


# Image Analysis at Scale with Generative AI

## GenAI Challenge: Showcase your best practices

**Other Publication****Author(s):**

Zhang, Hongyuan; [Garcia Otalora, Monica Andrea](#) 

**Publication date:**

2025

**Permanent link:**

<https://doi.org/https://doi.org/10.3929/ethz-b-000742167>

**Rights / license:**

[Creative Commons Attribution-NonCommercial 4.0 International](#)

# Image Analysis at Scale with Generative AI

Hongyuan Zhang and Dr. Monica Andrea Garcia Otalora

Plant Ecological Genetics Group, D-USYS, ETH Zurich

Contact: [hongyuan.zhang@usys.ethz.ch](mailto:hongyuan.zhang@usys.ethz.ch)

## Context and Challenges

At ETH Herbaria, the fungal collection curators are working to digitize the historical specimens. Each specimen is accompanied by a label containing essential information relevant to biodiversity research. However, most of these labels have yet to be digitized or cataloged, as the process of transcribing the printed information into digital format is both tedious and time-consuming (See Figure 1).

With powerful genAI models such as ChatGPT and Claude, researchers could easily achieve the desired optical character recognition (OCR) output with text prompts in web browsers. However, one would encounter problems while trying to process a batch of images at once, for example, "Claude hit the max length of this message...". The restricted context windows in the web browser not only limit the number of uploaded images to a few but also make the model gradually forget the instructions, leading to wrong outputs. **Here, we provide a solution for batch image OCR with our code available on GitHub.**

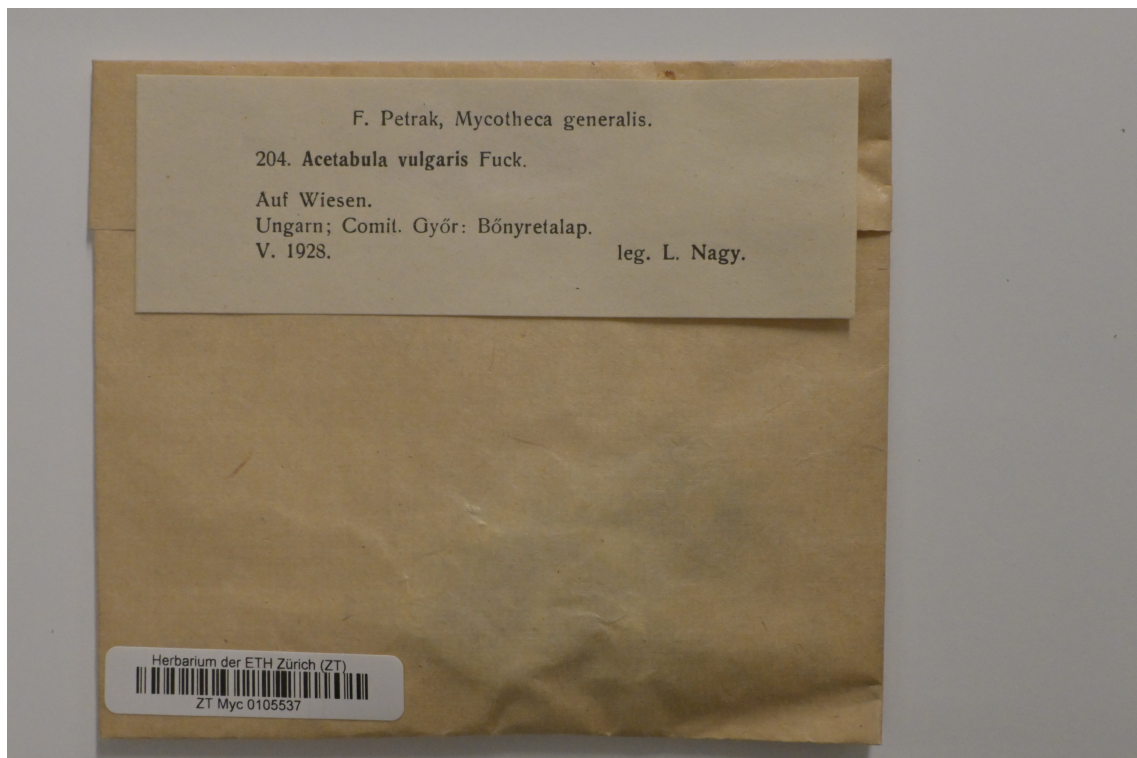


Figure 1: An example fungi sample to be digitized.

