Pixelmania

Table of Contents

Preamble

1.1 Disclaimer

This document and the software is a work in progress, Pixelmania reserves the right to update and change the procedures and processes described in this documentation at any time. We will try our best to inform you when these changes happen with the hopes of minimizing any impact on production.

1.2 Copyright

In addition to those names set forth on this page, the names of other actual companies and products mentioned in this User Guide (including and not limited to, those set forth below) may be the trademarks or service marks, or registered trademarks or service marks, of their respective owners in the United States and/or other countries, states and/or provinces. No association with any company or product is intended or inferred by the mention of its name in this User Guide.

Nuke is a trademark of The Foundry Visionmongers Ltd.
Linux is a registered trademark of Linus Torvalds.
CentOS is a trademark of Red Hat, Inc.
Rocky Linux is a trademark of Rocky Enterprise Software Foundation (RESF).
"Microsoft (R) Windows (R)" is a registered trademark of Microsoft Corporation in the United States and other countries.
Reprise License Manager is a trademark of Reprise Software, Inc.
Vimeo is a trademark of Vimeo, Inc.
Nukepedia is a trademark of OHUfx Ltd.

1.3 Acknowledgements and citations

We like to thank the authors of the following research work and publications:

```
@article{zhong2023clearer,
  title={Clearer Frames, Anytime: Resolving Velocity Ambiguity in Video Frame Interpolation},
  author={Zhong, Zhihang and Krishnan, Gurunandan and Sun, Xiao and Qiao, Yu and Ma, Sizhuo and
Wang, Jian},
  journal={arXiv preprint arXiv:2311.08007},
  year={2023}
}

@inproceedings{zhang2023extracting,
  title={Extracting Motion and Appearance via Inter-Frame Attention for Efficient Video Frame Interpolation},
  author={Zhang, Guozhen and Zhu, Yuhan and Wang, Haonan and Chen, Youxin and Wu, Gangshan and
Wang, Limin},
  booktitle={Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition
(CVPR)},
  year={2023}
}
```

2. Installation

2.1 System Requirements

- 64 bit Linux Operating System (for example CentOS 7.x, Rocky8.x or Rocky9.x), or
- 64 bit Windows Operating System (for example Windows 10 Pro or Windows 11 Pro), or
- 64 bit macOS Operating System (with support for Apple Silicon M processors (arm64), for example macOS Sonoma 14.x)
- Nuke13.1 or higher on Linux, Nuke14.1 or higher on Windows, Nuke15.1 or higher on macOS, with a valid license.
  Nuke Indie is supported (to run the plugin in Nuke Indie, you have to use the latest version of Nuke14.x, Nuke15.x, Nuke16.x or Nuke17.x).

2.2 Plugin Installation

NNRetime relies on the standard Nuke process for finding and using plug-ins. Hence you only need to unzip the files in the downloaded zip archive (that matches your OS, Nuke version and CUDA version) to a directory of your own choice.

You then append that full path to the **NUKE_PATH** environment variable. This will tell Nuke to load the supplied **init.py**, **menu.py** and **NNRetime.so** (**NNRetime.dll** on Windows or **NNRetime.dylib** on macOS) files at startup. To read more about the **NUKE_PATH** environment variable, visit the documentation page by Foundry:
https://learn.foundry.com/nuke/content/comp_environment/configuring_nuke/defining_nuke_plugin_path.html

An alternative way to using the **NUKE_PATH** environment variable to specify for Nuke where to look for plugins, is to use the python command:
nuke.pluginAddPath("/full/path/to/plugin/install/folder")

This can be added to your **init.py** file located in your user's "**.nuke**" folder. Read more at:
https://learn.foundry.com/nuke/developers/13.0/pythonreference/_autosummary/nuke.pluginAddPath.html

Below are examples of how to add the install path to the **NUKE_PATH** environment variable on different platforms:

Linux:
The simplest way to create this environment variable is to put a single line like this in your **.bashrc** file in your user's home folder:
export NUKE_PATH=/full/path/to/plugin/install/folder
You can easily edit this file by starting a new terminal and using the command:
nano .bashrc

macOS:
The simplest way to create this environment variable is to put a single line like this in your **.zshrc** file in your user's home folder:
export NUKE_PATH=/full/path/to/plugin/install/folder
You can easily edit this file by starting a new terminal and using the command:
nano .zshrc

Windows:
In Windows 10/11, you have to add the environment variable using the system preferences. The easiest way to get to the right place is using the search bar on the Start Menu. Search for "environment variables", and choose "**Edit the system environment variables**". Add the "**NUKE_PATH**" variable and set the value to the full path to the folder where the plugin is installed.

