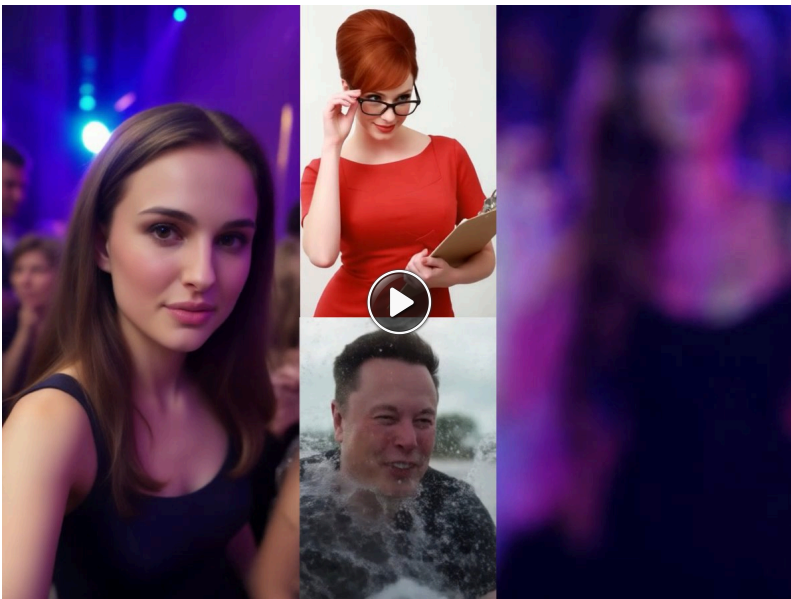


How to Train and Use Hunyuan Video LoRA Models

Originally published at <https://www.unite.ai/how-to-train-and-use-hunyuan-video-lora-models/>



This article will show you how to install and use Windows-based software that can train [Hunyuan video LoRA models](#), allowing the user to generate custom personalities in the Hunyuan Video foundation model:



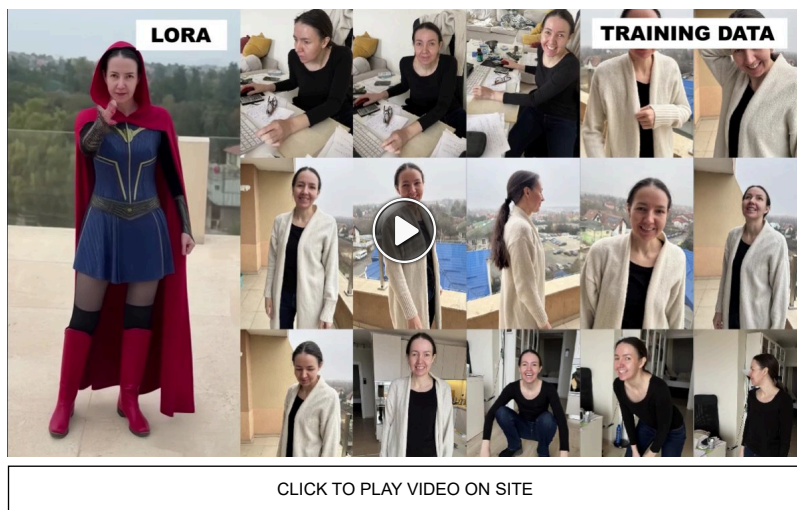
CLICK TO PLAY VIDEO ON SITE

Click to play. Examples from the recent explosion of celebrity Hunyuan LoRAs from the civit.ai community.

At the moment the two most popular ways of generating Hunyuan LoRA models locally are:

- 1) The [diffusion-pipe-ui Docker-based framework](#), which relies on [Windows Subsystem for Linux](#) (WSL) to handle some of the processes.
- 2) [Musubi Tuner](#), a new addition to the popular [Kohya ss](#) diffusion training architecture. Musubi Tuner does not require Docker and does not depend on WSL or other Linux-based proxies – but it can be difficult to get running on Windows.

Therefore this run-through will focus on Musubi Tuner, and on providing a completely local solution for Hunyuan LoRA training and generation, without the use of API-driven websites or commercial GPU-renting processes such as Runpod.



Click to play. Samples from LoRA training on Musubi Tuner for this article. All permissions granted by the person depicted, for the purposes of illustrating this article.

REQUIREMENTS

The installation will require at minimum a Windows 10 PC with a 30+/40+ series NVIDIA card that has at least 12GB of VRAM (though 16GB is recommended). The installation used for this article was tested on a machine with 64GB of system RAM and a NVIDIA 3090 graphics cards with 24GB of VRAM. It was tested on a dedicated test-bed system using a fresh install of Windows 10 Professional, on a partition with 600+GB of spare disk space.

WARNING

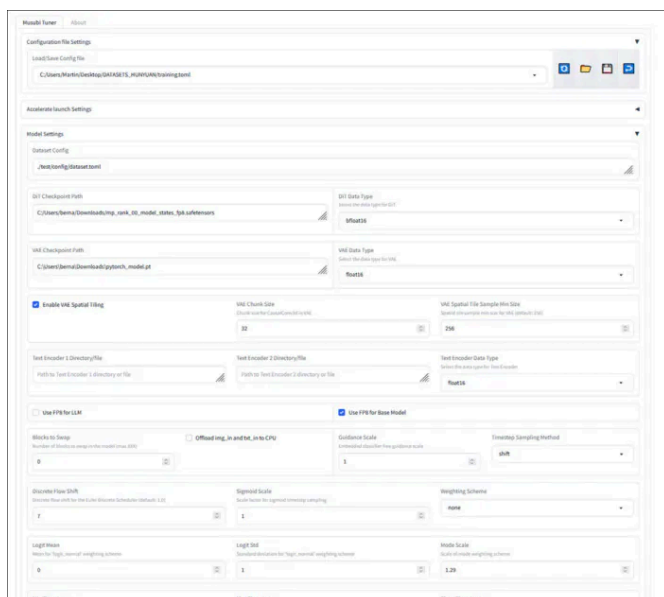
Installing Musubi Tuner and its prerequisites also entails the installation of developer-focused software and packages directly onto the main Windows installation of a PC. Taking the installation of ComfyUI into account, for the end stages, this project will require around 400-500 gigabytes of disk space. Though I have tested the procedure without incident several times in newly-installed test bed Windows 10 environments, neither I nor unite.ai are liable for any damage to systems from following these instructions. I advise you to back up any important data before attempting this kind of installation procedure.

Considerations

Is This Method Still Valid?

The generative AI scene is moving very fast, and we can expect better and more streamlined methods of Hunyuan Video LoRA frameworks this year.

...or even this week! While I was writing this article, the developer of Kohya/Musubi produced [musubi-tuner-gui](#), a sophisticated Gradio GUI for Musubi Tuner:



Obviously a user-friendly GUI is preferable to the BAT files that I use in this feature – once musubi-tuner-gui is working. As I write, it only went online five days ago, and I can find no account of anyone successfully using it.

According to posts in the repository, the new GUI is intended to be rolled directly into the Musubi Tuner project as soon as possible, which will end its current existence as a standalone GitHub repository.

Based on the present installation instructions, the new GUI gets cloned directly into the existing Musubi virtual environment; and, despite many efforts, I cannot get it to associate with the existing Musubi installation. This means that when it runs, it will find that it has no engine!

Once the GUI is integrated into Musubi Tuner, issues of this kind will surely be resolved. Though the author [concedes that the new project is 'really rough'](#), he is optimistic for its development and integration directly into Musubi Tuner.

Given these issues (also concerning default paths at install-time, and the use of the [UV Python package](#), which complicates certain procedures in the new release), we will probably have to wait a little for a smoother Hunyuan Video LoRA training experience. That said, it looks very promising!

But if you can't wait, and are willing to roll your sleeves up a bit, you can get Hunyuan video LoRA training running locally right now.

Let's get started.

Why Install *Anything* on Bare Metal?

(Skip this paragraph if you're not an advanced user)

Advanced users will wonder why I have chosen to install so much of the software on the bare metal Windows 10 installation instead of in a virtual environment. The reason is that the essential Windows port of the Linux-based [Triton package](#) is far more difficult to get working in a virtual environment. All the other bare-metal installations in the tutorial could not be installed in a virtual environment, as they must interface directly with local hardware.

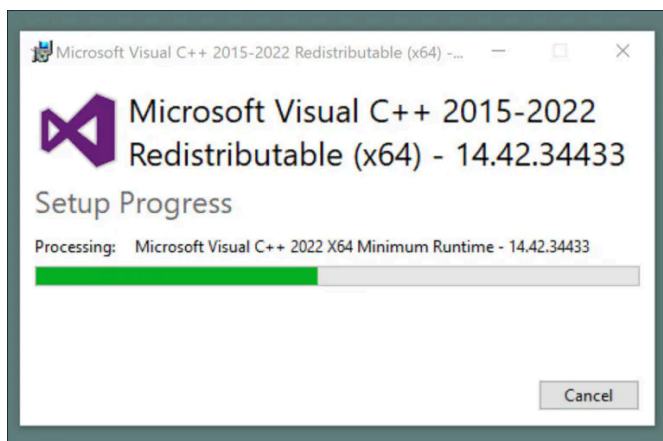
Installing Prerequisite Packages and Programs

For the programs and packages that must be initially installed, the order of installation matters. Let's get started.

1: Download Microsoft Redistributable

Download and install the Microsoft Redistributable package from https://aka.ms/vs/17/release/vc_redist.x64.exe.

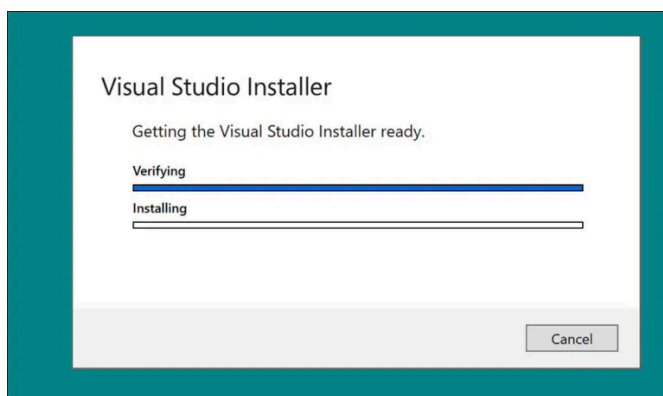
This is a straightforward and rapid installation.



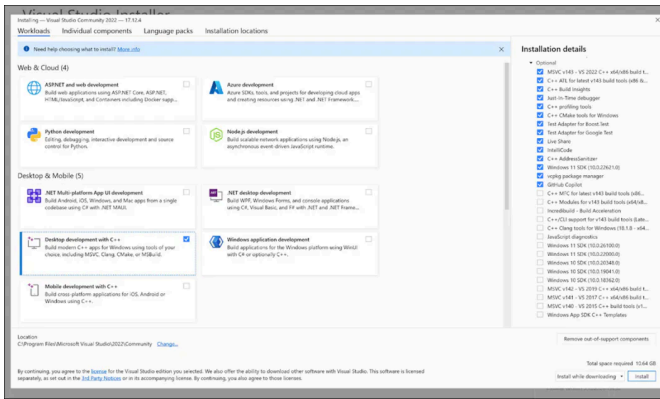
2: Install Visual Studio 2022

Download the Microsoft Visual Studio 2022 Community edition from <https://visualstudio.microsoft.com/downloads/?cid=learn-onpage-download-install-visual-studio-page-cta>

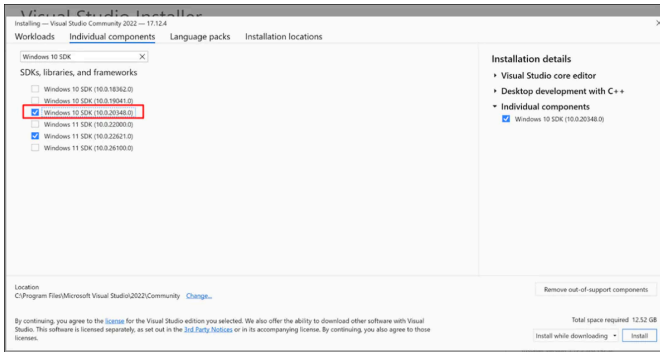
Start the downloaded installer:



We don't need every available package, which would be a heavy and lengthy install. At the initial *Workloads* page that opens, tick *Desktop Development with C++* (see image below).



Now click the *Individual Components* tab at the top-left of the interface and use the search box to find 'Windows SDK'.

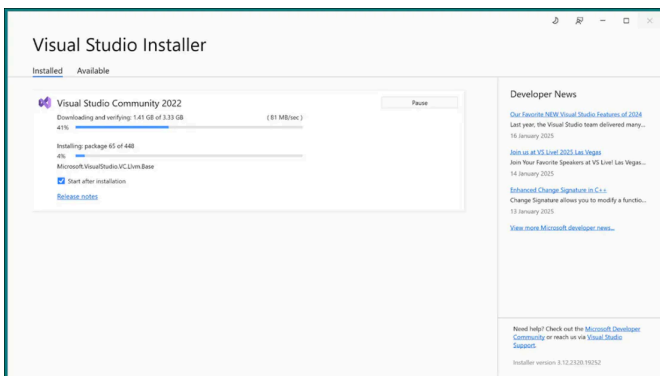


By default, only the *Windows 11 SDK* is ticked. If you are on Windows 10 (this installation procedure has not been tested by me on Windows 11), tick the latest Windows 10 version, indicated in the image above.

