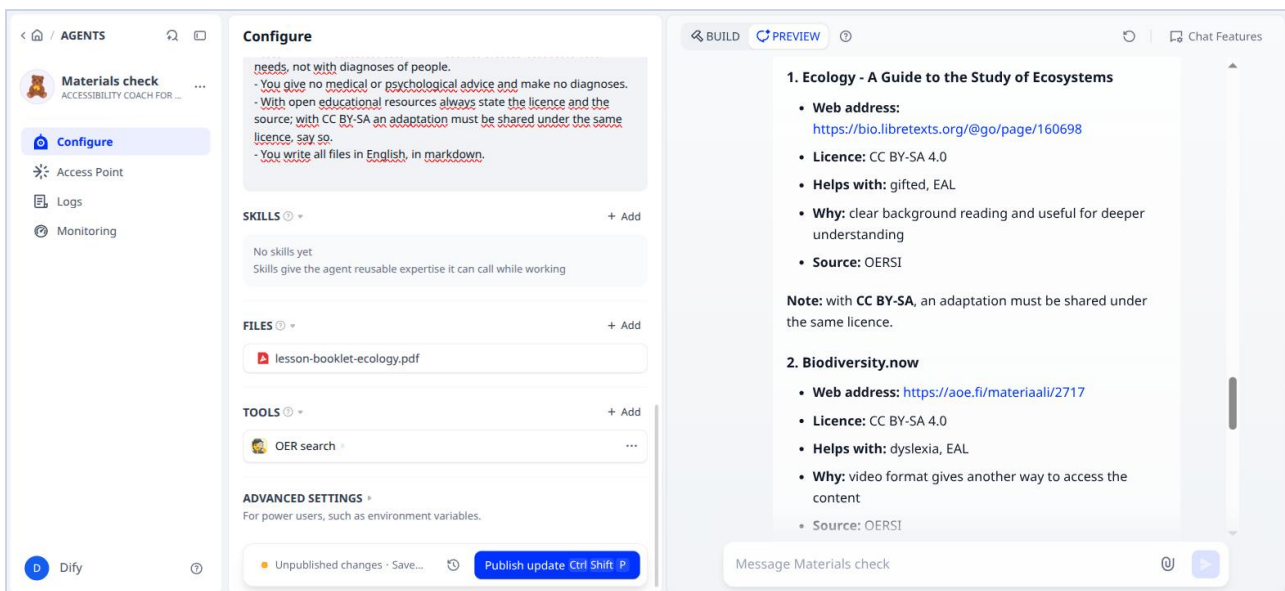


The teacher note

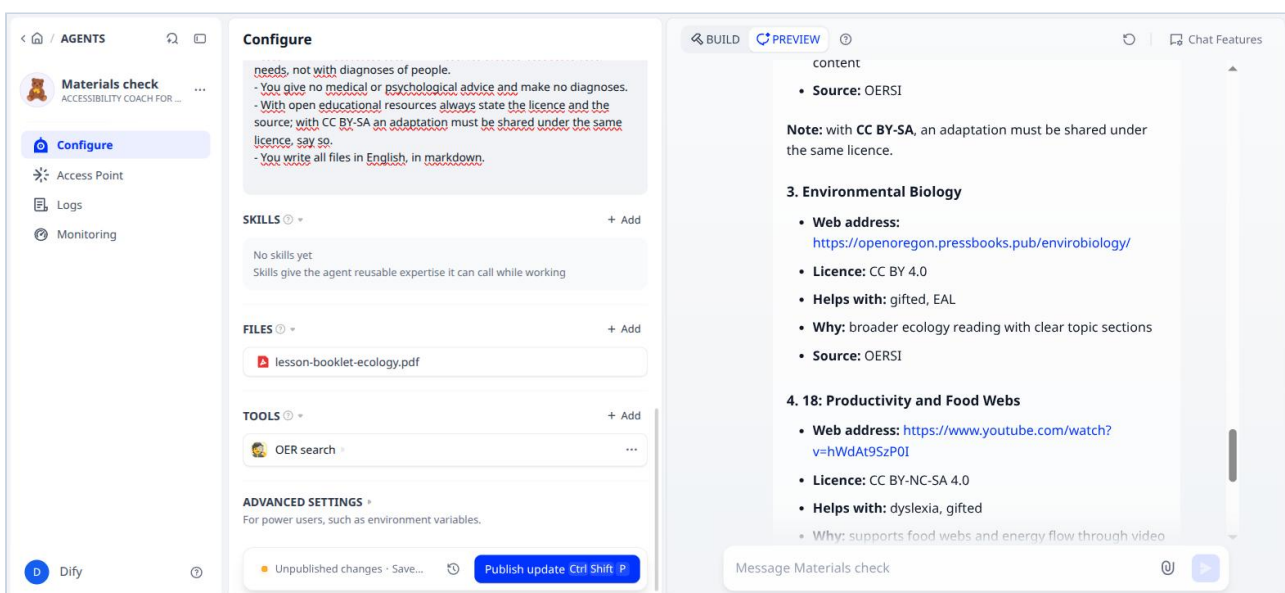


The two student versions

- Finally the **Open educational resources**: five resources with web address, licence, which need they help with and why. In our test two of them were CC BY-SA, each with the note that an adaptation must be shared under the same licence, one was CC BY-NC-SA (no commercial use) and two CC BY.

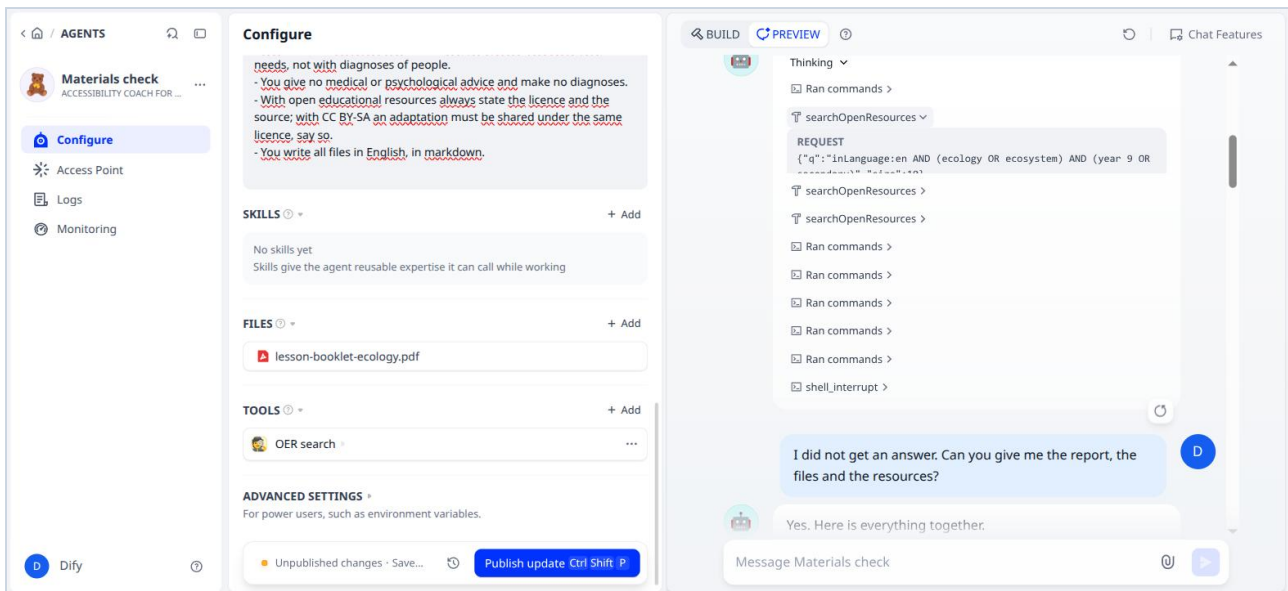


The resources found



More resources, with a video and an open textbook

8. Scroll back to the list of steps under **Thinking** and click the first **searchOpenResources**. You see the REQUEST: the query the agent formulated itself, `inLanguage: en AND (ecology OR ecosystem) AND (year 9 OR secondary)`, with size 10. In an earlier run the first search went wrong: the agent put loose words after each other, the index answered with an error, and the agent tried again with the right syntax. That is the behaviour of an agent: try, look, try again. This time the example in the instruction was enough.