For a more detailed guide of how to do this, you can search on Google for "windows 10 set environment variable permanently", and you will get links to pages like:
https://www.architectryan.com/2018/08/31/how-to-change-environment-variables-on-windows-10/
https://www.opentechguides.com/how-to/article/windows-10/113/windows-10-set-path.html

2.3 Environment Variables

PIXELMANIA_NNRETIME_DONT_RENDER_USING_GUI_LICENSES
The default behavior of the node in batch/render mode is to look for dedicated NNRetime render licenses. If it can't find a render license, it will instead look for a GUI license. If it finds a GUI license, it will use it to render the node's output. This behavior is not preferred if you only got a few GUI licenses available, i.e. you don't want the render farm to steal the artists' GUI licenses. To enforce the node to not use GUI licenses, please set the environment variable PIXELMANIA_NNRETIME_DONT_RENDER_USING_GUI_LICENSES to the value of "1". Please note that if you already got a site license, you don't want to set this environment variable since you already got an unlimited number of floating GUI licenses, so it's fine for the node to use those for rendering as well as in GUI mode.

PIXELMANIA_NNRETIME_FORCE_RENDER_USING_GUI_LICENSES
The default behavior of the node in batch/render mode is to look for dedicated NNRetime render licenses. If you want to force the plugin to directly checkout a GUI license instead of looking for a render license, please set the environment variable PIXELMANIA_NNRETIME_FORCE_RENDER_USING_GUI_LICENSES to the value of "1".
If you are relying on rendering using GUI licenses (**such as when you have a site license**), setting this variable will actually make the render startup process a bit faster because the node can directly checkout the correct license type. You will also get less log clutter in the license server because of directly checking out the correct license type.

PIXELMANIA_SUPPRESS_KERNEL_WARNINGS
The NNRetime node will print out a warning during creation of the node if it's being dependent on JIT-compiled processing kernels. Basically this is happening if you have a very modern GPU that is using a compute capability that is higher than the native compute capability support of the specific build of NNRetime that you are running. The warning is there to inform you (the first time you run the plugin) that the kernels will get JIT-compiled which will take about half an hour (and appear like the computer is just hanged, even if it's heavily processing). The node will also check the environment variable CUDA_CACHE_MAXSIZE to see that it's set with a good value that will make it possible for the JIT-compilation to succeed (please see below for more info about this).
With all the above in mind, the environment variable PIXELMANIA_SUPPRESS_KERNEL_WARNINGS can turn these warning messages off if it's set to the value of "1".

This variable does NOT apply to the macOS build (since the macOS build is accelerated by MPS instead of CUDA).

CUDA_CACHE_MAXSIZE
If you are using one of the CUDA11.8 versions of the NNRetime plugin (as an example), and you are having a modern compute capability 12.0 GPU, i.e. the Blackwell generation of cards (for example RTX5080) or later, you need to set the CUDA CACHE for JIT-compiled kernels to a high value using the **CUDA_CACHE_MAXSIZE environment variable**. We recommend setting it to **4294967296** because that is the maximum limit of the setting (4Gb).

You need to do this because you are using an older version of CUDA that was released before the 12.0 cards were released. You will then rely on JIT-compiling the internal PTX code instead of using the natively compiled kernels for your GPU architecture. These kernels will only need to be compiled once, and will be saved to the CUDA CACHE in your user's home directory. But for this operation to succeed the cache size needs to be set higher than the default value.
The kernel compilation can take around 30 minutes or so (usually less), and then you are set to use the plugin without needing to do this again. Be aware of this need though, since the process is silent and might be experienced as a strange hanging without any progress bars and such.