Search for 'C++ CMake' and check that *C++ CMake tools for Windows* is checked.



This installation will take at least 13 GB of space.



Once Visual Studio has installed, it will attempt to run on your computer. Let it open fully. When the Visual Studio's full-screen interface is finally visible, close the program.

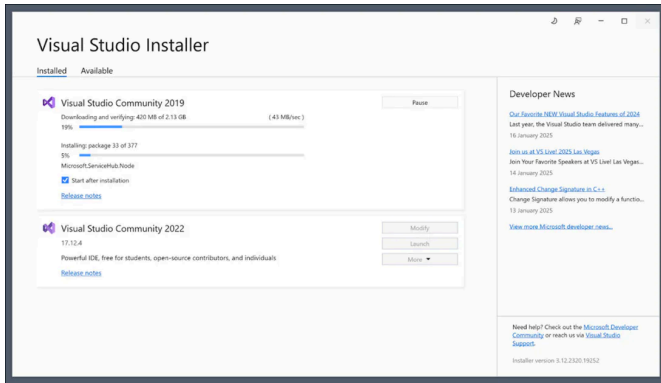
3: Install Visual Studio 2019

Some of the subsequent packages for Musubi are expecting an older version of Microsoft Visual Studio, while others need a more recent one.

Therefore also download the free Community edition of Visual Studio 19 either from Microsoft (<https://visualstudio.microsoft.com/vs/older-downloads/> – account required) or Techspot (<https://www.techspot.com/downloads/7241-visual-studio-2019.html>).

Install it with the same options as for Visual Studio 2022 (see procedure above, except that *Windows SDK* is already ticked in the Visual Studio 2019 installer).

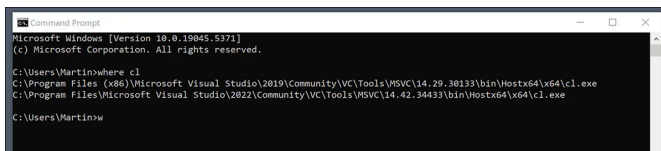
You'll see that the Visual Studio 2019 installer is already aware of the newer version as it installs:



When installation is complete, and you have opened and closed the installed Visual Studio 2019 application, open a Windows command prompt (Type *CMD* in Start Search) and type in and enter:

```
where cl
```

The result should be the known locations of the two installed Visual Studio editions.



If you instead get INFO: Could not find files for the given pattern(s), see the *Check Path* section of this article below, and use those instructions to add the relevant Visual Studio paths to Windows environment.

Save any changes made according to the *Check Paths* section below, and then try the where cl command again.

4: Install CUDA 11 + 12 Toolkits

The various packages installed in Musubi need different versions of [NVIDIA CUDA](#), which accelerates and optimizes training on NVIDIA graphics cards.

The reason we installed the Visual Studio versions *first* is that the NVIDIA CUDA installers search for and integrate with any existing Visual Studio installations.

Download an 11+ series CUDA installation package from:

https://developer.nvidia.com/cuda-11-8-0-download-archive?target_os=Windows&target_arch=x86_64&target_version=11&target_type=exe_local (download 'exe (local'))

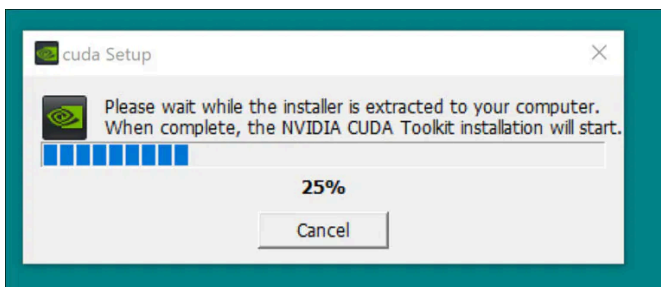
Download a 12+ series CUDA Toolkit installation package from:

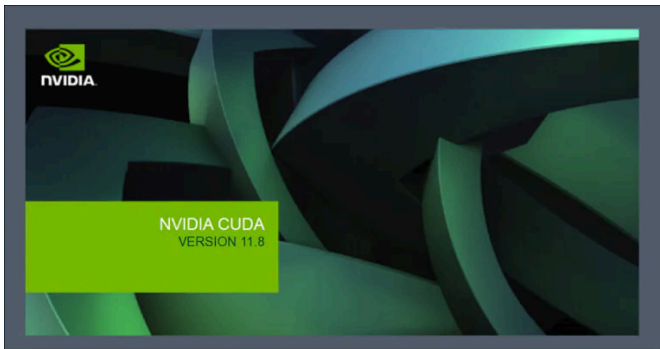
https://developer.nvidia.com/cuda-downloads?target_os=Windows&target_arch=x86_64

The installation process is identical for both installers. Ignore any warnings about the existence or non-existence of installation paths in Windows Environment variables – we are going to attend to this manually later.

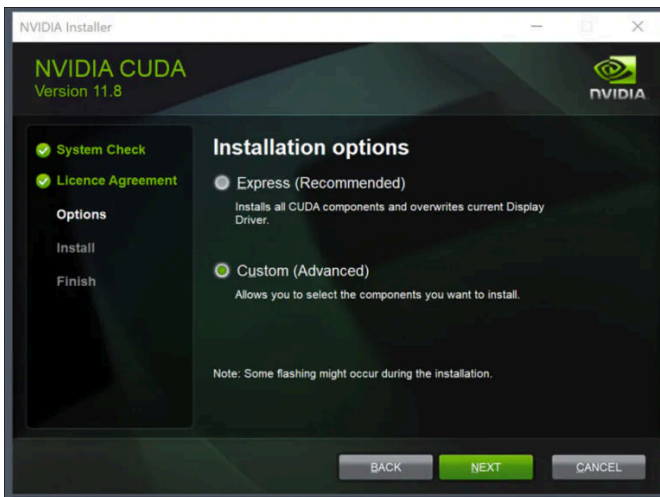
Install NVIDIA CUDA Toolkit V11+

Start the installer for the 11+ series CUDA Toolkit.

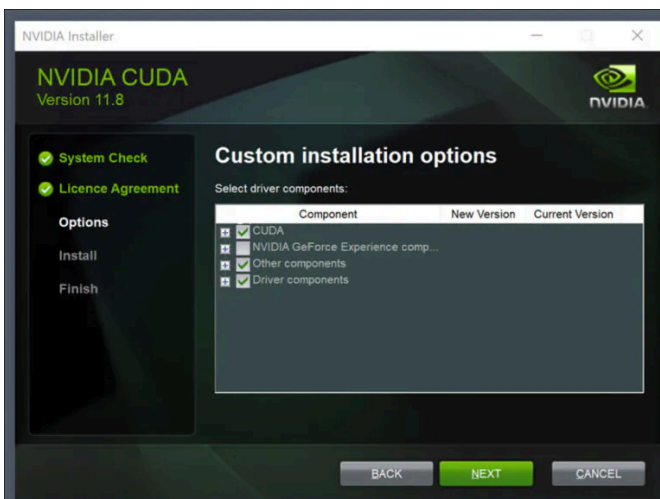




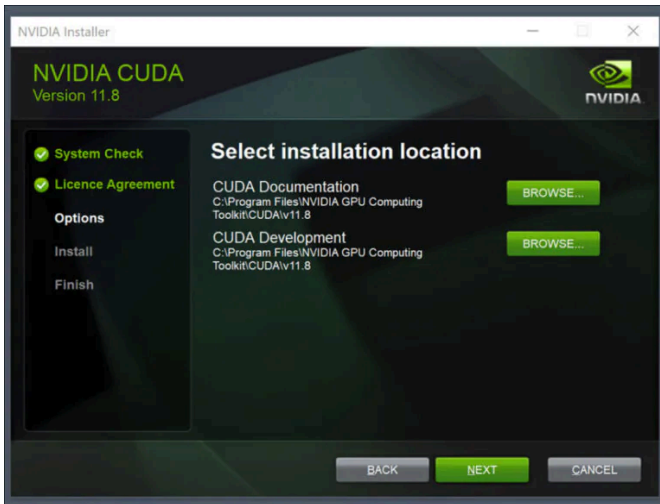
At *Installation Options*, choose *Custom (Advanced)* and proceed.



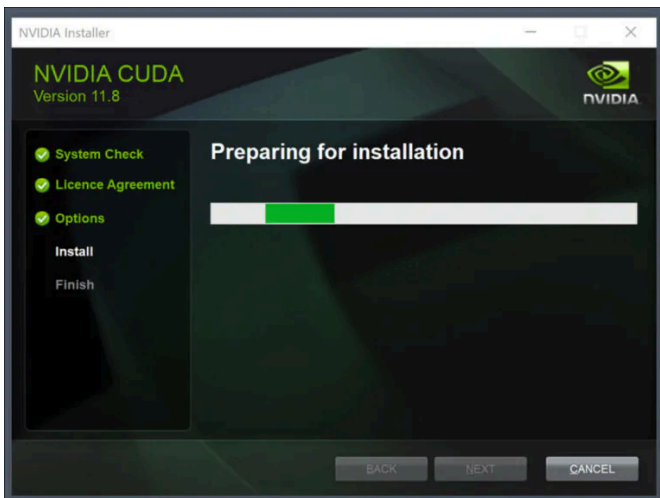
Uncheck the NVIDIA GeForce Experience option and click *Next*.



Leave *Select Installation Location* at defaults (this is important):



Click *Next* and let the installation conclude.



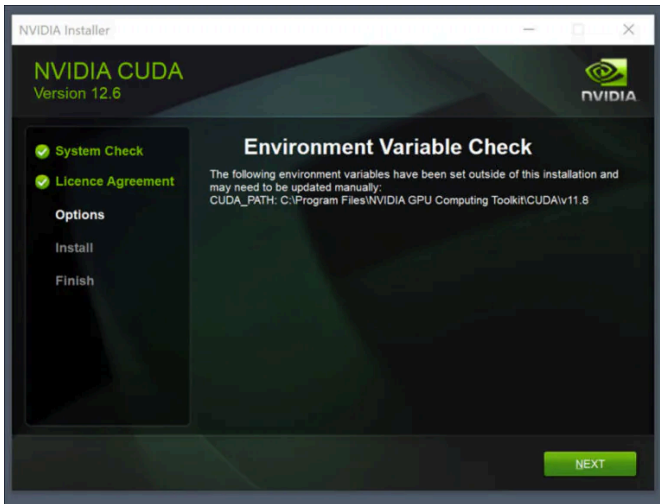
Ignore any warning or notes that the installer gives about *Nsight Visual Studio* integration, which is not needed for our use case.

Install NVIDIA CUDA Toolkit V12+

Repeat the entire process for the separate 12+ NVIDIA Toolkit installer that you downloaded:



The install process for this version is identical to the one listed above (the 11+ version), except for one warning about environment paths, which you can ignore:



When the 12+ CUDA version installation is completed, open a command prompt in Windows and type and enter:

```
nvcc --version
```

This should confirm information about the installed driver version:

```

C:\Users\Martin>nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2024 NVIDIA Corporation
Built on Wed_Oct_30_01:18:48_Pacific_Daylight_Time_2024
Cuda compilation tools, release 12.6, V12.6.85
Build cuda_12.6.r12.6/compiler.35059454_0

C:\Users\Martin>

