



Polyhedron Suite

User Documentation · v372

by Polyhedron

The complete ComfyUI toolset in one manual: unified media I/O — a pinned-folder loader with live tile browser, batch modes, trimming and masks, paired with one Save node for stills, sequences, videos and masks — group-aware LoRA management built for workflows that run 10-25 LoRAs at once — and segmented prompt encoding that keeps a long prompt readable without paying for it in tokens.

[Part I Media Loader & Save](#) · [Part II LoRA Stack](#) · [Part III CLIP Text Encode](#) · [Part IV Sampler](#) · [Part V Upscale & Interpolate](#) · [Part VI Attention & grading](#)

© Polyhedron · All rights reserved
civitai.com/user/Polyhedron_AI · patreon.com/c/polyhedron_ai

Contents

Part I — Media Loader & Save

At a glance — every output pin	4
1 · Folders & files (Media Loader)	6
2 · Selection & views	9
3 · Video	13
4 · Audio	15
5 · Batch	17
6 · Outputs & masks	21
7 · Polyhedron Save	22

Part II — LoRA Stack

1 · Overview	25
2 · Installation	25
3 · Polyhedron LoRA Stack	26
4 · Polyhedron LoRA Engine	36
5 · Polyhedron LoRA Inspector	37
6 · Connecting the outputs	40
7 · Tips and recommended settings	40
8 · Preview images and trigger words	41
9 · Polyhedron Token Counter	42
10 · Polyhedron Select Model Switch	44
11 · Polyhedron WAN Bridge	46
12 · Polyhedron Wan Frame Inflate (T2I LoRA fix)	48
13 · Sigma Nodes	50
14 · Polyhedron Merge Analyzer	55

Part III — CLIP Text Encode

1 · Overview	60
2 · Segments — the prompt in parts	61
3 · Comments — headings that cost nothing	62
4 · The negative	65
5 · External text	66
6 · The footer — words, characters, tokens	68

Part IV — Sampler

1 · Overview	70
2 · Single — the everyday case	71
3 · High + Low — two experts, one node	72
4 · Live preview — watching it work	74
5 · Putting it together	75

Part V — Upscale & Interpolate

1 · Fast Upscale — size without invention	77
2 · Power Upscale — the thorough one	78
3 · Power Upscale — watching it work	80
4 · Interpolate — the frames in between	81
5 · Choosing between them	82

Part VI — Attention & grading

1 · Attention — the machinery underneath	84
2 · Attention — choosing a backend	85
3 · NAG — guidance without CFG	86
4 · Filter — grading you can see	87
5 · Audio Stretch — sound that follows the picture	88