## Tools

As the sample labels are well structured, it is reasonable to set an AI workflow to ensure reproducible outputs at scale. For example, we could simply define a workflow in dify (see Figure 2). This workflow enables users to upload sample images, and the first model will conduct OCR and store the results in a structured JSON file. The second model can then convert this data into an Excel output. However, AI workflow tools such as dify currently do not support vision tasks well. It only allows a maximum of 10 images to be uploaded per batch and does not support state-of-the-art vision models. Therefore, we will adhere to the workflow logic but utilize other tools for the OCR.

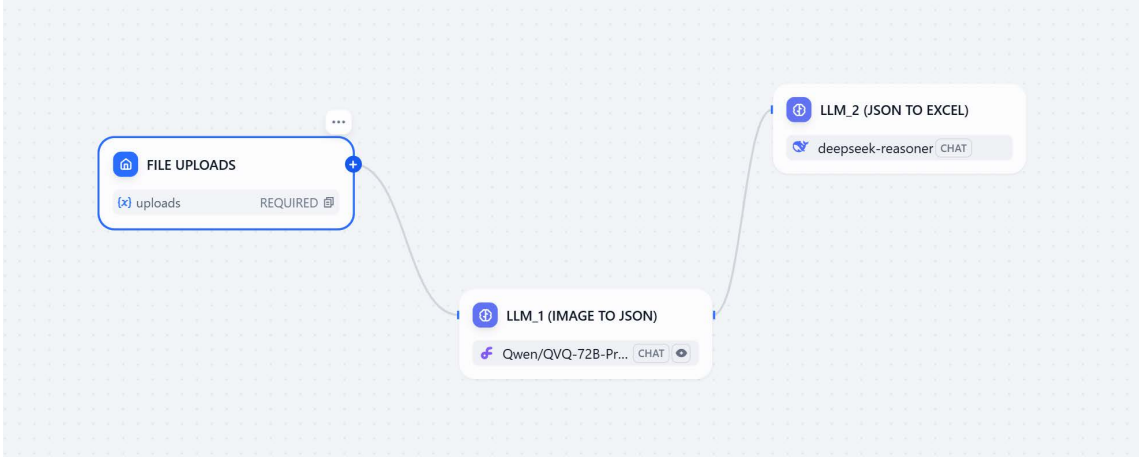


Figure 2: An OCR AI workflow defined in Dify

When we chat with large language models (LLMs) in a web browser, it is allowed to upload images. However, most LLMs, such as DeepSeek-R1, cannot directly handle images themselves but instead request external tools (agents) to do so, which is known as functional calling. Meanwhile, there are also multiple modality models, such as GPT-4o, that could process both images and text. Therefore, to batch process images, we need to either use agents or models with vision capacity via API calls. With the recent development of model context protocol (MCP), interacting with agents will be much easier in the future, but still not yet. Hence, we demonstrate the API call approach here.

## Methods and Results

According to OpenAI docs, processing images with a vision model requires:

- API keys. A paid authentication code to get results from OpenAI models.
- Model names. We need to select a model that can perform OCR on images.
- Prompts. The system (developer) prompt tells the model how to behave generally. The user prompts the model to generate the desired outputs.
- Output formats (optional). Define the desired output JSON schema structure.

With the assistance of Cursor, we can easily implement the key components above into code. Essentially, we defined a Python class called `FungariumOCR` to encompass all the functions required for our OCR tasks. It requires OpenAI API keys `openai_apikey` to perform OCR over a single image with the function `visison_model_ocr`, which also takes arguments of our key components. Once we defined the directory of images `input_dir`, we can loop the OCR function over all images with the function `batch_ocr`. The final results are structured JSON files, which can then be checked and recorded into the Natural Science Collections online platform (NAHIMA). Some outputs examples look like this:

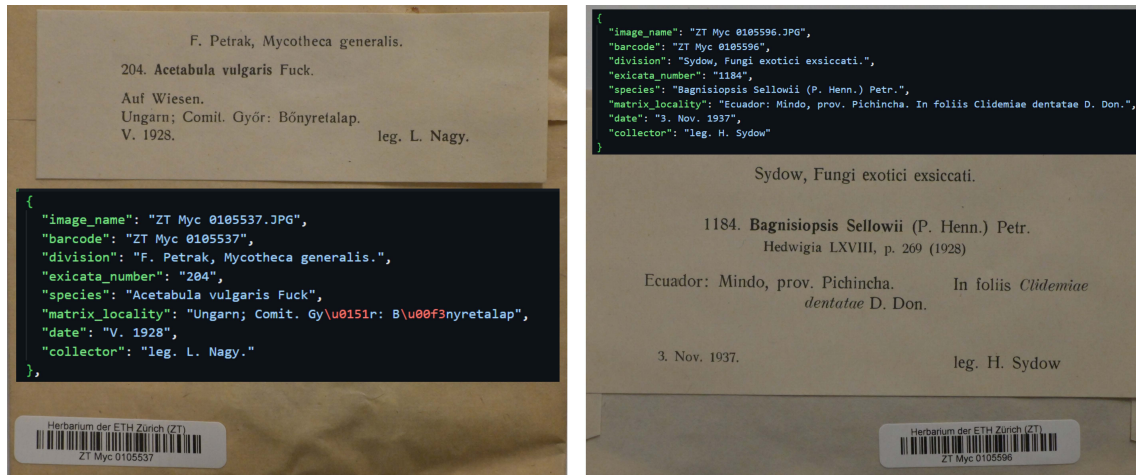


Figure 3: OCR examples. Original images with their results in JSON format

**Prompts and structured output JSON schema are the keys to achieving such results.** For prompts, we need to specify the OCR context, desired input and output structure, and format, which is the **system prompt**:

The goal is to extract structured text from Fungi sample images. The rules are:

1. Each image contains two sections of text chunks. One is the bar code, and the other is the sample information.
2. The languages are only in English or German.
3. The bar code chunk starts with "Herbarium der ETH Zurich (ZT)" followed by a barcode with text such as "ZT Myc 0105537". The task here is to extract the barcode text.
4. The sample information chunk started with a division separator, such as "Dr. F. Petrak, Mycotheca generalis."

This will become the value for the division column. After the division separator, there are other structures defined by the following:

5. Exicata Number and Species: Lines. For example, "204. Acetabula vulgaris Fuck", contains two pieces of information: \* Exicata number the number before the period (e.g., 204) \* Species name everything after the period (e.g., Acetabula vulgaris Fuck.)
6. Matrix and Locality: A line that holds the information (e.g., Ungarn; Comit. Gyor: Bonyretalap). If a sample is missing a Matrix and Locality line, leave it blank. Extract as it is, no more added information
7. Date: A line that has a Roman numeral month plus year (e.g., V.1920, X.1924, XII.1924), from which you split out: \* Month Roman numeral (e.g., V, X, XII) \* Year numeric year (e.g., 1920, 1924). Note, sometimes the month can be a normal English month with abbreviations
8. Collector: A line beginning with leg. indicates the collector (e.g., leg. J. Cogolludo.)
9. Image name: I will define later

The output should be a JSON like structured dictionary with keys (image\_name, barcode, division, excata\_number, species, matrix\_locality, date, collector).

Remove unnecessary '\n' etc. Dont output anything else.

Next, we define the **user prompt** to specify what to do. Through trial and error, the key point here is to instruct the model to use its own vision capacity for the OCR. Otherwise, the model will generate Python scripts for traditional OCR, resulting in inaccurate results.

Directly extract information with your own vision capabilities, not Python packages such as pytesseract

Furthermore, we need to define a strict and structured output format to ensure consistent outputs, following the corresponding documentation:

```
from pydantic import BaseModel

class OutputFormat(BaseModel):
    image_name: str
    barcode: str
    division: str
    exicata_number: str
    species: str
    matrix_locality: str
    date: str
    collector: str
```

Finally, we still need to manually verify the model output. The results are pretty accurate in our case. Now, it's your turn to try it out and adapt it to your specific needs!

## Code availability

<https://github.com/Alias-z/FungariumOCR>.