```

To check that your card is recognized, type and enter:

```
nvidia-smi
```

```

C:\Users\Martin>nvidia-smi
Wed Jan 22 09:02:02 2025

+-----+
| NVIDIA-SMI 561.17              Driver Version: 561.17      CUDA Version: 12.6   |
+-----+-----+
| GPU Name   | Driver-Model | Bus-Id | Disp.A | Volatile Uncomm. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap |      Memory-Usage | GPU-Util  Compute M. |
|=====+=====+
| 0  NVIDIA GeForce RTX 3090   | WDDM        | 00000000:26:00:00 | On     | N/A                  |
| 0%   39C   P8      15W / 370W | 562MiB / 24576MiB | 0%         Default   |
+-----+-----+
| Processes: |
| GPU   GI   CI        PID   Type   Process name                      GPU Memory |
|=====+=====+
| 0     N/A  N/A       7164   C+G   C:\Windows\explorer.exe           N/A       |
| 0     N/A  N/A       7208   C+G   ...5nih2txyewy\ShellExperienceHost.exe N/A       |
| 0     N/A  N/A       8104   C+G   ...Search_cw5nih2txyewy\SearchApp.exe N/A       |
| 0     N/A  N/A       8876   C+G   ...crosoft\Edge\Application\msedge.exe N/A       |
| 0     N/A  N/A       8920   C+G   ...UI3Apps\PowerToys.AdvancedPaste.exe N/A       |
| 0     N/A  N/A      10260   C+G   ...CBS_cw5nih2txyewy\TextInputHost.exe N/A       |
| 0     N/A  N/A      11720   C+G   ...table\App\Notepad++64\notepad++.exe N/A       |
+-----+-----+

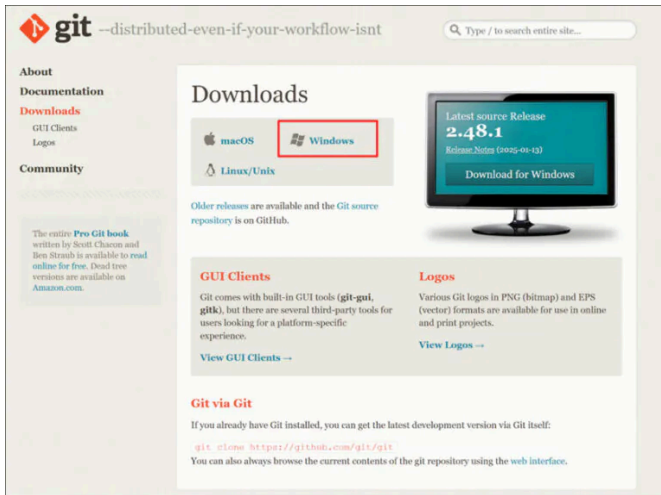
C:\Users\Martin>

```

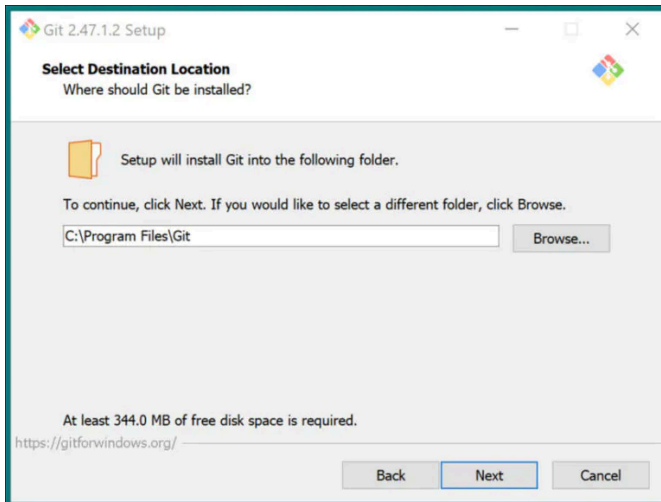
5: Install GIT

GIT will be handling the installation of the Musubi repository on your local machine. Download the GIT installer at:

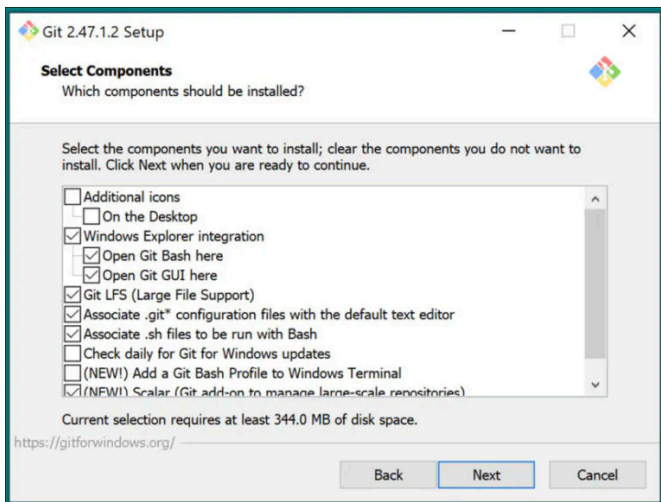
<https://git-scm.com/downloads/win> ('64-bit Git for Windows Setup')



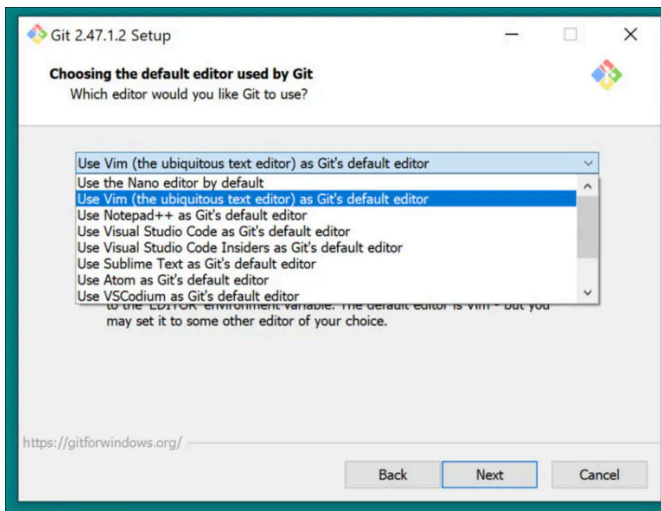
Run the installer:



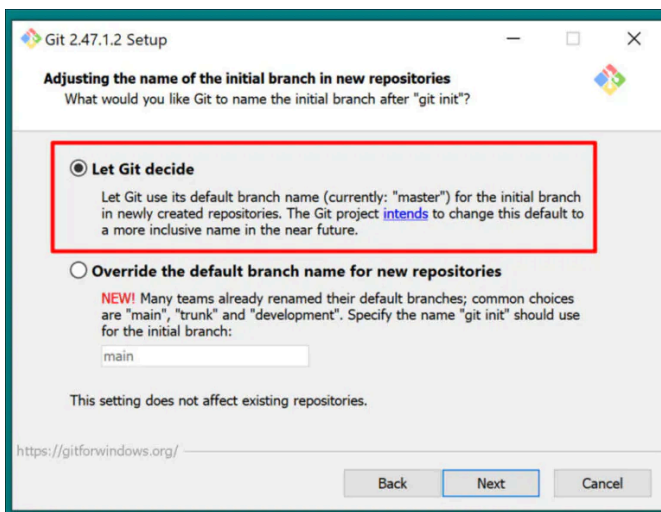
Use default settings for *Select Components*:



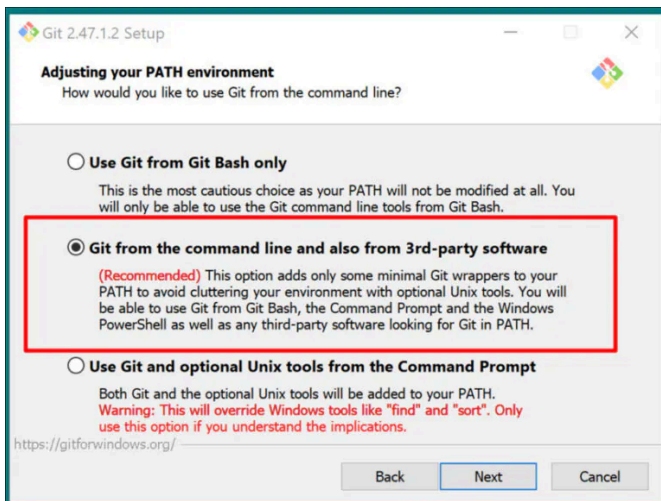
Leave the default editor at *Vim*:



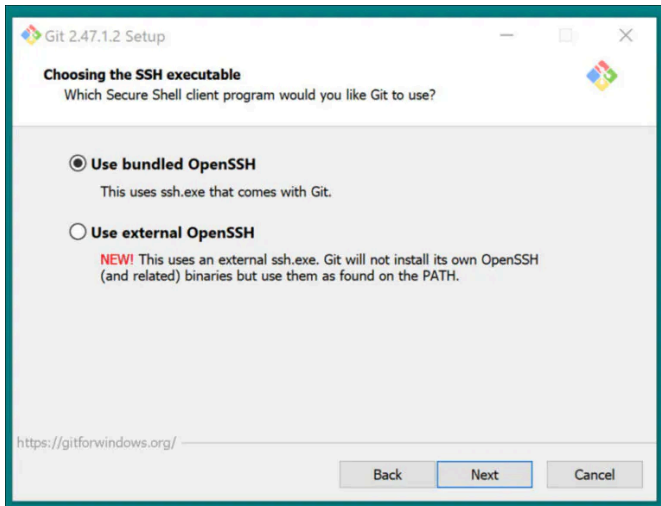
Let GIT decide about branch names:



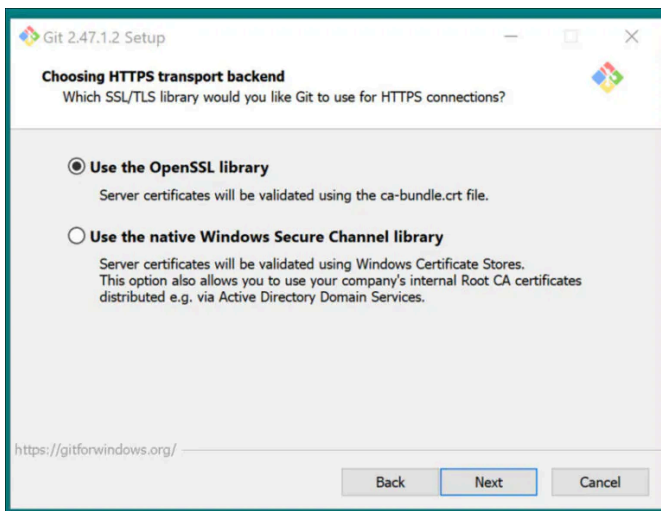
Use recommended settings for the *Path* Environment:



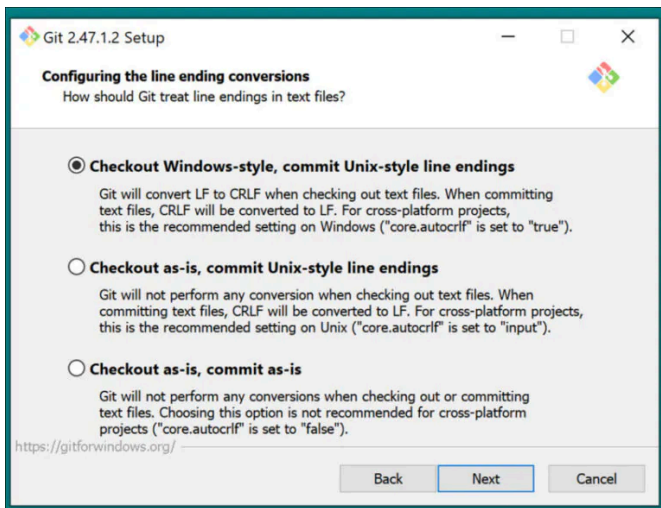
Use recommended settings for SSH:



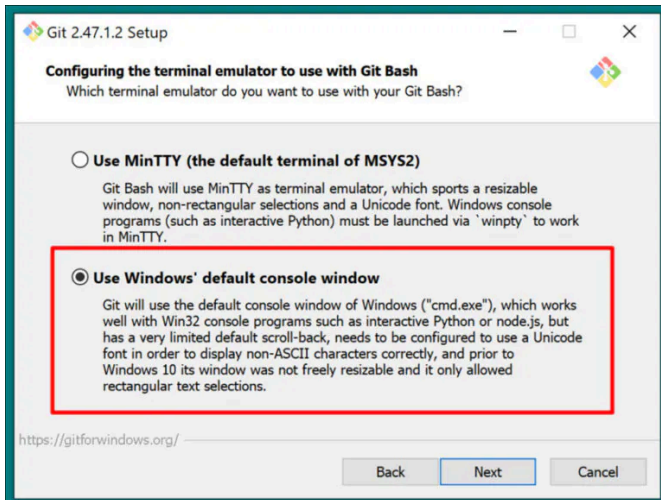
Use recommended settings for *HTTPS Transport backend*:



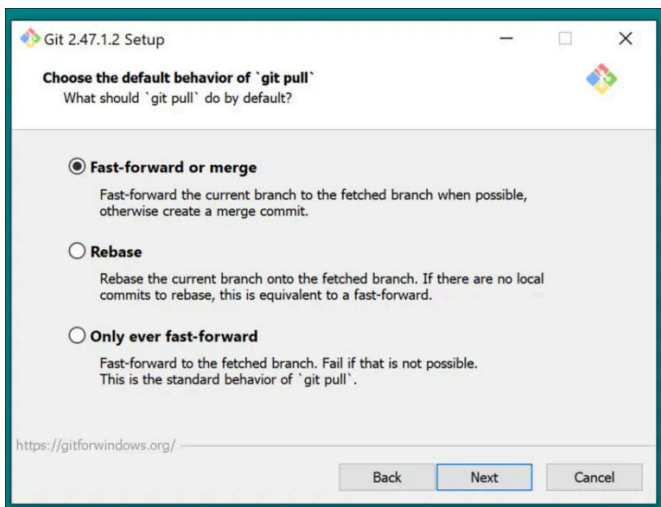
Use recommended settings for line-ending conversions:



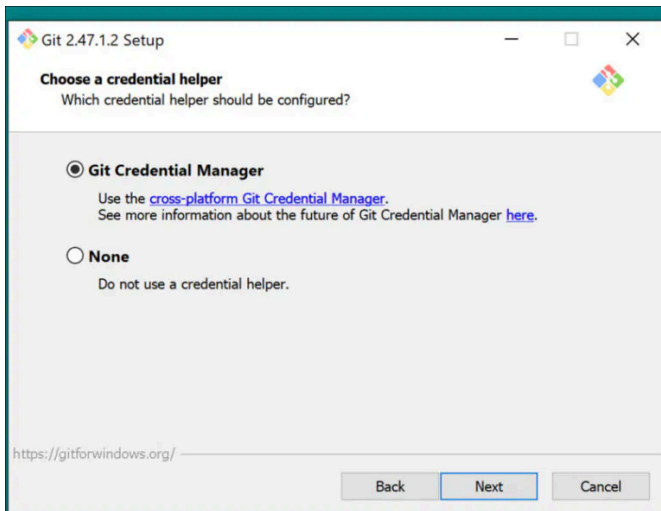
Choose Windows default console as the Terminal Emulator:



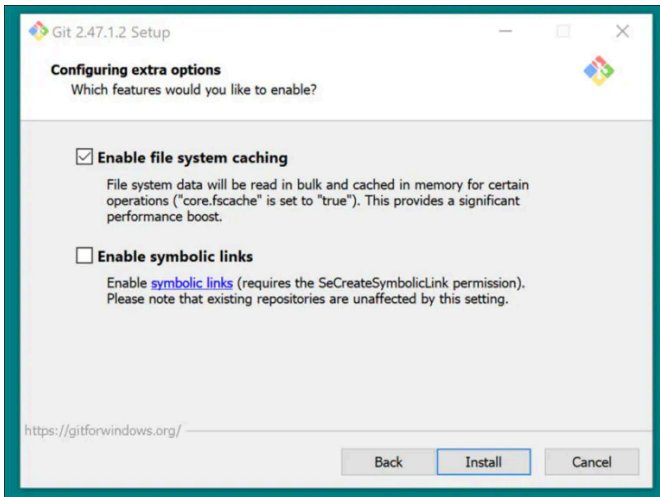
Use default settings (*Fast-forward or merge*) for Git Pull:



Use Git-Credential Manager (the default setting) for Credential Helper:

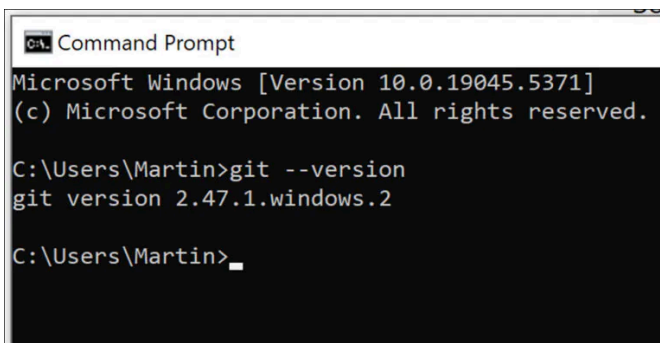


In *Configuring extra options*, leave *Enable file system caching* ticked, and *Enable symbolic links* unticked (unless you are an advanced user who is using hard links for a centralized model repository).



Conclude the installation and test that Git is installed properly by opening a CMD window and typing and entering:

```
git --version
```



GitHub Login

Later, when you attempt to clone GitHub repositories, you may be challenged for your GitHub credentials. To anticipate this, log into your GitHub account (create one, if necessary) on any browsers installed on your Windows system. In this way, the OAuth authentication method (a pop-up window) should take as little time as possible.

After that initial challenge, you should stay authenticated automatically.

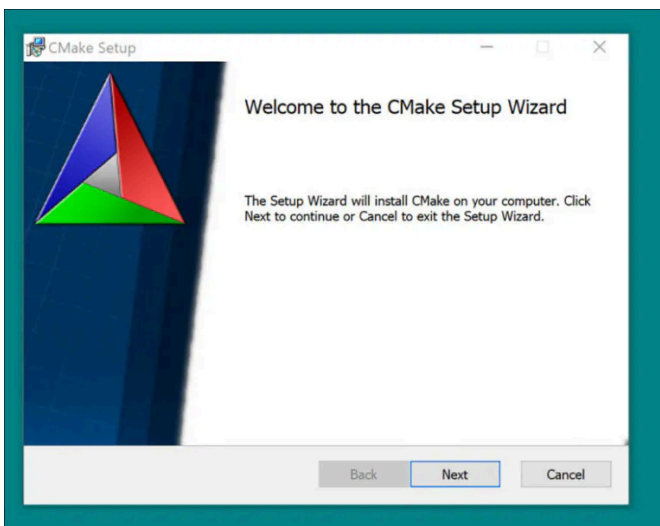
6: Install CMake

CMake 3.21 or newer is required for parts of the Musubi installation process. CMake is a cross-platform development architecture capable of orchestrating diverse compilers, and of compiling software from source code.

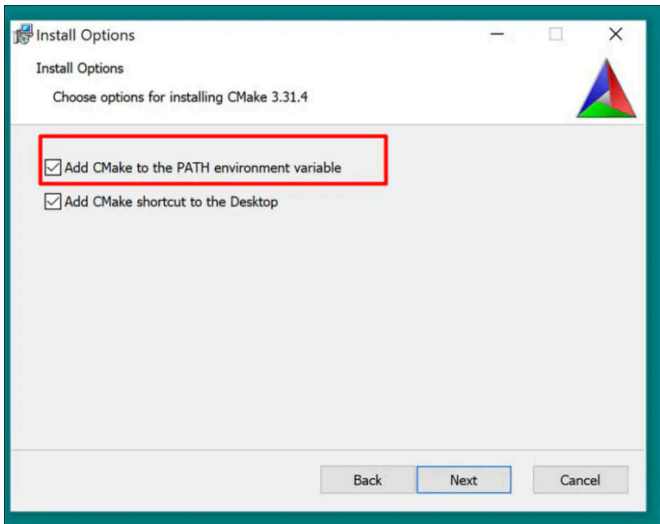
Download it at:

<https://cmake.org/download/> ('Windows x64 Installer')

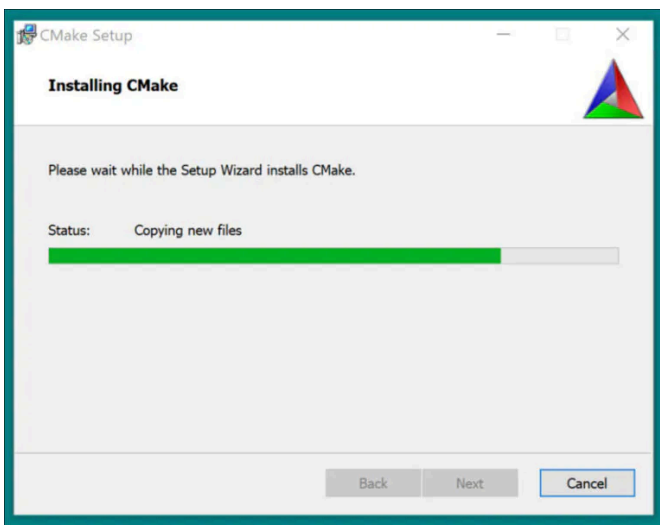
Launch the installer:



Ensure *Add Cmake to the PATH environment variable* is checked.



Press *Next*.

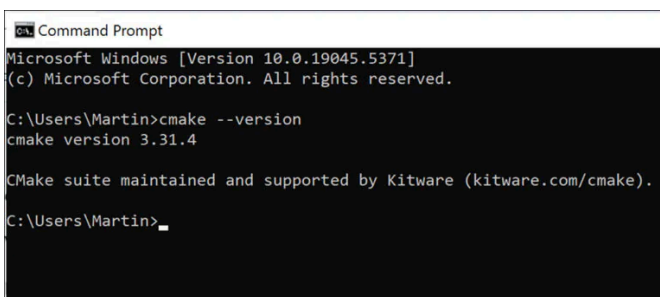


Type and enter this command in a Windows Command prompt:

```
cmake --version
```

If CMake installed successfully, it will display something like:

```
cmake version 3.31.4
CMake suite maintained and supported by Kitware (kitware.com/cmake).
```

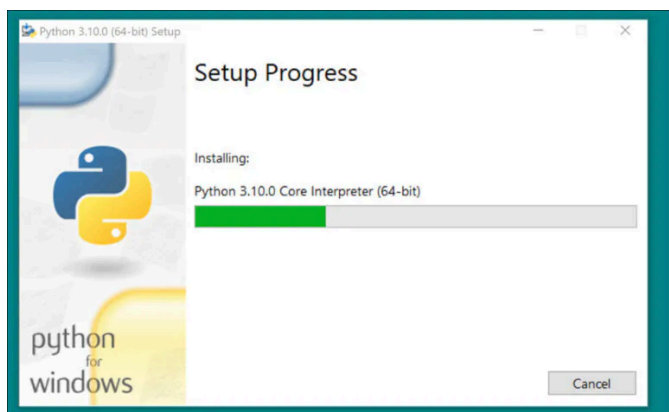
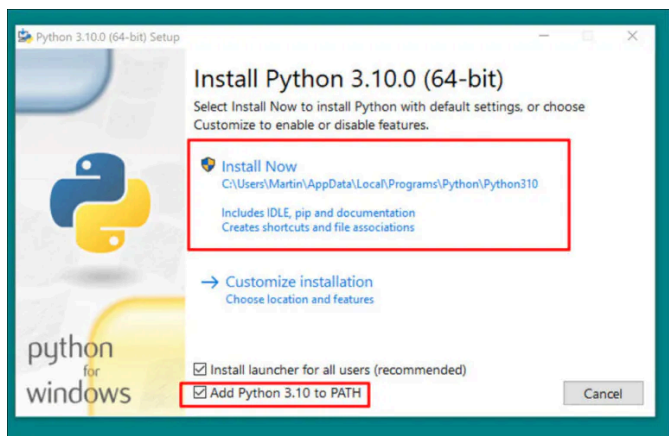


7: Install Python 3.10

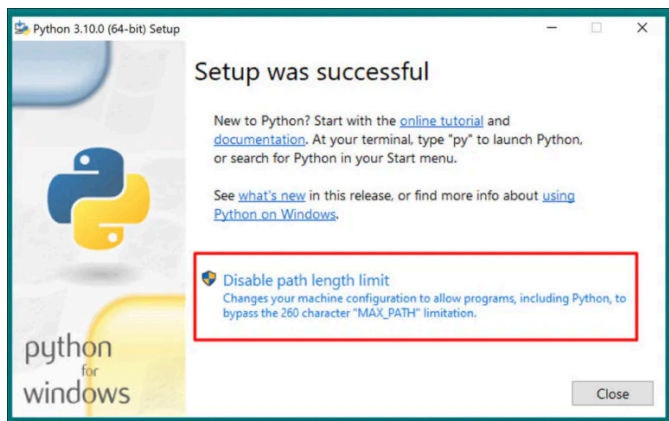
The Python interpreter is central to this project. Download the 3.10 version (the best compromise between the different demands of Musubi packages) at:

<https://www.python.org/downloads/release/python-3100/> ('Windows installer (64-bit)')

Run the download installer, and leave at default settings:



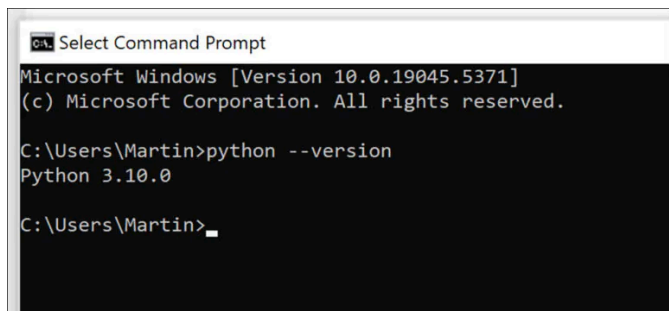
At the end of the installation process, click *Disable path length limit* (requires UAC admin confirmation):



In a Windows Command prompt type and enter:

```
python --version
```

This should result in Python 3.10.0

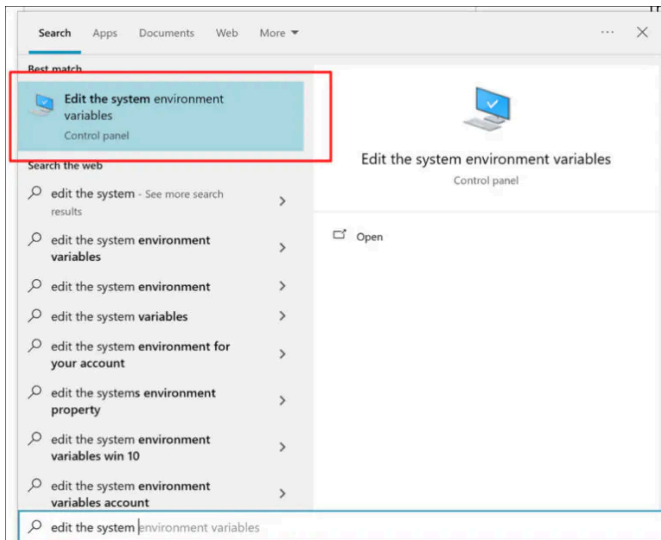


Check Paths

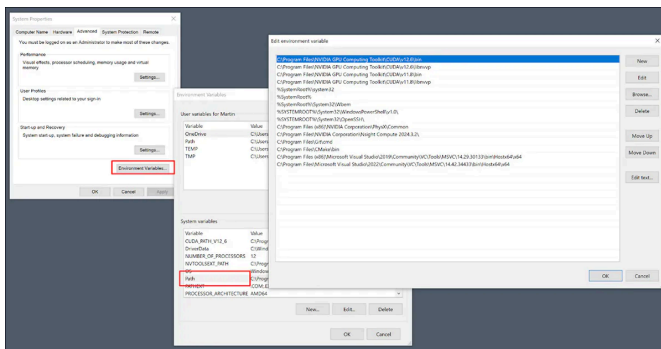
The cloning and installation of the Musubi frameworks, as well as its normal operation after installation, requires that its components know the path to several important external components in Windows, particularly CUDA.