If you don't want to do all this, you should download and use one of the CUDA12.8 compiled versions of NNRetime instead. These will run directly and potentially even faster on your modern GPU.
The only downside then is if you are using Nuke13.x/Nuke14.x and Foundry's own machine learning toolset like the CopyCat node. Nuke 13.x is only shipped with support for CUDA10.1/cuDNN8.0.5, Nuke 14.0 is only shipped with support for CUDA11.1/ cuDNN8.4.1 and Nuke 14.1 is only shipped with support for CUDA11.8/cuDNN8.4.1. If you are using NNRetime with another version of CUDA/cuDNN you can not run them in the same script without problems/crashing.

When it comes to the more modern releases of Nuke, i.e. Nuke14.1/15.0/15.1/15.2/16.0/16.1, we are only shipping NNRetime compiled against the exact same CUDA version and cuDNN version as Nuke is using internally. This is because Foundry have updated to a CUDA version that natively supports the RTX40xx series of NVIDIA GPUs. NNRetime for Nuke 14.1/15.0/15.1/15.2/16.0/16.1 is hence using CUDA 11.8 and cuDNN 8.4.1 to fully be compatible with Nuke's own AIR tools.

When it comes to the newly released Nuke17.0, we are only shipping NNRetime compiled against the exact same CUDA version and cuDNN version as Nuke is using internally. This is because Foundry have updated to a CUDA version that natively supports the RTX50xx series of NVIDIA GPUs. NNRetime for Nuke17.0 is hence using CUDA 12.8 and cuDNN 9.7.1 to fully be compatible with Nuke's own AIR tools.

To see a chart of CUDA compatibilities with different NNRetime builds, please visit this page on our website: https://pixelmania.se/cuda-compatibility-chart/

This environment variable does NOT apply to the macOS build (since the macOS build is accelerated by MPS instead of CUDA).

3. Usage and Knob Reference

3.1 Usage of the Node

The NNRetime node performs neural network based frame interpolation and retiming. Given the two source frames that surround a requested sub-frame time, it runs a pre-trained deep learning model to synthesize the in-between frame. This makes it possible to produce smooth slow-motion, to change the speed of a clip, or to remap timing in any way you like – all without the usual stepping or ghosting you get from a simple frame blend. Conceptually the node is similar to classic optical-flow retimers such as Nuke's own Kronos and OFlow. The big difference is that NNRetime synthesizes the in-between

frames using a neural network that has been trained on large amounts of footage, which often holds up better on difficult motion, occlusions and fine detail. You simply feed it an RGB(A) image sequence on the single input, choose a timing mode, and the node produces the retimed result.

There are two timing modes available (set with the “timing” knob):
- **Output Speed** – retime the clip by a constant speed factor. This is the mode to reach for when you want slow-motion or fast-motion. The speed is set with the “speed” knob.
- **Frame** – map a specific, keyframeable source frame to each frame on the timeline. Animate the “frame” knob to remap the timing of the clip arbitrarily.

The node processes the channels you select with the “channels” knob (“all” by default). This means that you can process every layer of a multichannel stream, which you can then write out using a standard Write node set to “channels: all”.
NNRetime is a commercial plugin. Without a checked-out license the node runs in a free demo mode: it still renders in the Nuke GUI, but the output is watermarked with noise, and batch/command-line rendering is disabled. Once a valid license is found the watermark is removed and batch rendering is enabled (please see the “About” tab and the Frequently Asked Questions for more about licensing).

