Running models locally

There's lots of generative AI models you can run locally, instead of using online services. Here's a couple of options!

- Running generative models locally
- Running Large Language Models locally
- Experiment with Dutch local LLMs

Running generative models locally

All the big models run on external servers and are usually only available through a (paid) account. There are some alternatives available that you can run locally on your own machine. Installing these usually involves complex installation procedures, but there's a trend for 'one-click-installers' that get you set up relatively painlessly. Below you can find some simple installers for various generative AI's.

> **Note 1 All of these models take up significant amount of space on your computer.** The programs can take up to 3Gb, and the models are even larger. Make sure you have around 30Gb free once you get these models running!

> **Note 2 Most of these models need (recent and beefy) Nvidia graphics cards to run, or Apple M1/M2.**

> If you don't have a system that can run these models and you also don't want to use the online services, or need some help with installation, **please visit us in Oudenoord ON0.50**. We have some models set up here that you can experiment with. Open for both students and employees (of HKU.)

Image generation on your own computer

These models are based on Stable Diffusion. You will not get the latest version, but you can re-train the model, or download variants from the internet. Automatic11111 also allows you to combine models.

Stable Diffusion WebUI by Automatic11111: https://github.com/AUTOMATIC1111/stable-diffusion-webui (Win, Linux, Mac)

Easy Diffusion https://github.com/easydiffusion/easydiffusion (Win, Linux, Mac)

Both programs above can be downloaded from the Github page directly, resulting in a folder containing a .bat file. Run this and it will start downloading and installing all necessary files. Once it's done, run the .bat again and the program will start in your browser.

DiffusionBee https://diffusionbee.com/ for Mac. Also runs on older Intel macs, but those will take a very long time to generate images.

Text generation ('ChatGPT') on your own computer

If you want more control over the installation, pick between various models and give the model your own instructions, check the Ollama bookstack description page here.

If you want a no-fuss installation with a web interface, you can try something like oobabooga: https://github.com/oobabooga/text-generation-webui (Win, Linux, Mac)

Running Large Language Models locally

Ollama is currently a popular options for running LLMs locally. With the newer versions you can download other models than llama too, like Google's Gemma or task-specific smaller models. You can download Ollama from ollama.com, available for all systems. Again: you need a bit of a beefy computer for this, preferrably with a recent NVIDIA graphics card and quite a bit of storage. Once installed, Ollama disappears to the background. On Windows, you can still see it running among the icons in the lower right of the taskbar. On Mac, there will be an icon at the top of your screen.

When installed, open a terminal (win key, type cmd) and run ollama from here.

Ollama starter tips

No idea where to start? Type

```
ollama -h
```

See all installed models by typing

```
ollama list
```

Add a model with the following command. Replace [model name] with one of the models you can find in the ollama model list.

```
ollama run [model name]
```

All models (and instructions, see below) are saved as blob files in C:\Users\[Username]\.ollama\models\blobs. This means you can't manually remove or edit models. To delete a model, type

```
ollama rm [model name]
```

Creating Characters: build your own instructionset

One of the ways you can modify the model is to give it additional instructions before it runs conversation mode. This way you can give a model character. You can instruct it to talk a certain way, use specific kind of vocabulary or express itself in a different way. This can improve the responses you get from the model, but can also be used to make interesting interactions.

> Please note that this is **not the same as training your own model,** just an additional set of instructions to a pre-trained model. The names ollama gave to this process are a bit confusing, the modelfile they mention here is an instructionset to an existing model.

The way to do this is to copy the instructions of an existing model (like llama3:8b), modify those instructions, and then create a new model in ollama using those new instructions. For a long description of this process, see here. The short version:

Step 1: copy a model file

You can make a copy of an existing model in Ollama by using the following command:

```
ollama show [modelname] --modelfile > [newname]
```

where [modelname] is one of the models you have already installed (e.g. llama3:8b), and [newname] is the filename of the new instructionset you want to create (e.g. myfirstmodel). **Where does it save the new file?** In the folder that you are currently in while typing the command! On Windows, when opening a terminal this will be C:\Users\[Username] by default. In order to keep everything in one place it's a good idea to navigate to the folder where you want your workfiles to be before running these commands.

We also have two files prepared for you here: story and emo. These use the llama3:8b model, which will download automatically when you install and run this 'story' or 'emo' model (see below).

Step 2: modify the model file

Open the newly saved model file in a text editor. There's lots of things you can edit here, see the full description on the ollama page. If you only want to change the character of the AI that you are talking with, add a descriptor at "system Job Description:". In the 'story' file, the assistant will write all responses as a short story. With the 'emo' file, the assistant will reply only in one word. You can see this in the model file if you open it in a text editor.