So we need to open the path environment and check that all the requisites are in there.

A quick way to get to the controls for Windows Environment is to type *Edit the system environment variables* into the Windows search bar.



Clicking this will open the *System Properties* control panel. In the lower right of *System Properties*, click the *Environment Variables* button, and a window called *Environment Variables* opens up. In the *System Variables* panel in the bottom half of this window, scroll down to *Path* and double-click it. This opens a window called *Edit environment variables*. Drag the width of this window wider so you can see the full path of the variables:

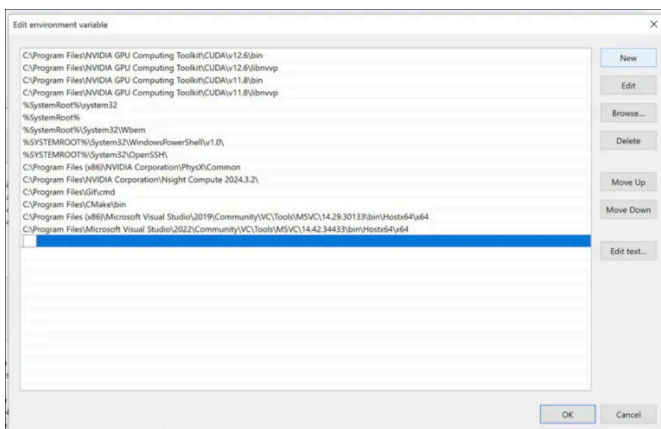


Here the important entries are:

```
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.6\bin
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.6\libnvvp
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.8\bin
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.8\libnvvp
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30133\bin\Hostx64\x64
C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Tools\MSVC\14.42.34433\bin\Hostx64\x64
C:\Program Files\Git\cmd
C:\Program Files\Make\bin
```

In most cases, the correct path variables should already be present.

Add any paths that are missing by clicking *New* on the left of the *Edit environment variable* window and pasting in the correct path:



Do NOT just copy and paste from the paths listed above; check that each equivalent path exists in your own Windows installation.

If there are minor path variations (particularly with Visual Studio installations), use the paths listed above to find the correct target folders (i.e., *x64* in *Host64* in your own installation). Then paste *those* paths into the *Edit environment variable* window.

After this, restart the computer.

Installing Musubi

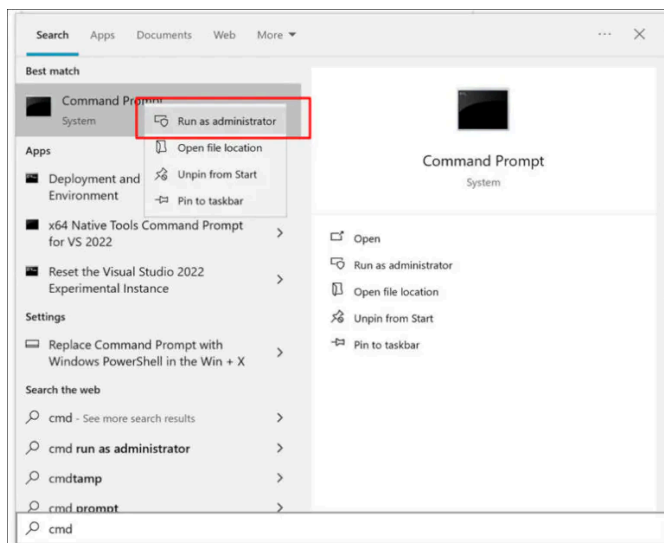
Upgrade PIP

Using the latest version of the PIP installer can smooth some of the installation stages. In a Windows Command prompt with administrator privileges (see *Elevation*, below), type and enter:

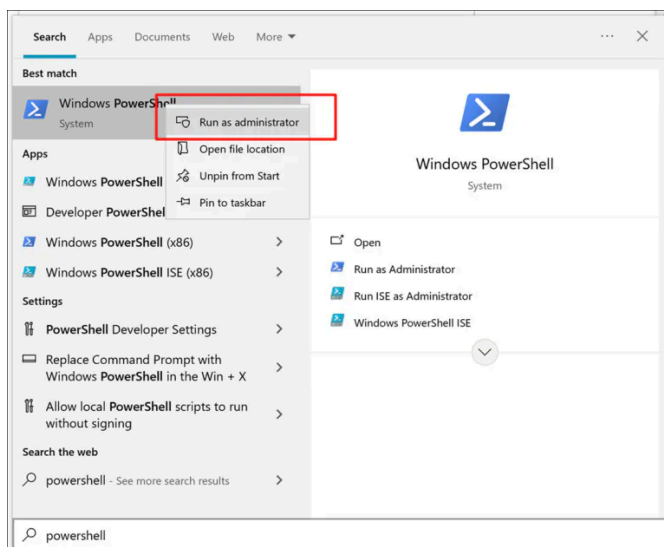
```
pip install --upgrade pip
```

Elevation

Some commands may require elevated privileges (i.e., to be run as an administrator). If you receive error messages about permissions in the following stages, close the command prompt window and reopen it in administrator mode by typing *CMD* into Windows search box, right-clicking on *Command Prompt* and selecting *Run as administrator*.



For the next stages, we are going to use Windows Powershell instead of the Windows Command prompt. You can find this by entering *Powershell* into the Windows search box, and (as necessary) right-clicking on it to *Run as administrator*.



Install Torch

In Powershell, type and enter:

```
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118
```

Be patient while the many packages install.

When completed, you can verify a GPU-enabled PyTorch installation by typing and entering:

```
python -c "import torch; print(torch.cuda.is_available())"
```

This should result in:

```
C:\WINDOWS\system32>python -c "import torch;
print(torch.cuda.is_available())"
True
```

Install Triton for Windows

Next, the installation of the *Triton for Windows* component. In elevated Powershell, enter (on a single line):

```
pip install https://github.com/woc0rdho/triton-windows/releases/download/v3.1.0-windows.post8/triton-3.1.0-cp310-cp310-win_amd64.whl
```

(The installer triton-3.1.0-cp310-cp310-win_amd64.whl works for both Intel and AMD CPUs as long as the architecture is 64-bit and the environment matches the Python version)

After running, this should result in:

```
Successfully installed triton-3.1.0
```

We can check if Triton is working by importing it in Python. Enter this command:

```
python -c "import triton; print('Triton is working')"
```

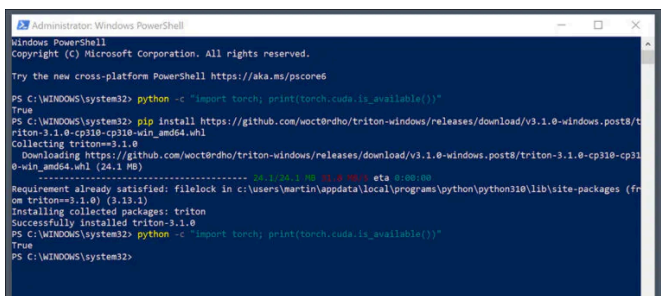
This should output:

```
Triton is working
```

To check that Triton is GPU-enabled, enter:

```
python -c "import torch; print(torch.cuda.is_available())"
```

This should result in True:



```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\WINDOWS\system32> python -c "import torch; print(torch.cuda.is_available())"
True
PS C:\WINDOWS\system32> pip install https://github.com/woc0rdho/triton-windows/releases/download/v3.1.0-windows.post8/t
riton-3.1.0-cp310-cp310-win_amd64.whl
Collecting triton==3.1.0
  Downloading https://github.com/woc0rdho/triton-windows/releases/download/v3.1.0-windows.post8/triton-3.1.0-cp310-cp310-win_amd64.whl (24.1 MB)
    -----
Requirement already satisfied: filelock in c:\users\martin\appdata\local\programs\python\python310\lib\site-packages (fr
om triton==3.1.0) (3.13.1)
Installing collected packages: triton
Successfully installed triton-3.1.0
PS C:\WINDOWS\system32> python -c "import torch; print(torch.cuda.is_available())"
True
PS C:\WINDOWS\system32>
```

Create the Virtual Environment for Musubi

From now on, we will install any further software into a *Python virtual environment* (or [venv](#)). This means that all you will need to do to uninstall all the following software is to drag the venv's installation folder to the trash.

Let's create that installation folder: make a folder called *Musubi* on your desktop. The following examples assume that this folder exists: C:\Users\[Your Profile Name]\Desktop\Musubi\.

In Powershell, navigate to that folder by entering:

```
cd C:\Users\[Your Profile Name]\Desktop\Musubi
```

We want the virtual environment to have access to what we have installed already (especially Triton), so we will use the `--system-site-packages` flag. Enter this:

```
python -m venv --system-site-packages musubi
```

Wait for the environment to be created, and then activate it by entering:

```
.\musubi\Scripts\activate
```

From this point on, you can tell that you are in the activated virtual environment by the fact that (musubi) appears at the beginning of all your prompts.

```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\WINDOWS\system32> cd C:\Users\Martin\Desktop\Musubi
PS C:\Users\Martin\Desktop\Musubi> python -m venv --system-site-packages musubi
PS C:\Users\Martin\Desktop\Musubi> .\musubi\Scripts\activate
(musubi) PS C:\Users\Martin\Desktop\Musubi>
```

Clone the Repository

Navigate to the newly-created *musubi* folder (which is inside the *Musubi* folder on your desktop):

```
cd musubi
```

Now that we are in the right place, enter the following command:

```
git clone https://github.com/kohya-ss/musubi-tuner.git
```

Wait for the cloning to complete (it will not take long).

```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\WINDOWS\system32> cd C:\Users\Martin\Desktop\Musubi
PS C:\Users\Martin\Desktop\Musubi> python -m venv --system-site-packages musubi
PS C:\Users\Martin\Desktop\Musubi> .\musubi\Scripts\activate
(musubi) PS C:\Users\Martin\Desktop\Musubi> cd musubi
(musubi) PS C:\Users\Martin\Desktop\Musubi\musubi> git clone https://github.com/kohya-ss/musubi-tuner.git
Cloning into 'musubi-tuner'...
remote: Enumerating objects: 425, done.
remote: Counting objects: 100% (189/189), done.
remote: Compressing objects: 100% (88/88), done.
remote: Total 425 (delta 136), reused 129 (delta 99), pack-reused 236 (from 2)
Receiving objects: 100% (425/425), 423.54 KiB | 2.54 MiB/s, done.
Resolving deltas: 100% (245/245), done.
(musubi) PS C:\Users\Martin\Desktop\Musubi\musubi>
```

Installing Requirements

Navigate to the installation folder:

```
cd musubi-tuner
```

Enter:

```
pip install -r requirements.txt
```

Wait for the many installations to finish (this will take longer).

```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\WINDOWS\system32> cd C:\Users\Martin\Desktop\Musubi
PS C:\Users\Martin\Desktop\Musubi> python -m venv --system-site-packages musubi
PS C:\Users\Martin\Desktop\Musubi> .\musubi\Scripts\activate
(musubi) PS C:\Users\Martin\Desktop\Musubi> cd musubi
(musubi) PS C:\Users\Martin\Desktop\Musubi\musubi> git clone https://github.com/kohya-ss/musubi-tuner.git
Cloning into 'musubi-tuner'...
remote: Enumerating objects: 425, done.
remote: Counting objects: 100% (189/189), done.
remote: Compressing objects: 100% (88/88), done.
remote: Total 425 (delta 136), reused 129 (delta 99), pack-reused 236 (from 2)
Receiving objects: 100% (425/425), 423.54 KiB | 2.54 MiB/s, done.
Resolving deltas: 100% (245/245), done.
(musubi) PS C:\Users\Martin\Desktop\Musubi\musubi> cd musubi-tuner
(musubi) PS C:\Users\Martin\Desktop\Musubi\musubi\musubi-tuner> pip install -r requirements.txt
Collecting accelerate==1.2.1
  Downloading accelerate-1.2.1-py3-none-any.whl (336 kB)
  | 336 kB 2.2 MB/s
Collecting av==14.0.1
  Downloading av-14.0.1-cp310-cp310-win_amd64.whl (25.8 MB)
  | 25.8 MB 6.8 MB/s
Collecting bitsandbytes==0.45.0
  Downloading bitsandbytes-0.45.0-py3-none-win_amd64.whl (68.5 MB)
  | 68.5 MB 6.4 MB/s
Collecting diffusers==0.32.1
  Downloading diffusers-0.32.1-py3-none-any.whl (3.2 MB)
  | 3.2 MB ...
Collecting einops==0.7.0
  Downloading einops-0.7.0-py3-none-any.whl (44 kB)
  | 44 kB ...
Collecting huggingface-hub==0.26.5
  Downloading huggingface-hub-0.26.5-py3-none-any.whl (447 kB)
  | 447 kB ...
Collecting opencv-python==4.10.0.84
  Downloading opencv-python-4.10.0.84-cp37-ab13-win_amd64.whl (38.8 MB)
  | 38.8 MB ...
```

Automating Access to the Hunyuan Video Venv

To easily activate and access the new venv for future sessions, paste the following into Notepad and save it with the name *activate.bat*, saving it with *All files* option (see image below).