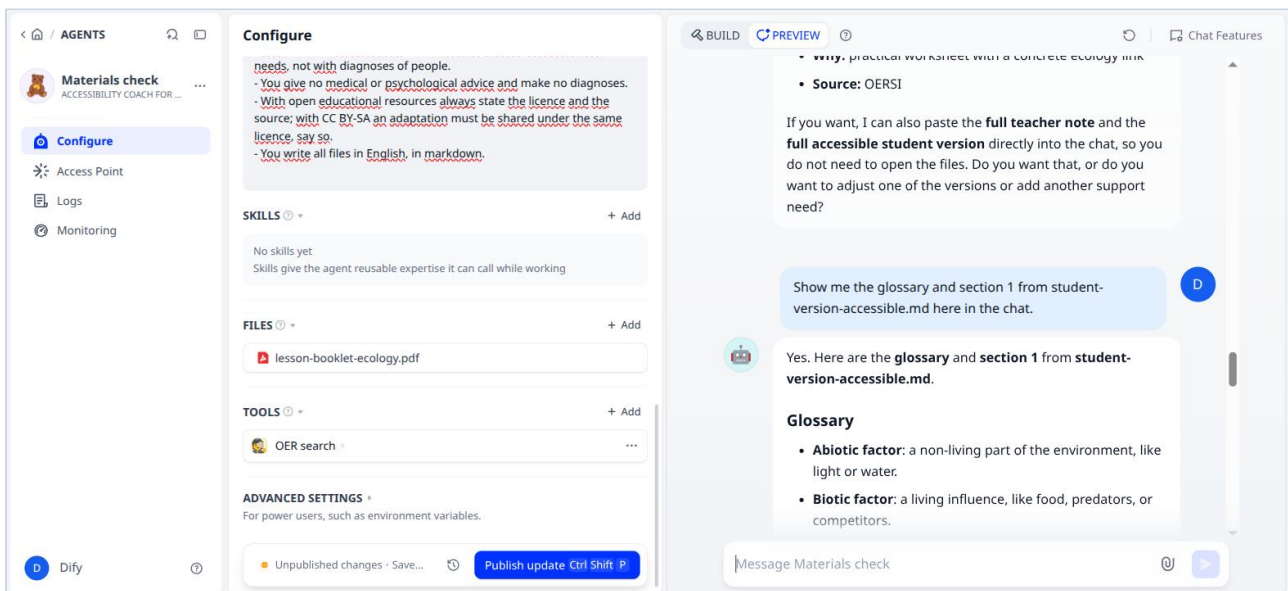


The first search, with the query the agent wrote

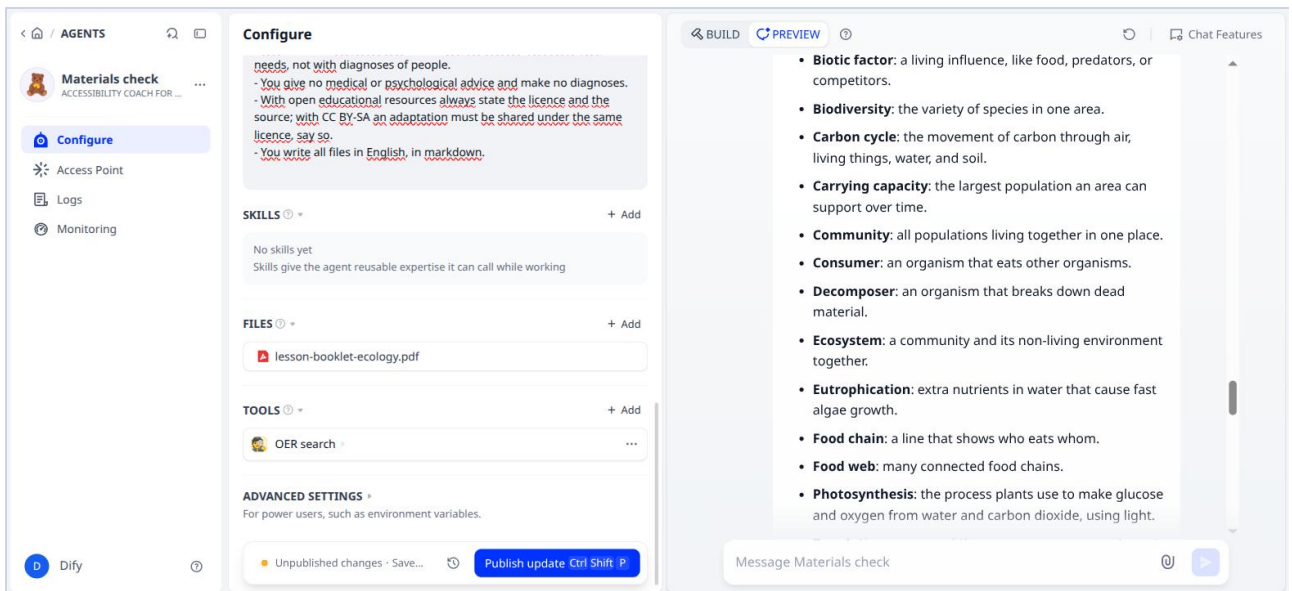
9. Ask to see a piece of the student version.