3.2 Knob Reference

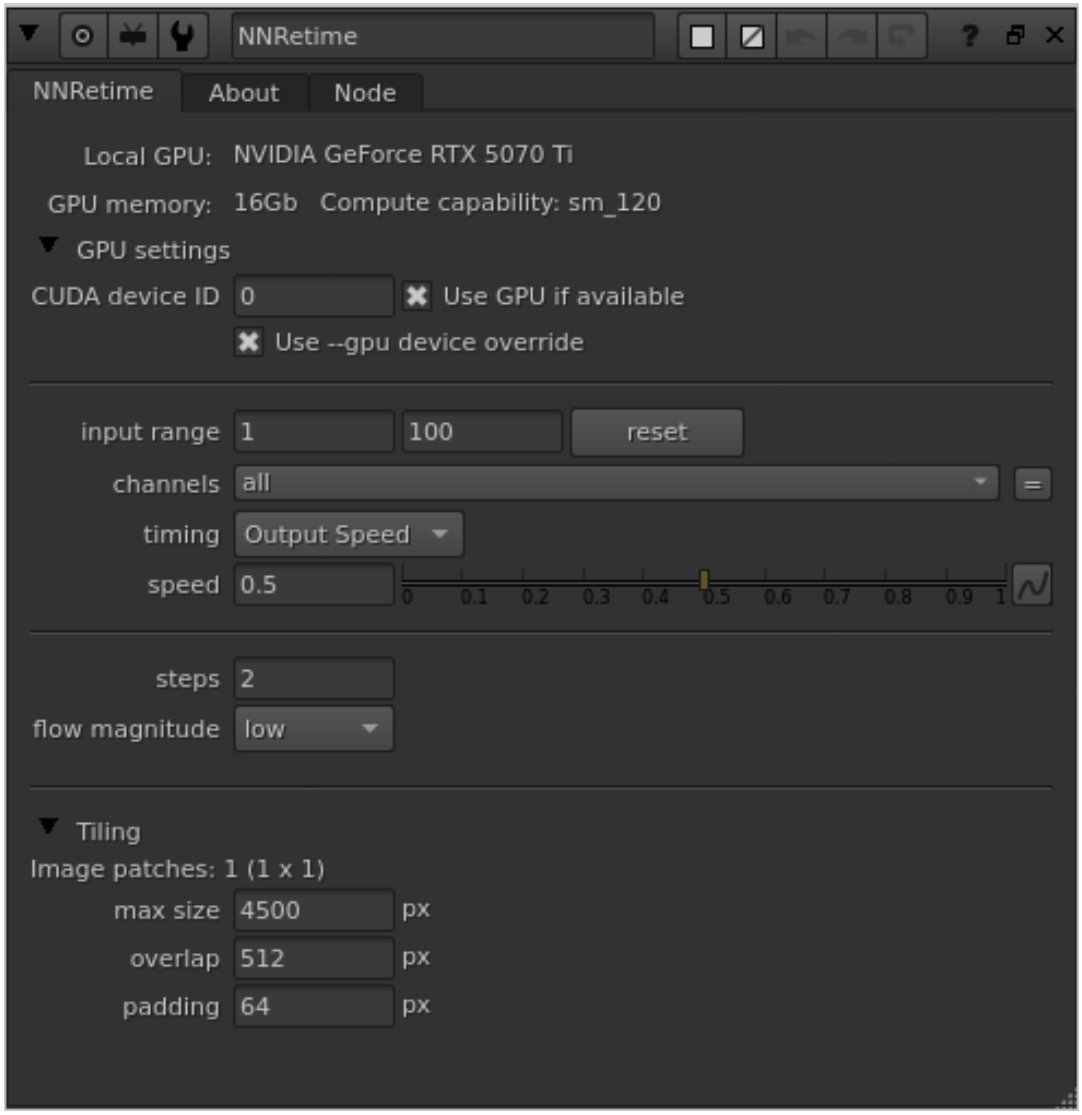
Here we go through each knob that is available in NNRetime. A few of the knobs are related to being able to retime very large resolution sequences using the limited VRAM on the graphics card (see the section below called “Tiling”). Even a small resolution input requires a fair amount of memory, so larger frames may need to be split up into several passes/patches to fit into memory. This is all done and handled transparently “under the hood” so the artist can focus on more important things. You might need to tweak the settings though depending on the use case and available hardware on your workstation.

Local GPU / GPU memory
At the very top of the panel the node prints the name of the GPU it has detected and how much memory it has available (on macOS this is the unified memory of the Apple Silicon chip). On a CUDA build, if your GPU’s compute capability is not supported by the current build it will be shown in red and the GPU processing options will be disabled.