Appendix

Running on a network (--listen)	89
---------------------------------	----



Polyhedron Media Loader & Save

User Documentation · v665

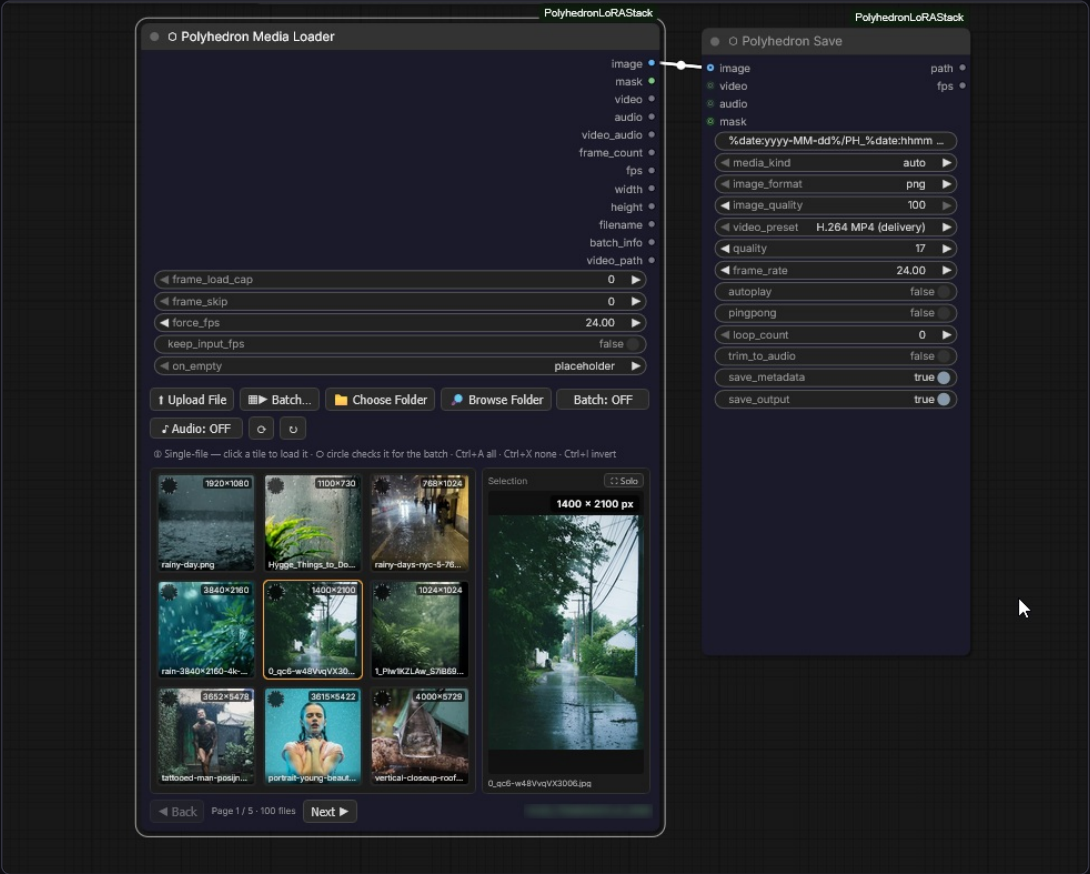
by Polyhedron

The pack's unified media I/O: a pinned-folder loader with a live tile browser, batch modes, video/audio trimming and a mask-aware image path — paired with one Save node for stills, sequences, videos and masks.

© Polyhedron · All rights reserved
civitai.com/user/Polyhedron_AI · patreon.com/c/polyhedron_ai

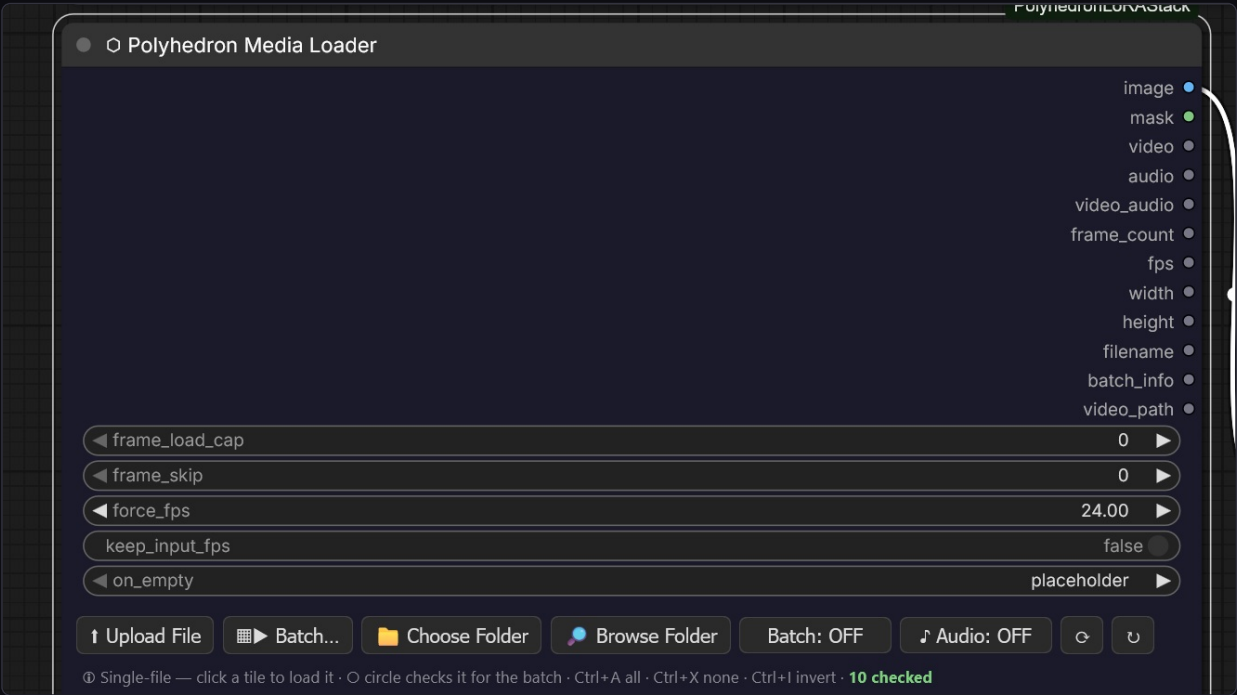
The **Media Loader** is the pack's one-stop input node: images, image batches, videos and audio from any folder on your machine, with a visual tile browser, single-file and batch modes, video/audio trimming and a mask-aware image path. The **Save** node is its counterpart on the way out: one node for stills, image sequences, videos (with muxed audio) and masks, with presets, metadata embedding and date-based file naming.