Message	What you type
3	Show me the glossary and section 1 from student-version-accessible.md here in the chat.

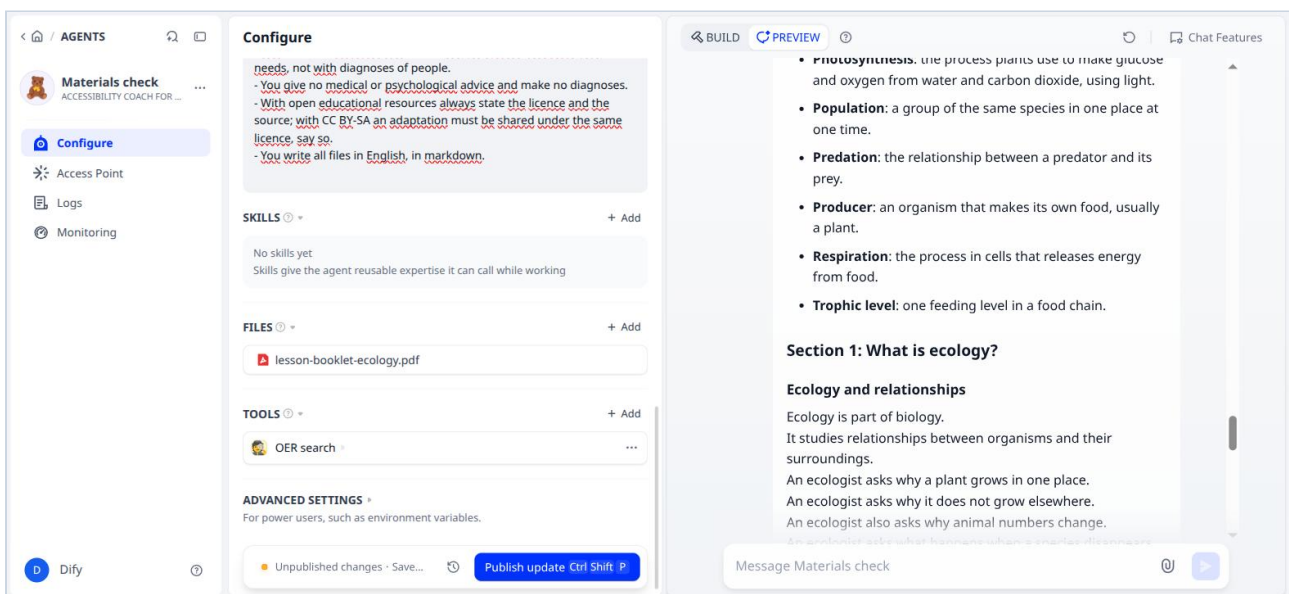
The glossary has eighteen terms, from abiotic factor to trophic level, each explained in one sentence. Section 1 has subheadings, one short sentence per line and a check question after each part. Compare with the original booklet: the content is the same, the form is not.