CUDA device ID
This knob is only present if you’ve installed a GPU version of the plugin. This knob specifies what GPU to use in the system by its CUDA device ID. It’s only relevant if you got multiple GPUs installed in the system. The default value is 0, which is the default CUDA processing device, usually the fastest/most modern GPU installed. Please refer to the output of running the command “nvidia-smi” in a terminal for retrieving the info of the specific GPU device IDs you have assigned to your GPUs in your particular system.

This knob does NOT exist in the macOS build (since the macOS build is accelerated by MPS instead of CUDA).

Use GPU if available
This knob is only present if you’ve installed a GPU version of the plugin. This knob is not changing the output of the plugin, but rather how the result is calculated. If it is “on” (which



is the default) the algorithm will run on the GPU hardware of the workstation, and if it's "off" the algorithm will run on the normal CPU of the workstation. If the plugin can't detect a CUDA compatible GPU, this knob will automatically be disabled/greyed out. This knob is similar to the one you'll find in Foundry's own GPU accelerated plugins like for example the ZDefocus node.

We highly recommend always having this knob "on", since the algorithm will run A LOT faster on the GPU than on the CPU. Worth knowing is that the interpolation model ships in 16-bit (half float) precision; when you render on the CPU the node automatically converts the model to 32-bit float so it produces correct results, but this CPU path is single-threaded and very slow. For any real work you want to be on the GPU. For speed references of processing, please download and test run the plugin on your own hardware/system.

Use --gpu device override
When this knob is set to "on" (which is the default), then node will not blindly use the device ID set in the knob "CUDA device ID" but will rather adhere to the --gpu command line parameter. This means that NNRetime will behave like Foundry's own GPU accelerated nodes, and render using the specified GPU in the Nuke batch render command. This enables you to have a setup in the GUI, when you are working with the

node as an artist, but still be able to farm the exact same Nuke script on different boxes with different GPU configurations.

input range
The input range (the two value fields) defines the first and last frame of the input clip that the node is allowed to read from when interpolating. It is auto-filled from the connected input's frame range when the node is created, but you can override it manually if needed. Setting this correctly matters because the node fetches the two source frames that surround each requested sub-frame; a sub-frame time can only be interpolated if both of its surrounding source frames fall inside this range.

reset
This button, next to the input range, re-reads the frame range from the currently connected input and writes it back into the input range fields. Use it whenever you change or re-connect the input.

channels
This is a classic input channels knob, where you define what channels/layers the node should interpolate. The default is "all", which will process every channel/layer of the input. But you can also set it to "rgb" to only process that if needed. The model works on three channels at a time: for a layer with fewer than three channels (such as a single depth channel) the channel is replicated; for a layer with a fourth channel (typically alpha) the model runs a second pass so the alpha gets interpolated too; any channels beyond the fourth, and any layers you have not selected, pass through unchanged from the nearest source frame.

timing
This drop down menu sets how source frames are mapped to output frames for the retime. There are two modes:

- **Output Speed** – retimes the whole clip by a constant speed factor, set with the "speed" knob below. The node derives the output frame range from the input range and the speed factor (using offset/range math that is compatible with NukeX's Kronos).
- **Frame** – lets you specify which source frame should appear at the current timeline frame, using the keyframeable "frame" knob. The output frame range follows the keyframe range of that knob.

speed
This knob is used in the "Output Speed" timing mode. It controls the speed of the retime relative to the output range. Values below 1.0 slow the clip down (for example 0.5 gives you half speed, i.e. 2x slow motion), and values above 1.0 speed the movement up. The default value is 0.5.

frame
This knob is used in the "Frame" timing mode. It specifies the source frame to show at the current frame on the timeline. The knob is keyframeable, so you can animate it to retime the clip in any way you like – whenever the value lands between two source frames, NNRetime synthesizes the in-between frame for you.

steps
Maximum subdivision depth used when locating the exact sub-frame time for the interpolation. Higher values provide more accurate frame positioning but require more computation. The default value is 2. If you see artifacts, try values between 3-5. The valid input range for the knob is 1-6.

flow magnitude
Higher setting catches higher motion magnitudes, i.e. larger/longer movement of objects, but can lead to less local detail. Recommended to be set only as high as necessary. Slightly increases computation time. Possible settings are “low”, “medium” and “high” with “low” being the default value.