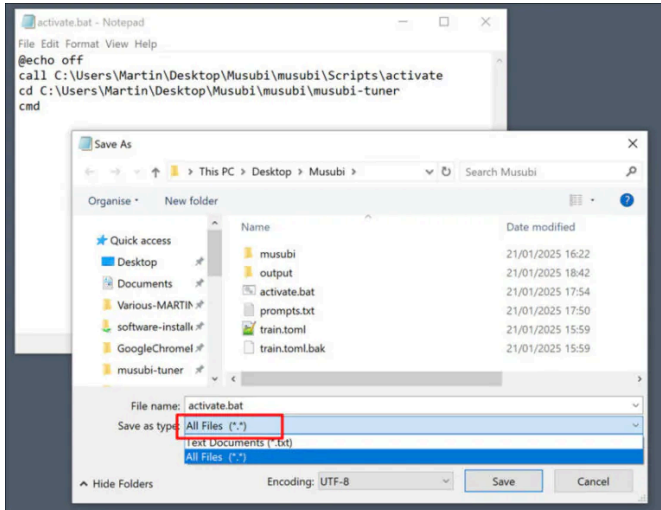
```
@echo off

call C:\Users\[Your Profile Name]\Desktop\Musubi\musubi\Scripts\activate

cd C:\Users\[Your Profile Name]\Desktop\Musubi\musubi\musubi-tuner
```

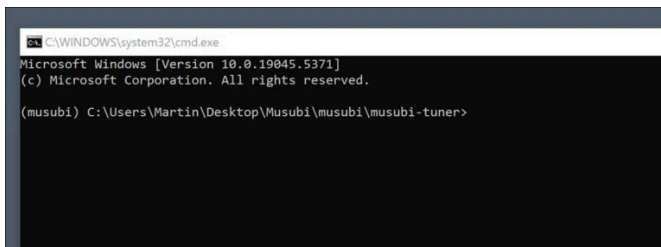
cmd

(Replace [Your Profile Name] with the real name of your Windows user profile)



It does not matter into which location you save this file.

From now on you can double-click *activate.bat* and start work immediately.



Using Musubi Tuner

Downloading the Models

The Hunyuan Video LoRA training process requires the downloading of at least seven models in order to support all the possible optimization options for pre-caching and training a Hunyuan video LoRA. Together, these models weigh more than 60GB.

Current instructions for downloading them can be found at <https://github.com/kohya-ss/musubi-tuner?tab=readme-ov-file#model-download>

However, these are the download instructions at the time of writing:

clip_l.safetensors
llava_llama3_fp16.safetensors and
llava_llama3_fp8_scaled.safetensors
can be downloaded at:
https://huggingface.co/Comfy-Org/HunyuanVideo_repackaged/tree/main/split_files/text_encoders

mp_rank_00_model_states.pt
mp_rank_00_model_states_fp8.pt and
mp_rank_00_model_states_fp8_map.pt
can be downloaded at:
<https://huggingface.co/tencent/HunyuanVideo/tree/main/hunyuan-video-t2v-720p/transformers>

pytorch_model.pt
can be downloaded at:
<https://huggingface.co/tencent/HunyuanVideo/tree/main/hunyuan-video-t2v-720p/vae>

Though you can place these in any directory you choose, for consistency with later scripting, let's put them in:

C:\Users\[Your Profile Name]\Desktop\Musubi\musubi\musubi-tuner\models\

This is consistent with the directory arrangement prior to this point. Any commands or instructions hereafter will assume that this is where the models are situated; and don't forget to replace [Your Profile Name] with your real Windows profile folder name.

Dataset Preparation

Ignoring community controversy on the point, it's fair to say that you will need somewhere between 10-100 photos for a training dataset for your Hunyuan LoRA. Very good results can be obtained even with 15 images, so long as the images are well-balanced and of good quality.

A Hunyuan LoRA can be trained both on images or very short and low-res video clips, or even a mixture of each – although using video clips as training data is challenging, even for a 24GB card.

However, video clips are only really useful if your character *moves in such an unusual way that the Hunyuan Video foundation model might not know about it*, or be able to guess.

Examples would include Roger Rabbit, a xenomorph, The Mask, Spider-Man, or other personalities that possess *unique* characteristic movement.

Since Hunyuan Video already knows how ordinary men and women move, video clips are not necessary to obtain a convincing Hunyuan Video LoRA human-type character. So we'll use static images.

Image Preparation

The Bucket List

The TLDR version:

It's best to either use images that are all the same size for your dataset, or use a 50/50 split between two different sizes, i.e., 10 images that are 512x768px and 10 that are 768x512px.

The training might go well even if you don't do this – Hunyuan Video LoRAs can be surprisingly forgiving.

The Longer Version

As with Kohya-ss LoRAs for static generative systems such as Stable Diffusion, [bucketing](#) is used to distribute the workload across differently-sized images, allowing larger images to be used without causing out-of-memory errors at training time (i.e., bucketing 'cuts up' the images into chunks that the GPU can handle, while maintaining the semantic integrity of the whole image).

For each size of image you include in your training dataset (i.e., 512x768px), a bucket, or 'sub-task' will be created for that size. So if you have the following distribution of images, this is how the bucket attention becomes unbalanced, and risks that some photos will be given greater consideration in training than others:

2x 512x768px images
7x 768x512px images
1x 1000x600px image
3x 400x800px images

We can see that bucket attention is divided unequally among these images:

Number of images	Image Ratio	Bucket Attention
2	512x768	15.0
7	768x512	54.0
1	1000x600	8.0
3	400x800	23.0

Therefore either stick to one format size, or try and keep the distribution of different sizes relatively equal.

In either case, avoid very large images, as this is likely to slow down training, to negligible benefit.

For simplicity, I have used 512x768px for all the photos in my dataset.

Disclaimer: *The model (person) used in the dataset gave me full permission to use these pictures for this purpose, and exercised approval of all AI-based output depicting her likeness featured in this article.*



My dataset consists of 40 images, in PNG format (though JPG is fine too). My images were stored at
C:\Users\Martin\Desktop\DATASETS_HUNYUAN\examplewoman

You should create a *cache* folder inside the training image folder:

Name	Size	Type	Modified	Attr
cache		File Folder	Today 3:10 PM	-----
examplewoman-0000001.png	514 KB	IrfanView PNG File	Today 2:44 PM	-a-----
examplewoman-0000002.png	487 KB	IrfanView PNG File	Today 2:44 PM	-a-----
examplewoman-0000003.png	536 KB	IrfanView PNG File	Today 2:44 PM	-a-----
examplewoman-0000004.png	512 KB	IrfanView PNG File	Today 2:44 PM	-a-----
examplewoman-0000005.png	575 KB	IrfanView PNG File	Today 2:44 PM	-a-----
examplewoman-0000006.png	530 KB	IrfanView PNG File	Today 2:44 PM	-a-----
examplewoman-0000007.png	558 KB	IrfanView PNG File	Today 2:44 PM	-a-----
examplewoman-0000008.png	569 KB	IrfanView PNG File	Today 2:44 PM	-a-----
examplewoman-0000009.png	562 KB	IrfanView PNG File	Today 2:44 PM	-a-----
examplewoman-0000010.png	484 KB	IrfanView PNG File	Today 2:44 PM	-a-----

Now let's create a special file that will configure the training.

TOML Files

The training and pre-caching processes of Hunyuan Video LoRAs obtains the file paths from a flat text file with the [.toml](#) extension.

For my test, the TOML is located at C:\Users\Martin\Desktop\DATASETS_HUNYUAN\training.toml

The contents of my training TOML look like this:

```
[general]

resolution = [512, 768]

caption_extension = ".txt"

batch_size = 1

enable_bucket = true

bucket_no_upscale = false

[[datasets]]

image_directory = "C:\\Users\\Martin\\Desktop\\DATASETS_HUNYUAN\\examplewoman"

cache_directory = "C:\\Users\\Martin\\Desktop\\DATASETS_HUNYUAN\\examplewoman\\cache"

num_repeats = 1
```

(The double back-slashes for image and cache directories are not always necessary, but they can help to avoid errors in cases where there is a space in the path. I have trained models with .toml files that used single-forward and single-backward slashes)

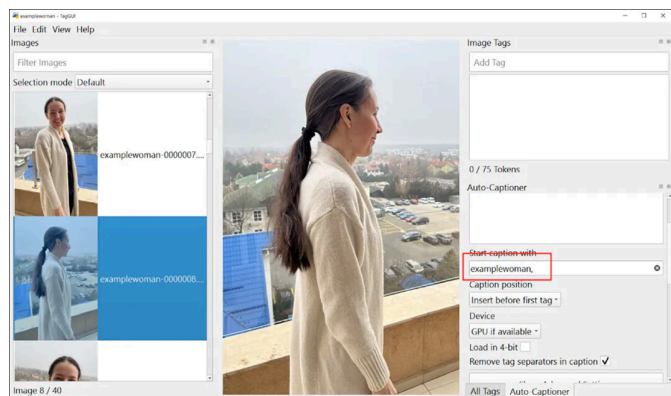
We can see in the resolution section that two resolutions will be considered – 512px and 768px. You can also leave this at 512, and still obtain good results.

Captions

Hunyuan Video is a *text+vision* foundation model, so we need descriptive captions for these images, which will be considered during training. The training process will fail without captions.

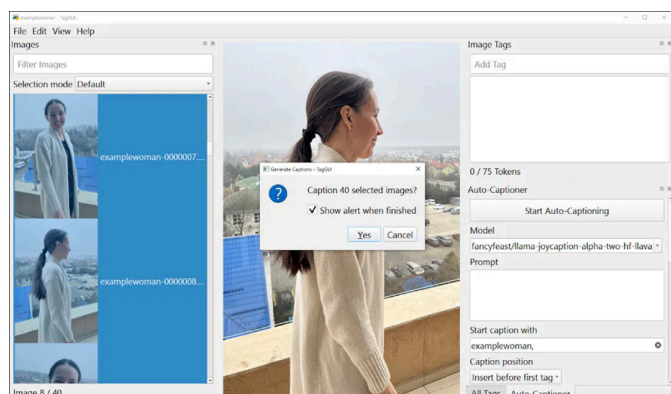
There are a [multitude](#) of open source captioning systems we could use for this task, but let's keep it simple and use the [taggui](#) system. Though it is stored at GitHub, and though it does download some very heavy deep learning models on first run, it comes in the form of a simple Windows executable that loads Python libraries and a straightforward GUI.

After starting Taggui, use *File > Load Directory* to navigate to your image dataset, and optionally put a token identifier (in this case, *examplewoman*) that will be added to all the captions:

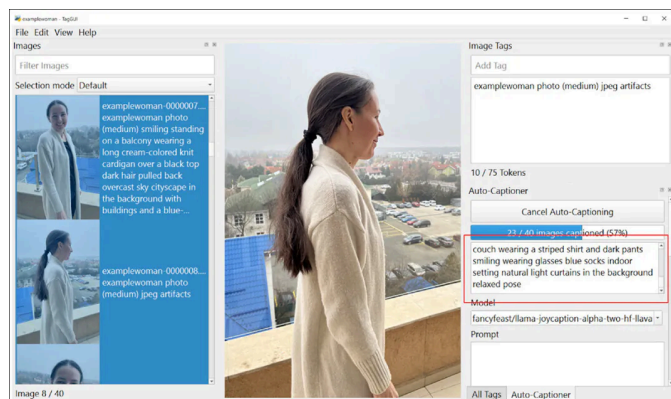


(Be sure to turn off *Load in 4-bit* when Taggui first opens – it will throw errors during captioning if this is left on)

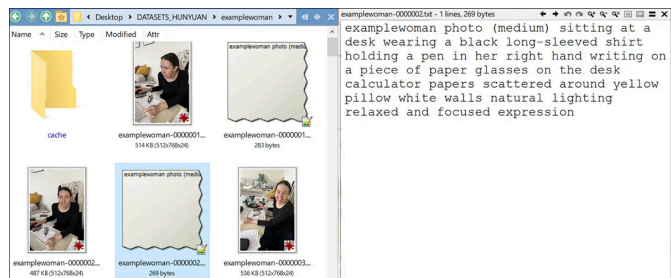
Select an image in the left-hand preview column and press CTRL+A to select all the images. Then press the Start Auto-Captioning button on the right:



You will see Taggui downloading models in the small CLI in the right-hand column, but only if this is the first time you have run the captioner. Otherwise you will see a preview of the captions.