They are built to work as a pair — what Save writes, the Loader reads back, including masks.



Media Loader wired into Save: tile grid, selection panel and the pinned folder

At a glance — every output pin



The Media Loader's output pins, top to bottom

Pin	Type	Carries
image	IMAGE [N,H,W,3]	The decoded frames — $N = 1$ for a still, the frame count for a video or batch. An audio-only pick emits a small placeholder frame so the graph keeps running.
mask	MASK [N,H,W]	The still's alpha (opaque $\rightarrow 0$, like core LoadImage). A grayscale still without alpha carries its luminance as the mask, white $\rightarrow 1$ — that is what makes a saved mask load back as the same mask. Plain video: blank; alpha-bearing video (VP9-alpha WEBM, ProRes 4444, transparent GIF): a real per-frame mask. Details in section 6.
video	VIDEO	A single video file losslessly with its own audio ; a trimmed video is re-encoded from the sliced frames with its sound cut to the same window; an image batch or a still-with-audio becomes a synthesized clip. <code>None</code> only when there are no frames.
audio	AUDIO	The paired audio, decoded and trimmed (section 4). <code>None</code> when nothing is paired.
video_audio	VIDEO	The visual's frames muxed with the paired audio — wire it into a video save for an mp4 with a sound track. <code>None</code> when nothing is paired.
frame_count	INT	<code>1</code> for a still.
fps	FLOAT	<code>0.0</code> for a still; native or forced for a video.
width / height	INT	The decoded frame size.
filename	STRING	The loaded file's name; empty in sequence mode.
batch_info	STRING	The live batch counter (section 5): Batch <code>0023 / 1334</code> per firing, Batch <code>1334</code> frames (one pass) , or Single: <code><name></code> .
video_path	STRING	The loaded file's full path — feeds the Batch Pipeline Source; empty in modes without a single source file.

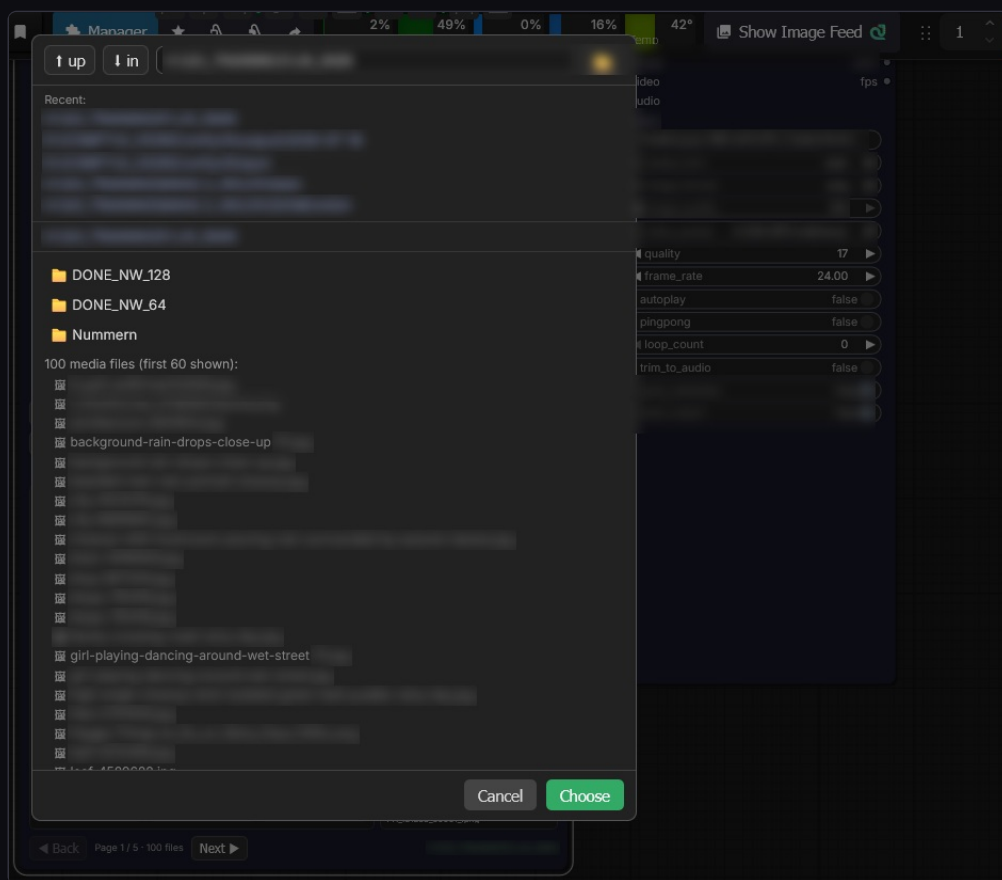
The five widget rows below the pins are the video levers (`frame_load_cap` , `frame_skip` , `force_fps` , `keep_input_fps` — section 3) and the empty-state switch (`on_empty` — section 3).