Tiling
These knobs live in a closed “Tiling” group. For frames that are too large to fit in GPU memory in a single pass (for example 8K), the node automatically splits the frame into overlapping tiles/patches, runs each one through the model, and cross-fade blends them back together. Below the chosen “max size” this all happens in a single pass with no overhead.

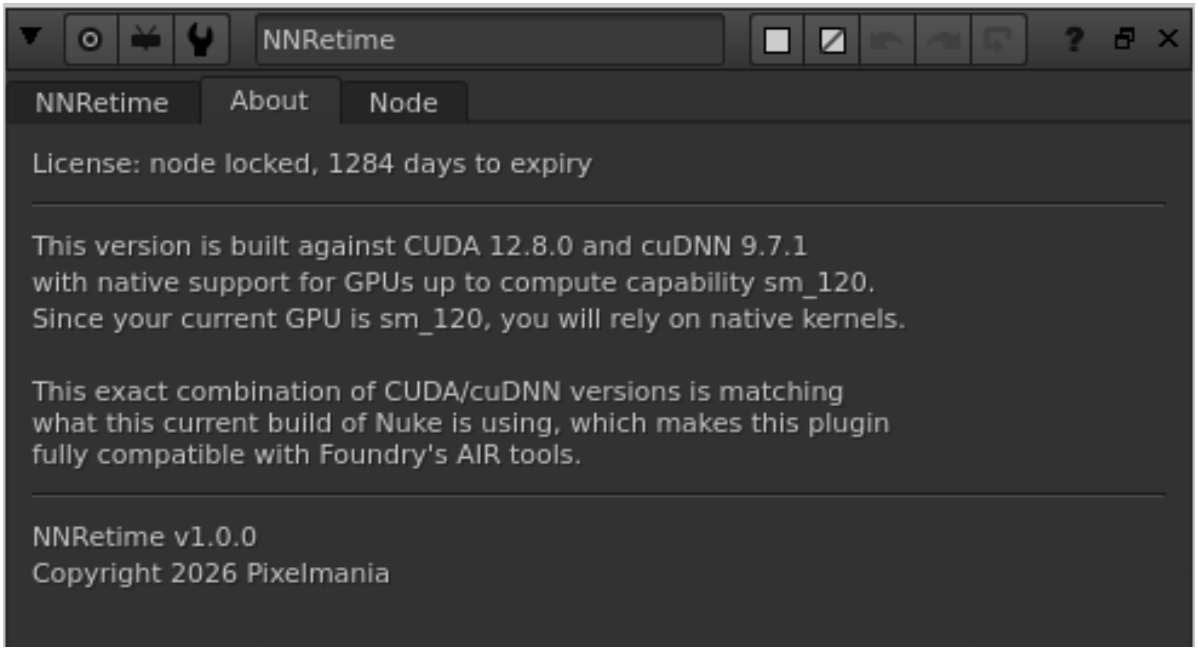
Image patches
This is a status read-out (shown above the tiling controls) that tells you how many tiles/patches the current input will be split into for the chosen max size and overlap. It updates live; a value of a single patch means the whole frame is processed in one pass.

max size
Max size sets the maximum length, in both dimensions, that every image patch can have, measured in pixels. Frames larger than this are processed in tiles to keep within the available GPU memory. The default value is 4500. If you run into a GPU out-of-memory error, lower this value (or render on the CPU); if you have a lot of VRAM available you can raise it so more of the frame is processed in one pass.

overlap
Since the plugin will run multiple patches through the neural network, there will be lots of patch edges present. The edges don’t get as satisfying results as a bit further into the patch, so all the patches are processed with some spatial overlap. The overlap knob sets the number of pixels of that overlap. The default value is 512.

padding
The padding is very connected to the overlap above. While the overlap sets the total amount of pixels the patches overlap, the padding then reduces the actual cross-fading area where the patches are blended, so the result doesn’t use the very edge pixels of each patch at all. It is also measured in pixels, and the default value is 64.

The “About” tab:
There is an additional tab called “About” in the knob panel for NNRetime. This tab prints out some additional info that can be very helpful for trouble shooting. The first section is printing out what type of license that is currently in use, and also how many more days it’s valid for.