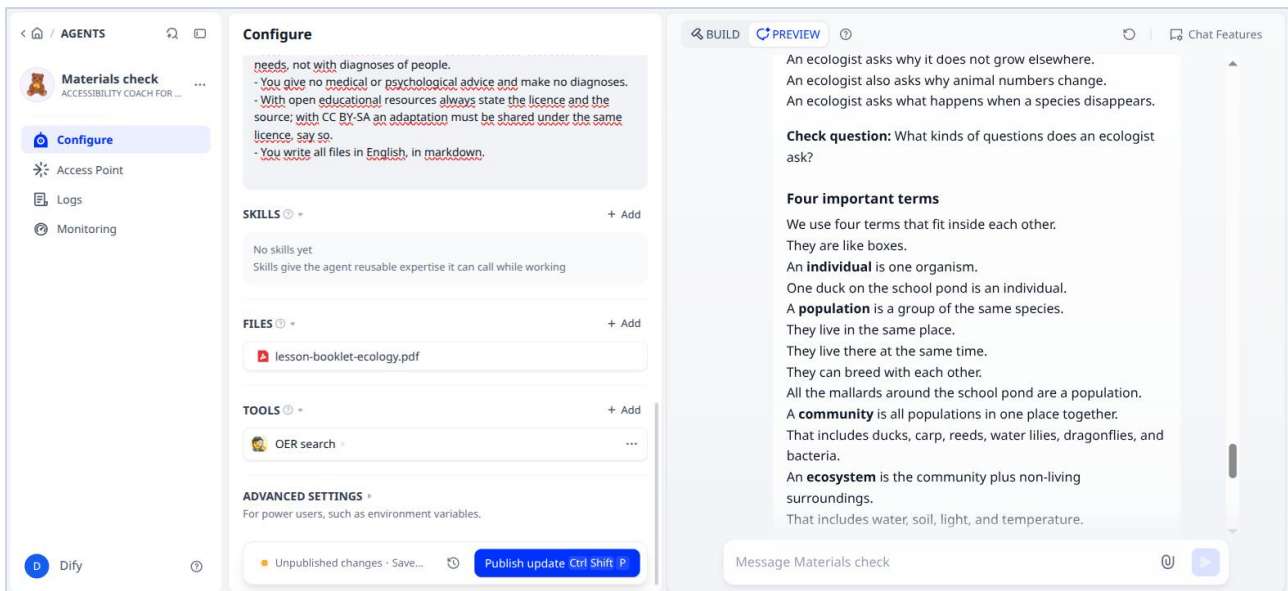
The glossary



The glossary, continued



Section 1 in the accessible version

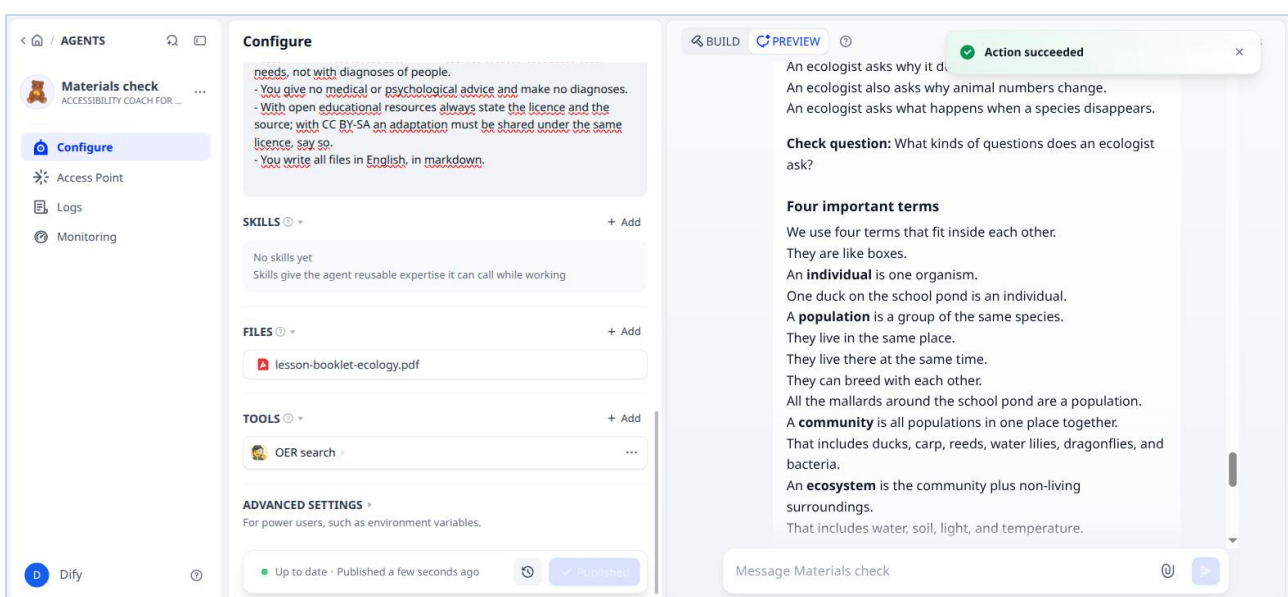


A check question after each part

The files sit in the agent's sandbox and can be downloaded through the links. They are markdown (.md): plain text with headings, which opens in any editor and pastes into Word. If you want a Word file straight away, put ".docx" instead of ".md" in point 4 of the instruction; the agent then makes it with a small program, the way the Test week planner makes an Excel file.