Step 3: install your model

To install your modified model file, type:

```
ollama create [name] -f [file location]
```

Under Windows, this often will give an error along the lines: "1 argument expected, 4 given". If that's the case, make sure you're command line is navigated to the folder where the modified model is located, and then use .\[filename] as location. So when you want to save the story file you just edited as a model called 'story', navigate to the folder where you have the file 'story' and type:

```
ollama create story -f .\story
```

To check: when successful, your new model should now show up when you type: ollama list.

All modified models will still use the same pre-trained model file. If all your modified models are based on the llama3:8b model, it will only download that model once (the *list* command shows all of them being 5 GB, but that is not the size on your disk, just the size it will use in your memory).

Step 4: run your model

```
ollama run [modelname]
```

Step 5: edit your model?

Once installed, **you cannot edit a model**.

> You can edit the model file you saved locally, but **this will NOT update the model in ollama**.

To update a model, re-do all steps above: change the local file, remove the old model, create the new model.

Creating a chain of models

If you want to have models building on each other's outputs, or models talking to each other, you can chain your characters by using python scripts. For instance, using the 'story' and 'emo' models above, you can chain these together using Python and the following script:

```
import ollama


def get_response(model, message):
    response = ollama.chat(model=model, messages=[
      {
```

```
        'role': 'user',
        'content': message,
      },
    ])
    return response['message']['content']

def chain_models():
    inputIntoFirst = 'The summary of the day'
    # First, get the response from the 'emo' model
    emo_response = get_response('emo', inputIntoFirst)
    print("The 1st response was: ", emo_response)
    # Then, use the response from the 'emo' model as input to the 'story' model
    story_response = get_response('story', emo_response)
    # Now you can use 'story_response' as you wish
    print("The response was: ", story_response)

chain_models()
```

(to use ollama in Python, use pip install ollama. You will probably have to re-install your models if you ran ollama from the command line before)

Using an interface

There are lots of ways of adding an interface to your LLM, like LM Studio, or Open WebUI. These change all the time so there's not much use of making a list of them here. The interfaces usually allow you to make Characters (like above) directly in the browser, to add (text) files as input or to add additional local models such as Stable Diffusion for image generation. LM Studio makes it even more user friendly by allowing to to manage and run models within the UI directly.

Experiment with Dutch local LLMs

MacWhisper (Mac only) is a great local transcription tool that converts audio to text. It also has an option to add LLMs through Ollama so that you can summarize your transcripts, which would make a great complete suite for recording and summarizing meetings running only local models. For English this seems to work rather well, but local LLMs are known for not being all to reliable for Dutch. So we ran an experiment, summarizing transcripts of the HKU en AI podcast.

> **Conclusion**: No success yet for summarizing Dutch texts, in Dutch. (May 2025)

Setup

Setting up LLMs in MacWhisper is quite straightforward: install Ollama and install your model. Next, open MacWhisper and go to Global (the settings menu), then AI, Services. If Ollama is running, you should be able to select all installed models in MacWhisper by clicking Ollama under Add another service. Now once you've made a transcript you can interact with ollama under the AI tab (three stars) at the top right.

Testing Dutch in MacWhisper (May '25)

Interacting with any model through MacWhisper in Dutch gives strange results. Replies are often in English, or seem to ignore the prompt completely.

- Gemma3 and Mistral give pretty accurate summaries, but in English only. Interestingly it does seem to understand the Dutch contents of the transcript (although it misses some key points as well).
- Deepseek goes completely off the rails
- Llama3.2 gives a very short summary that misses key points.

Changing prompts or moving to chat mode does not seem to improve anything.

Testing Dutch in LLM directly (May '25)

I thought MacWhisper might be interfering in some way (as I could not get the LLM to react to anything else than 'summarize'), so I moved to Open WebUI. In this way I could still interact with the LLM and add a text file as imput. The textfile was the transcript export from MacWhisper.

While did this improve the interaction as I could talk to the LLM directly, results were similar to above. Some additional models tested here:

- Granite3.2 gave mixed bulletpoints, some accurate, some wildly off. Granite did respond in Dutch.
- Phi3,5 and Phi4 had nonsense results, although in Dutch.
- Two Dutch LLM models Geitje-7b en Fietje-2b completely derailed. They did not answer any questions but went rambling about daycare for young children, paper crafting and Dutch politics. It's clear what these models were trained on...