Now, each photo has a corresponding .txt caption with a description of its image contents:



You can click *Advanced Options* in Taggui to increase the length and style of captions, but that is beyond the scope of this run-through.

Quit Taggui and let's move on to...

Latent Pre-Caching

To avoid excessive GPU load at training time, it is necessary to create two types of pre-cached files – one to represent the latent image derived from the images themselves, and another to evaluate a text encoding relating to caption content.

To simplify all three processes (2x cache + training), you can use interactive.BAT files that will ask you questions and undertake the processes when you have given the necessary information.

For the latent pre-caching, copy the following text into Notepad and save it as a.BAT file (i.e., name it something like *latent-precache.bat*), as earlier, ensuring that the file type in the drop down menu in the *Save As* dialogue is *All Files* (see image below):

```
@echo off

REM Activate the virtual environment

call C:\Users\[Your Profile Name]\Desktop\Musubi\musubi\Scripts\activate.bat

REM Get user input

set /p IMAGE_PATH=Enter the path to the image directory:

set /p CACHE_PATH=Enter the path to the cache directory:

set /p TOML_PATH=Enter the path to the TOML file:

echo You entered:

echo Image path: %IMAGE_PATH%

echo Cache path: %CACHE_PATH%

echo TOML file path: %TOML_PATH%

set /p CONFIRM=Do you want to proceed with latent pre-caching (y/n)?

if /i "%CONFIRM%"=="y" (

REM Run the latent pre-caching script

python C:\Users\[Your Profile Name]\Desktop\Musubi\musubi\musubi-tuner\cache_latents.py --dataset_config %TOML_PATH% --vae
C:\Users\[Your Profile Name]\Desktop\Musubi\musubi\musubi-tuner\models\pytorch_model.pt --vae_chunk_size 32 --vae_tiling

) else (

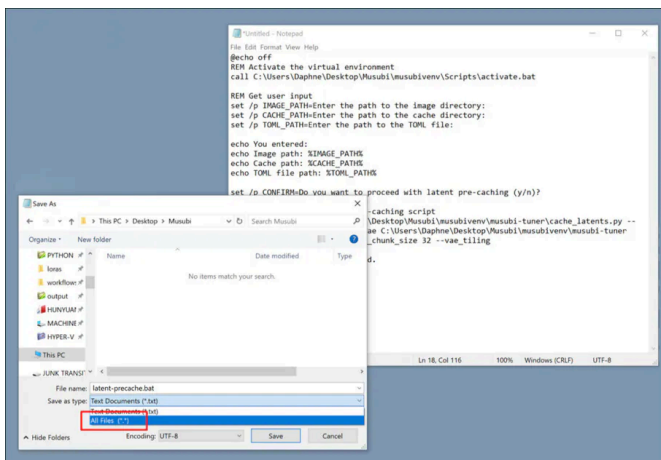
echo Operation canceled.

)

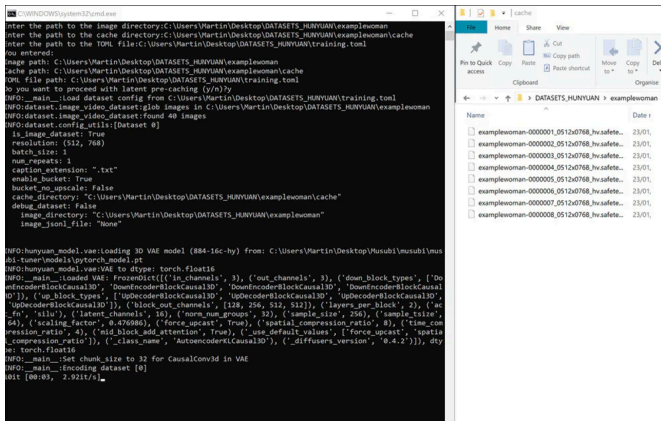
REM Keep the window open

pause
```

(Make sure that you replace [Your Profile Name] with your real Windows profile folder name)



Now you can run the.BAT file for automatic latent caching:



When prompted to by the various questions from the BAT file, paste or type in the path to your dataset, cache folders and TOML file.

Text Pre-Caching

We'll create a second BAT file, this time for the text pre-caching.

```
@echo off

REM Activate the virtual environment

call C:\Users\[Your Profile Name]\Desktop\Musubi\musubi\Scripts\activate.bat

REM Get user input

set /p IMAGE_PATH=Enter the path to the image directory:

set /p CACHE_PATH=Enter the path to the cache directory:

set /p TOML_PATH=Enter the path to the TOML file:

echo You entered:

echo Image path: %IMAGE_PATH%

echo Cache path: %CACHE_PATH%

echo TOML file path: %TOML_PATH%

set /p CONFIRM=Do you want to proceed with text encoder output pre-caching (y/n)?

if /i "%CONFIRM%"=="y" (

REM Use the python executable from the virtual environment

python C:\Users\[Your Profile Name]\Desktop\Musubi\musubi\musubi-tuner\cache_text_encoder_outputs.py --dataset_config
%TOML_PATH% --text_encoder1 C:\Users\[Your Profile Name]\Desktop\Musubi\musubi\musubi-
tuner\models\llava_llama3_fp16.safetensors --text_encoder2 C:\Users\[Your Profile Name]\Desktop\Musubi\musubi\musubi-
tuner\models\clip_l1.safetensors --batch_size 16

) else (

echo Operation canceled.

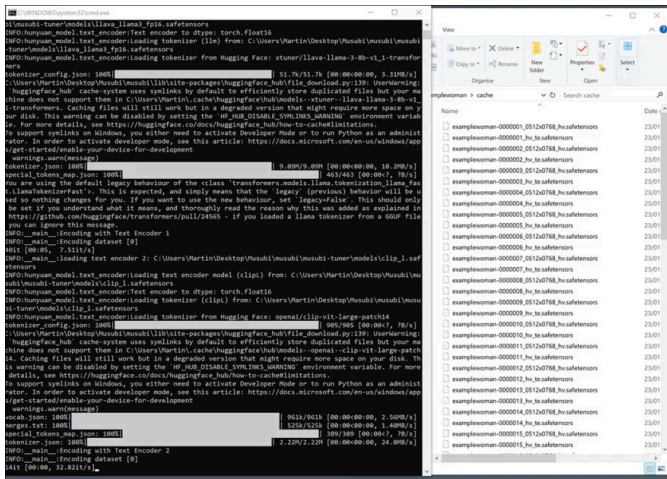
)

REM Keep the window open

pause
```

Replace your Windows profile name and save this as *text-cache.bat* (or any other name you like), in any convenient location, as per the procedure for the previous BAT file.

Run this new BAT file, follow the instructions, and the necessary text-encoded files will appear in the *cache* folder:



Training the Hunyuan Video Lora

Training the actual LoRA will take considerably longer than these two preparatory processes.

Though there are also multiple variables that we could worry about (such as batch size, repeats, epochs, and whether to use full or quantized models, among others), we'll save these considerations for another day, and a deeper look at the intricacies of LoRA creation.

For now, let's minimize the choices a little and train a LoRA on 'median' settings.

We'll create a third BAT file, this time to initiate training. Paste this into Notepad and save it as a BAT file, like before, as *training.bat* (or any name you please):

```
@echo off

REM Activate the virtual environment

call C:\Users\[Your Profile Name]\Desktop\Musubi\musubi\Scripts\activate.bat

REM Get user input

set /p DATASET_CONFIG=Enter the path to the dataset configuration file:

set /p EPOCHS=Enter the number of epochs to train:

set /p OUTPUT_NAME=Enter the output model name (e.g., example0001):

set /p LEARNING_RATE=Choose learning rate (1 for 1e-3, 2 for 5e-3, default 1e-3):

if "%LEARNING_RATE%"=="1" set LR=1e-3

if "%LEARNING_RATE%"=="2" set LR=5e-3

if "%LEARNING_RATE%"==" " set LR=1e-3

set /p SAVE_STEPS=How often (in steps) to save preview images:

set /p SAMPLE_PROMPTS=What is the location of the text-prompt file for training previews?

echo You entered:

echo Dataset configuration file: %DATASET_CONFIG%

echo Number of epochs: %EPOCHS%

echo Output name: %OUTPUT_NAME%

echo Learning rate: %LR%

echo Save preview images every %SAVE_STEPS% steps.

echo Text-prompt file: %SAMPLE_PROMPTS%

REM Prepare the command

set CMD=accelerate launch --num_cpu_threads_per_process 1 --mixed_precision bf16 ^

C:\Users\[Your Profile Name]\Desktop\Musubi\musubi\musubi-tuner\hv_train_network.py ^

--dit C:\Users\[Your Profile Name]\Desktop\Musubi\musubi\musubi-tuner\models\mp_rank_00_model_states.pt ^

--dataset_config %DATASET_CONFIG% ^
```

```

--sdpa ^
--mixed_precision bf16 ^
--fp8_base ^
--optimizer_type adamw8bit ^
--learning_rate %LR% ^
--gradient_checkpointing ^
--max_data_loader_n_workers 2 ^
--persistent_data_loader_workers ^
--network_module=networks.lora ^
--network_dim=32 ^
--timestep_sampling sigmoid ^
--discrete_flow_shift 1.0 ^
--max_train_epochs %EPOCHS% ^
--save_every_n_epochs=1 ^
--seed 42 ^
--output_dir "C:\Users\[Your Profile Name]\Desktop\Musubi\Output Models" ^
--output_name %OUTPUT_NAME% ^
--vae C:/Users/[Your Profile Name]/Desktop/Musubi/musubi/musubi-tuner/models/pytorch_model.pt ^
--vae_chunk_size 32 ^
--vae_spatial_tile_sample_min_size 128 ^
--text_encoder1 C:/Users/[Your Profile Name]/Desktop/Musubi/musubi/musubi-tuner/models/llava_llama3_fp16.safetensors ^
--text_encoder2 C:/Users/[Your Profile Name]/Desktop/Musubi/musubi/musubi-tuner/models/clip_l.safetensors ^
--sample_prompts %SAMPLE_PROMPTS% ^
--sample_every_n_steps %SAVE_STEPS% ^
--sample_at_first
echo The following command will be executed:
echo %CMD%
set /p CONFIRM=Do you want to proceed with training (y/n)?
if /i "%CONFIRM%"=="y" (
%CMD%
) else (
echo Operation canceled.
)
REM Keep the window open
cmd /k

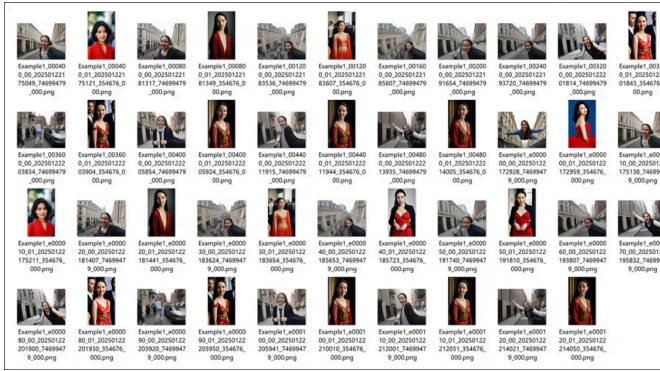
```

As usual, be sure to replace all instances of [Your Profile Name] with your correct Windows profile name.

Ensure that the directory C:\Users\[Your Profile Name]\Desktop\Musubi\Output Models\ exists, and create it at that location if not.

Training Previews

There is a very basic training preview feature recently enabled for Musubi trainer, which allows you to force the training model to pause and generate images based on prompts you have saved. These are saved in an automatically created folder called *Sample*, in the same directory that the trained models are saved.



To enable this, you will need to save at least one prompt in a text file. The training BAT we created will ask you to input the location of this file; therefore you can name the prompt file to be anything you like, and save it anywhere.

Here are some prompt examples for a file that will output three different images when requested by the training routine:



As you can see in the example above, you can put flags at the end of the prompt that will affect the images:

- w is **width** (defaults to 256px if not set, according to [the docs](#))
- h is **height** (defaults to 256px if not set)
- f is the **number of frames**. If set to 1, an image is produced; more than one, a video.
- d is the seed. If not set, it is random; but you should set it to see one prompt evolving.
- s is the number of steps in generation, defaulting to 20.

See [the official documentation](#) for additional flags.

Though training previews can quickly reveal some issues that might cause you to cancel the training and reconsider the data or the setup, thus saving time, do remember that every extra prompt slows down the training a little more.