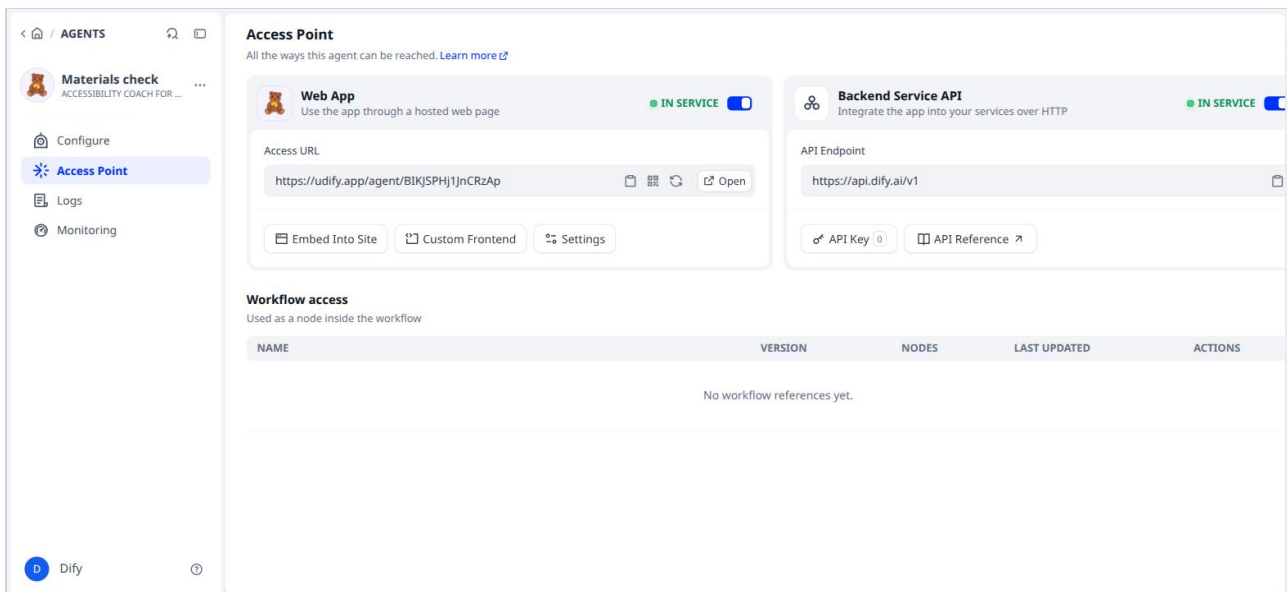
Step 7. Publish

1. Click **Publish update**. "Action succeeded" appears and the button turns grey with **Published**.



The agent is published

2. Click **Access Point** for the webapp link.



The Access Point screen with the Access URL

Whoever gets the link can send their own booklet via the paperclip and gets the same report, the same files and the same search results. Conversations through the webapp are in **Logs** (see guide 4, step 8).

Checking that it worked

- Under Integrations → Swagger API as Tool is OER search with one action, and the test gives hits.total.value greater than 0
 - Under the agent's TOOLS is OER search; under FILES is the booklet
 - The agent first asks about support needs and only then assesses
 - The report has seven points with a score
 - There are two files (three if gifted is chosen) and at most five resources with licence
 - The Access URL opens a working chat
-