The next section (if you are running a GPU based build) is printing out what CUDA and cuDNN this version is built against. It also lists what compute capability your current GPU is supporting and what max compute capability the current build of the plugin is natively supporting. It also writes out if your current configuration will use native or JIT-compiled kernels. In addition to that, it also writes out if the current plugin build is matching what versions of CUDA and cuDNN that the current Nuke build is using, and hence if it's compatible with the AIR tools or not.

Finally it's listing what version of the plugin that is running.
(the screenshot above is just a snapshot of how this info can look, captured on one of our development workstations)

4. Frequently Asked Questions

Is Nuke Indie supported?
Yes, Nuke Indie is supported. To have the plugin working in Nuke Indie you have to use the latest version available from Foundry of Nuke14.x, Nuke15.x, Nuke16.x or Nuke17.x.

How are licenses consumed?
All of Pixelmania's licenses are per host, i.e. you can have multiple jobs using NNRetime on the same host and only use a single license. If you query the license server it may report multiple handles for those jobs, but it's still only using a single license token.

We do support dedicated render licenses as well. The default behavior when rendering using Nuke's batch mode (i.e. command line rendering), is to check out render licenses. If it can't find a render license, it will instead try and check out a GUI license. This behavior is preferred if you got a node locked license, or if you got a site license. It is not preferred if you have only a few GUI licenses and then a bunch of render licenses. To stop the node

from using GUI licenses when rendering, you can set the environment variable PIXELMANIA_NNRETIME_DONT_RENDER_USING_GUI_LICENSES to the value of “1” (please see more about this in the section “2.3 Environment Variables”).

To instead force the node to use GUI licenses when rendering, you can set the environment variable PIXELMANIA_NNRETIME_FORCE_RENDER_USING_GUI_LICENSES to the value of “1”. This can be good in a farm environment and you are having a site license. (please see more about this in the section “2.3 Environment Variables”).

Do you offer educational discount?
As of now, we are not offering any default educational discount. We’ve chosen to sell our licenses for a low price instead to accommodate most people and companies to afford a license anyway. But you are of course always welcome to contact us directly and state your case.

Am I able to transfer my NNRetime license to another machine?
Yes, please fill out the License Transfer Form available at this address:
https://www.pixelmania.se/license-transfer-form
The form should be filled in, signed, scanned and emailed to licenses@pixelmania.se, and we will send you a new license file as soon as we can.

Do you really need CUDA installed to be able to use the GPU?
You don’t need to have the CUDA Toolkit installed to use our plugins and have them GPU accelerated. All of our plugin downloads come bundled with the needed CUDA and cuDNN libraries from NVIDIA. What you need to have them work correctly is a supported GPU (please see below) and a modern enough NVIDIA graphics driver version installed.

You can download the latest graphics drivers from NVIDIA’s website here:
https://www.nvidia.com/Download/index.aspx

What NVIDIA graphic card architectures are supported?
The currently supported CUDA architectures are the following compute capabilities (see https://en.wikipedia.org/wiki/CUDA):

- 5.0 (Maxwell)
- 5.2 (Maxwell)
- 6.0 (Pascal)
- 6.1 (Pascal)
- 7.0 (Volta)
- 7.5 (Turing)
- 8.6 (Ampere)*
- 8.9 (Ada Lovelace)**
- 12.0 (Blackwell)***

* The Ampere architecture, i.e. the NVIDIA RTX30xx series of cards, works with all CUDA variants of NNRetime (**CUDA10.1, CUDA11.1, CUDA11.2, CUDA11.8)**. The CUDA11.1/ CUDA11.2/CUDA11.8 versions got native support and will work directly. The CUDA10.1 versions will need to compile the CUDA kernels from the PTX code. Please see more info above in the environment variables section.

** The Ada Lovelace architecture, i.e. the NVIDIA RTX40xx series of cards, works with all CUDA variants of NNRetime (**CUDA10.1, CUDA11.1, CUDA11.2, CUDA11.8)**. The CUDA11.8 versions got native support and will work directly. The CUDA10.1/CUDA11.1/ CUDA11.2 versions will need to compile the CUDA kernels from the PTX code. Please see more info above in the environment variables section.