Also, the bigger the training preview image's width and height (as set in the flags listed above), the more it will slow training down.

Launch your training BAT file.

Question #1 is 'Enter the path to the dataset configuration. Paste or type in the correct path to your TOML file.

Question #2 is 'Enter the number of epochs to train'. This is a trial-and-error variable, since it's affected by the amount and quality of images, as well as the captions, and other factors. In general, it's best to set it too high than too low, since you can always stop the training with Ctrl+C in the training window if you feel the model has advanced enough. Set it to 100 in the first instance, and see how it goes.

Question #3 is 'Enter the output model name'. Name your model! May be best to keep the name reasonably short and simple.

Question #4 is 'Choose learning rate', which defaults to 1e-3 (option 1). This is a good place to start, pending further experience.

Question #5 is 'How often (in steps) to save preview images. If you set this too low, you will see little progress between preview image saves, and this will slow down the training.

Question #6 is 'What is the location of the text-prompt file for training previews?'. Paste or type in the path to your prompts text file.

The BAT then shows you the command it will send to the Hunyuan Model, and asks you if you want to proceed, y/n.

Go ahead and begin training:

sample	343 MB	File Folder	Today	3:17 AM	-a-----
Example1-safetensors	225 MB	SAFETENSORS File	Yesterday	9:40 PM	-a-----
Example1-000119.safetensors	225 MB	SAFETENSORS File	Yesterday	9:37 PM	-a-----
Example1-000118.safetensors	225 MB	SAFETENSORS File	Yesterday	9:35 PM	-a-----
Example1-000117.safetensors	225 MB	SAFETENSORS File	Yesterday	9:33 PM	-a-----
Example1-000116.safetensors	225 MB	SAFETENSORS File	Yesterday	9:31 PM	-a-----
Example1-000115.safetensors	225 MB	SAFETENSORS File	Yesterday	9:29 PM	-a-----
Example1-000114.safetensors	225 MB	SAFETENSORS File	Yesterday	9:28 PM	-a-----
Example1-000113.safetensors	225 MB	SAFETENSORS File	Yesterday	9:26 PM	-a-----
Example1-000112.safetensors	225 MB	SAFETENSORS File	Yesterday	9:24 PM	-a-----
Example1-000111.safetensors	225 MB	SAFETENSORS File	Yesterday	9:22 PM	-a-----
Example1-000110.safetensors	225 MB	SAFETENSORS File	Yesterday	9:19 PM	-a-----
Example1-000109.safetensors	225 MB	SAFETENSORS File	Yesterday	9:17 PM	-a-----
Example1-000108.safetensors	225 MB	SAFETENSORS File	Yesterday	9:15 PM	-a-----
Example1-000107.safetensors	225 MB	SAFETENSORS File	Yesterday	9:13 PM	-a-----
Example1-000106.safetensors	225 MB	SAFETENSORS File	Yesterday	9:11 PM	-a-----
Example1-000105.safetensors	225 MB	SAFETENSORS File	Yesterday	9:09 PM	-a-----
Example1-000104.safetensors	225 MB	SAFETENSORS File	Yesterday	9:07 PM	-a-----
Example1-000103.safetensors	225 MB	SAFETENSORS File	Yesterday	9:05 PM	-a-----
Example1-000102.safetensors	225 MB	SAFETENSORS File	Yesterday	9:03 PM	-a-----
Example1-000101.safetensors	225 MB	SAFETENSORS File	Yesterday	9:02 PM	-a-----
Example1-000100.safetensors	225 MB	SAFETENSORS File	Yesterday	8:59 PM	-a-----
Example1-000099.safetensors	225 MB	SAFETENSORS File	Yesterday	8:56 PM	-a-----
Example1-000098.safetensors	225 MB	SAFETENSORS File	Yesterday	8:54 PM	-a-----

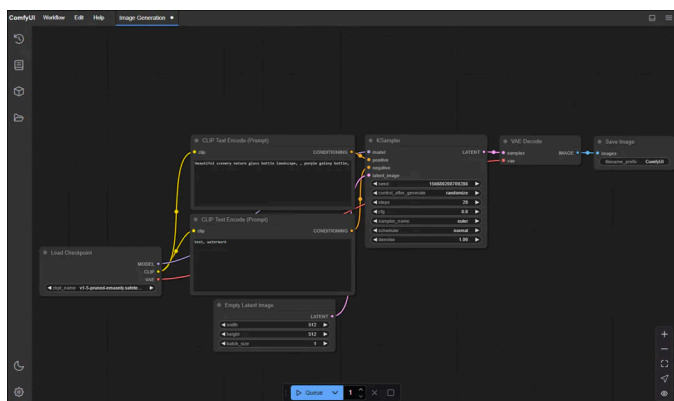
This saving frequency can be changed by the user to save more or less frequently, as desired, by amending the `--save_every_n_epochs [N]` number in the training BAT file. If you added a low figure for saves-per-steps when setting up training with the BAT, there will be a high number of saved checkpoint files.

Which Checkpoint to Choose?

As mentioned earlier, the earliest-trained models will be most flexible, while the later checkpoints may offer the most detail. The only way to test for these factors is to run some of the LoRAs and generate a few videos. In this way you can get to know which checkpoints are most productive, and represent the best balance between flexibility and fidelity.

ComfyUI

The most popular (though not the only) environment for using Hunyuan Video LoRAs, at the moment, is [ComfyUI](https://github.com/comfyanonymous/ComfyUI), a node-based editor with an elaborate Gradio interface that runs in your web browser.



Source: <https://github.com/comfyanonymous/ComfyUI>

Installation instructions are straightforward and [available at the official GitHub repository](https://github.com/comfyanonymous/ComfyUI) (additional models will have to be downloaded).

Converting Models for ComfyUI

Your trained models are saved in a (diffusers) format that is not compatible with most implementations of ComfyUI. Musubi is able to convert a model to a ComfyUI-compatible format. Let's set up a BAT file to implement this.

Before running this BAT, create the `C:\Users\[Your Profile Name]\Desktop\Musubi\CONVERTED\` folder that the script is expecting.

```
@echo off

REM Activate the virtual environment

call C:\Users\[Your Profile Name]\Desktop\Musubi\musubi\Scripts\activate.bat

:START

REM Get user input

set /p INPUT_PATH=Enter the path to the input Musubi safetensors file (or type "exit" to quit):

REM Exit if the user types "exit"

if /i "%INPUT_PATH%"=="exit" goto END
```

```

REM Extract the file name from the input path and append 'converted' to it
for %%F in ("%INPUT_PATH%") do set FILENAME=%%~nF

set OUTPUT_PATH=C:\Users\[Your Profile Name]\Desktop\Musubi\Output Models\CONVERTED\%FILENAME%_converted.safetensors

set TARGET=other

echo You entered:

echo Input file: %INPUT_PATH%

echo Output file: %OUTPUT_PATH%

echo Target format: %TARGET%

set /p CONFIRM=Do you want to proceed with the conversion (y/n)?

if /i "%CONFIRM%"=="y" (

REM Run the conversion script with correctly quoted paths

python C:\Users\[Your Profile Name]\Desktop\Musubi\musubi\musubi-tuner\convert_lora.py --input "%INPUT_PATH%" --output
"%OUTPUT_PATH%" --target %TARGET%

echo Conversion complete.

) else (

echo Operation canceled.

)

REM Return to start for another file

goto START

:END

REM Keep the window open

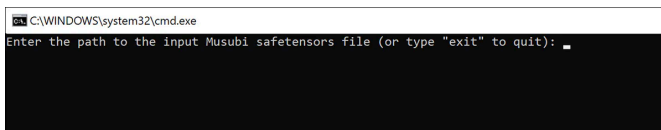
echo Exiting the script.

pause

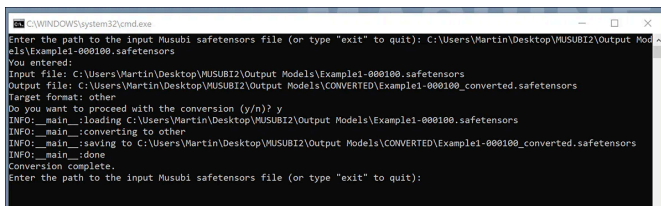
```

As with the previous BAT files, save the script as 'All files' from Notepad, naming it *convert.bat* (or whatever you like).

Once saved, double-click the new BAT file, which will ask for the location of a file to convert.



Paste in or type the path to the trained file you want to convert, click y, and press enter.



After saving the converted LoRA to the *CONVERTED* folder, the script will ask if you would like to convert another file. If you want to test multiple checkpoints in ComfyUI, convert a selection of the models.

When you have converted enough checkpoints, close the BAT command window.

You can now copy your converted models into the *models\loras* folder in your ComfyUI installation.

Typically the correct location is something like:

C:\Users\[Your Profile Name]\Desktop\ComfyUI\models\loras\

Creating Hunyuan Video LoRAs in ComfyUI

Though the node-based workflows of ComfyUI seem complex initially, the settings of other more expert users can be loaded by dragging an image (made with the other user's ComfyUI) directly into the ComfyUI window. Workflows can also be exported as JSON files, which can be imported manually, or dragged into a ComfyUI window.

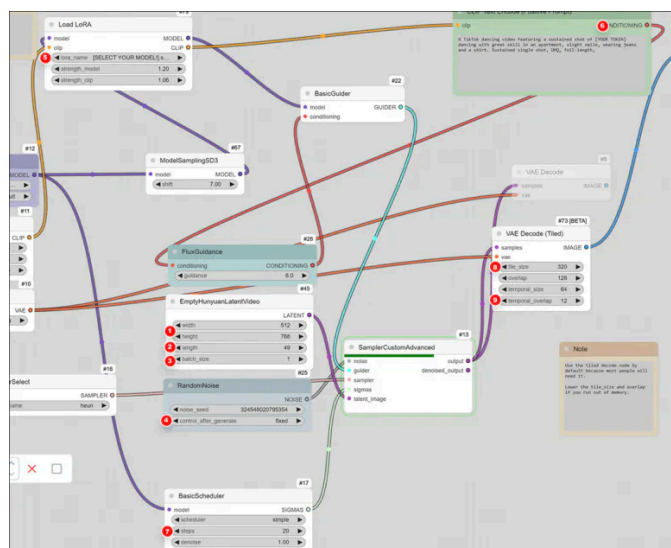
Some imported workflows will have dependencies that may not exist in your installation. Therefore install [ComfyUI-Manager](#), which can fetch missing modules automatically.



Source: <https://github.com/ltddrdata/ComfyUI-Manager>

To load one of the workflows used to generate videos from the models in this tutorial, download [this JSON file](#) and drag it into your ComfyUI window (though there are far better workflow examples available at the various Reddit and Discord communities that have adopted Hunyuan Video, and my own is adapted from one of these).

This is not the place for an extended tutorial in the use of ComfyUI, but it is worth mentioning a few of the crucial parameters that will affect your output if you download and use the JSON layout that I linked to above.



1) Width and Height

The larger your image, the longer the generation will take, and the higher the risk of an out-of-memory (OOM) error.

2) Length

This is the numerical value for the number of frames. How many seconds it adds up to depend on the frame rate (set to 30fps in this layout). You can convert seconds>frames based on fps [at Omniculator](#).

3) Batch size

The higher you set the batch size, the quicker the result may come, but the greater the burden of VRAM. Set this too high and you may get an OOM.

4) Control After Generate

This controls the random seed. The options for this sub-node are *fixed*, *increment*, *decrement* and *randomize*. If you leave it at *fixed* and do not change the text prompt, you will get the same image every time. If you amend the text prompt, the image will change to a limited extent. The *increment* and *decrement* settings allow you to explore nearby seed values, while *randomize* gives you a totally new interpretation of the prompt.

5) Lora Name

You will need to select your own installed model here, before attempting to generate.

6) Token

If you have trained your model to trigger the concept with a token, (such as *'example-person'*), put that trigger word in your prompt.

7) Steps

This represents how many steps the system will apply to the diffusion process. Higher steps may obtain better detail, but there is a ceiling on how effective this approach is, and that threshold can be hard to find. The common range of steps is around 20-30.

8) Tile Size

This defines how much information is handled at one time during generation. It's set to 256 by default. Raising it can speed up generation, but raising it too high can lead to a particularly frustrating OOM experience, since it comes at the very end of a long process.

9) Temporal Overlap

Hunyuan Video generation of people can lead to 'ghosting', or unconvincing movement if this is set too low. In general, the current wisdom is that this should be set to a higher value than the number of frames, to produce better movement.

Conclusion

Though further exploration of ComfyUI usage is beyond the scope of this article, community experience at Reddit and Discords can ease the learning curve, and there are several [online guides](#) that introduce the basics.

First published Thursday, January 23, 2025

Writer on machine learning, domain specialist in human image synthesis. Former head of research content at Metaphysic.ai, until its dissolution into DNEG's Brahma.ai.

Portfolio site: martinanderson.ai

Contact: martin@martinanderson.ai



Discover More