When things go wrong

What you see	What is going on
When pasting the schema no tool appears under Available Tools	The JSON is not valid: a comma or brace too many or too few. Paste the file <code>oer-search.openapi.json</code> again
Test gives an error about the query	Words without AND or OR between them, or unbalanced brackets (step 3, orange box)
Test gives hits.total.value 0	The keyword does not occur, or <code>inLanguage:en</code> is misspelled
The agent names resources that are not in the RESPONSE	The model fills in from its own memory. The rule “invent no resources” is in the instruction; always check the links yourself
The agent makes no files, only text	The word “sandbox” is missing from point 4 of the instruction, or the model is not compatible (guide 4, step 2)
The student version leaves parts out	The sentence “Leave nothing out; simplify the form, not the content” is missing
The agent asks for names of students	The rule “You name no students and do not ask for names” is missing
The answer has not arrived after a minute	Normal for this agent: reading, writing three files and searching several times takes time. Wait until Stop responding is gone
The steps end with <code>shell_interrupt</code> and no text follows	Beta behaviour: the agent did the work but did not write the answer. Ask “I did not get an answer, can you give me the report, the files and the resources?”; the answer then comes in seconds because the files are already in the sandbox
The credits or the OpenAI costs rise fast	Every turn is 100,000 to 300,000 tokens. Test with one or two needs, not with all six

Make it yours

The booklet is invented and the six needs are an example. This chapter helps you rebuild the agent for your school and your subject. From easy to hard.

Change	Where	Difficulty	What you get for it
Your own lesson booklet as the example	Step 5, replace the file	easy	A demo with material colleagues recognise
Other support needs	Step 4, points 2 and 4 of the prompt	easy	A list that matches your support structure
Files as .docx instead of .md	Step 4, point 4	easy	Opens straight away in Word
Another language than English in the search	Step 4, point 5 (<code>inLanguage:de</code> , <code>inLanguage:fr</code>)	easy	Resources in the language of your school
A score per UDL principle instead of seven separate points	Step 4, point 3	medium	A report that matches the UDL language of your school
A fourth file: a parent letter about the adapted version	Step 4, point 4	medium	Communication included, in the same turn
Curriculum objectives from your national curriculum database	Step 3, a second custom tool (many curriculum bodies publish an API)	medium	The agent says which objectives the material covers and what is missing
The agent as a step in a workflow	Access point → Workflow access; or call the agent from a Chatflow	hard	An intake form before and a fixed layout after
Connection to the LMS or the student information system via MCP	Step 2, MCP tab, on a school platform	hard	Material straight from and to the place where it is used

Three assignments

1. **Find the flaw.** Have the agent assess your own lesson booklet and read the report critically: which score is wrong? Adjust the sentence for that point in the instruction until the report is right. That way you learn that the seven points are yours, not the model's.
2. **Check the resources.** Open the links the agent gave and judge them yourself for level and usefulness. How many would you have chosen? Add to the instruction what makes a resource useful for you.
3. **Make a second tool.** Describe another public API in a schema (a weather service, a statistics office, a dictionary) and add it. Let the agent use it and look under REQUEST and RESPONSE at what goes back and forth. After that you understand every tool you will ever meet.

Checklist before you use your own material

- [] The material you upload is your own or has a licence that allows adaptation
- [] There is no student data in the material or in the conversation
- [] Adapted versions are available to all students, not only to those with a formal diagnosis
- [] For resources under CC BY-SA you share your adaptation under the same licence, with attribution
- [] Someone reads the student version before it goes into the classroom; the agent sometimes simplifies just a little too much
- [] There is one owner of the agent and of the tool

Housekeeping

What Dify keeps. Conversations through the webapp are in Logs, including the uploaded material and the files created. For learning material that is no problem; it is a reason not to put student data in the chat.

Costs. This is the most expensive agent of the series: 100,000 to 300,000 tokens per turn, 30 cents to a euro through your own key. A department that has ten booklets assessed spends a few euros. Put a monthly limit on the key and do not let the agent circulate as an open link.

The custom tool. OERSI is a public service without a key. If the search interface changes, the tool fails; you notice that by a RESPONSE with an error message instead of JSON. The schema in `oer-search.openapi.json` is then the only thing you need to change.

Curriculum objectives. Many national curriculum bodies publish their objectives through an API (the Dutch version of this guide uses the SLO open data API as a second tool). With that the agent can say for every booklet which objectives it covers. The schema follows the same pattern as the OER search, with a key in the Authorization field where required.

From prototype to your school's platform. Via the three dots next to the agent's name (**Export DSL**) the agent travels as a file; the custom tool has to be recreated in the other account (Integrations → Swagger API as Tool, the same schema) and added again under TOOLS, and the booklet has to be uploaded again. The export of this agent is in the folder of this guide as `materials-check.dify.yml`.

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