*** The Blackwell architecture, i.e. the NVIDIA RTX50xx series of cards, works with all CUDA variants of NNRetime (**CUDA10.1, CUDA11.1, CUDA11.2, CUDA11.8, CUDA12.8)**. The CUDA12.8 version got native support and will work directly. The CUDA10.1/ CUDA11.1/CUDA11.2/CUDA11.8 versions will need to compile the CUDA kernels from the PTX code. Please see more info above in the environment variables section.

The above section does NOT apply to the macOS build (since the macOS build is accelerated by MPS instead of CUDA).

Why is the plugin so large?
This is a result of including a lot of needed static libraries for neural network processing, and also for supporting lots of different graphic cards for acceleration. The plugin could have been much smaller if all these pieces of software used were required to be installed as dependencies on the system instead. That would kind of defeat the nice ecosystem of having self contained plugins that work simply by themselves, hence the plugin need to be a rather large file.
Since the release of Nuke14.1/15.0/15.1 our plugins are linked directly against the bundled PyTorch version that is released together with Nuke. This makes the binaries much smaller than before, even up to around 2Gb smaller.

I’m having problems running NNRetime in the same Nuke environment as KeenTools FaceTracker/FaceBuilder
We have noticed that trying to load both NNRetime and FaceTracker in the same Nuke environment makes the plugins clash with each other. The error you get is that NNRetime is not finding the GOMP_4.0 symbols in the libgomp.so shared library that is loaded. This is because both NNRetime and FaceTracker are using the libgomp.so shared library, but KeenTools have decided to ship their plugin with a version of this library. This specific library version they are shipping is a rather old version of the library. So what happens is that NNRetime is trying to use the version from FaceTracker but it’s expecting a newer version, and it’s failing. The fix we have found working well is simply to delete the libgomp.so file that is shipped with KeenTools. You probably already got a newer libgomp.so file installed on your system. If that is the case both NNRetime and FaceTracker will pick up the newer libgomp.so shared library from the system, and since this library is backwards compatible everything will work just fine. If you haven’t got libgomp.so installed on your system, you need to install it to run the plugins. How that is

done is of course dependent on your operating system and such. On CentOS you can install it by simply running the command “yum install libgomp“.

I’m having problems/crashing when using NNRetime together with Foundry’s AIR nodes in Nuke13.x
You are highly likely using the CUDA11.2/cuDNN8.1.0 build or the CUDA11.8/cuDNN8.4.1 build if this is happening. These builds of NNRetime are for modern GPUs like the RTX3080 or RTX4080 cards, and will work fast and nice except for when you need to process NNRetime output together with for example CopyCat nodes. If you need to support this, you have to use the CUDA10.1/cuDN8.0.5 build instead. Please see more info about this above in the environment variables section.

I’m having problems/crashing when using NNRetime together with Foundry’s AIR nodes in Nuke14.0
You are highly likely using the CUDA11.8/cuDNN8.4.1 build if this is happening. This build of NNRetime is for modern GPUs like the RTX4080 card, and will work fast and nice except for when you need to process NNRetime output together with for example CopyCat nodes. If you need to support this, you have to use the CUDA11.1/cuDN8.4.1 build instead. Please see more info about this above in the environment variables section.

5. Change Log

v1.0.0 - June 2026 - Initial release
- Neural network frame interpolation and retiming, synthesizing in-between frames from the two surrounding source frames.
- Two timing modes: “Output Speed” (retime by a speed factor) and “Frame” (map a keyframeable source frame to each timeline frame).
- Support for Nuke 13.1+ on Linux, Nuke 14.1+ on Windows and Nuke 15.1+ on macOS.
- Support for Nuke Indie.
- GPU acceleration with CUDA on Linux/Windows and MPS (Metal) on Apple Silicon, with the option of CPU only processing.
- Per-channel processing with a “channels” knob (RGBA by default, optionally all layers, with alpha handled by a second interpolation pass).
- Automatic large-image tiling for frames too large to fit in GPU memory, with cross-fade blending.
- Dedicated render-license support and Nuke --gpu device override for farm rendering.