1 · Folders & files (Media Loader)

The loader reads from a **pinned folder** — any folder on the machine running ComfyUI, not just the ComfyUI input directory. The current pin is shown at the bottom right of the node.

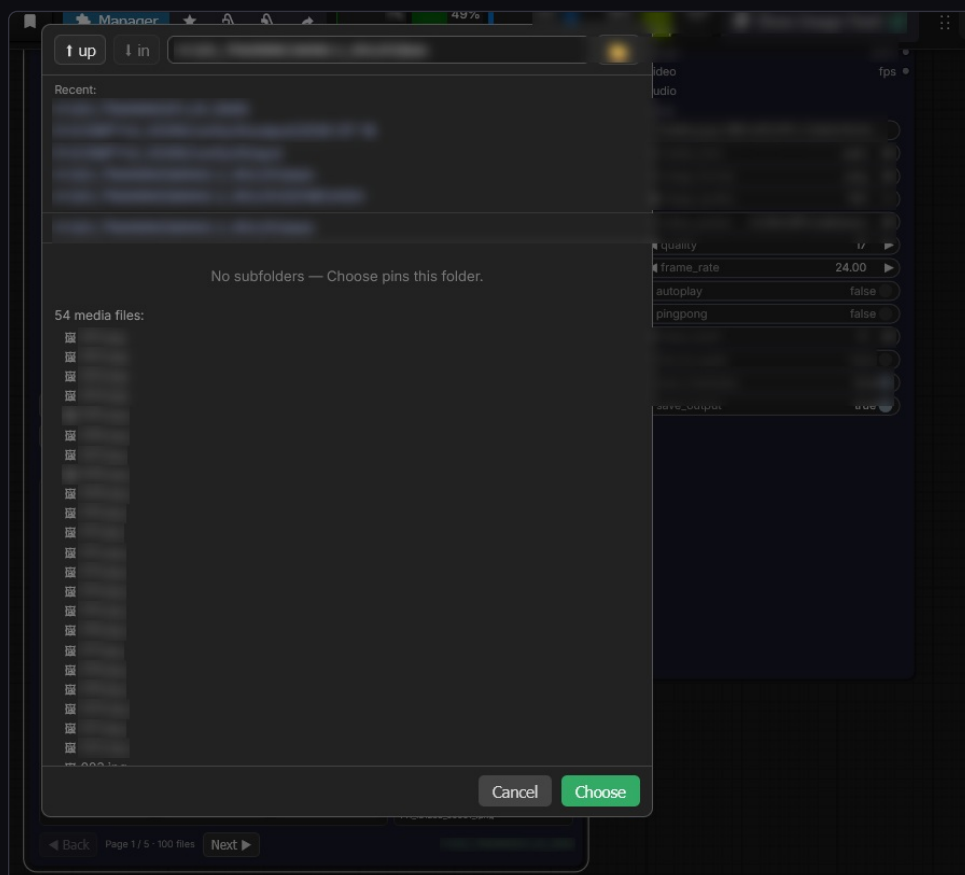
Picking a folder

📁 **Choose Folder** opens the in-canvas picker:

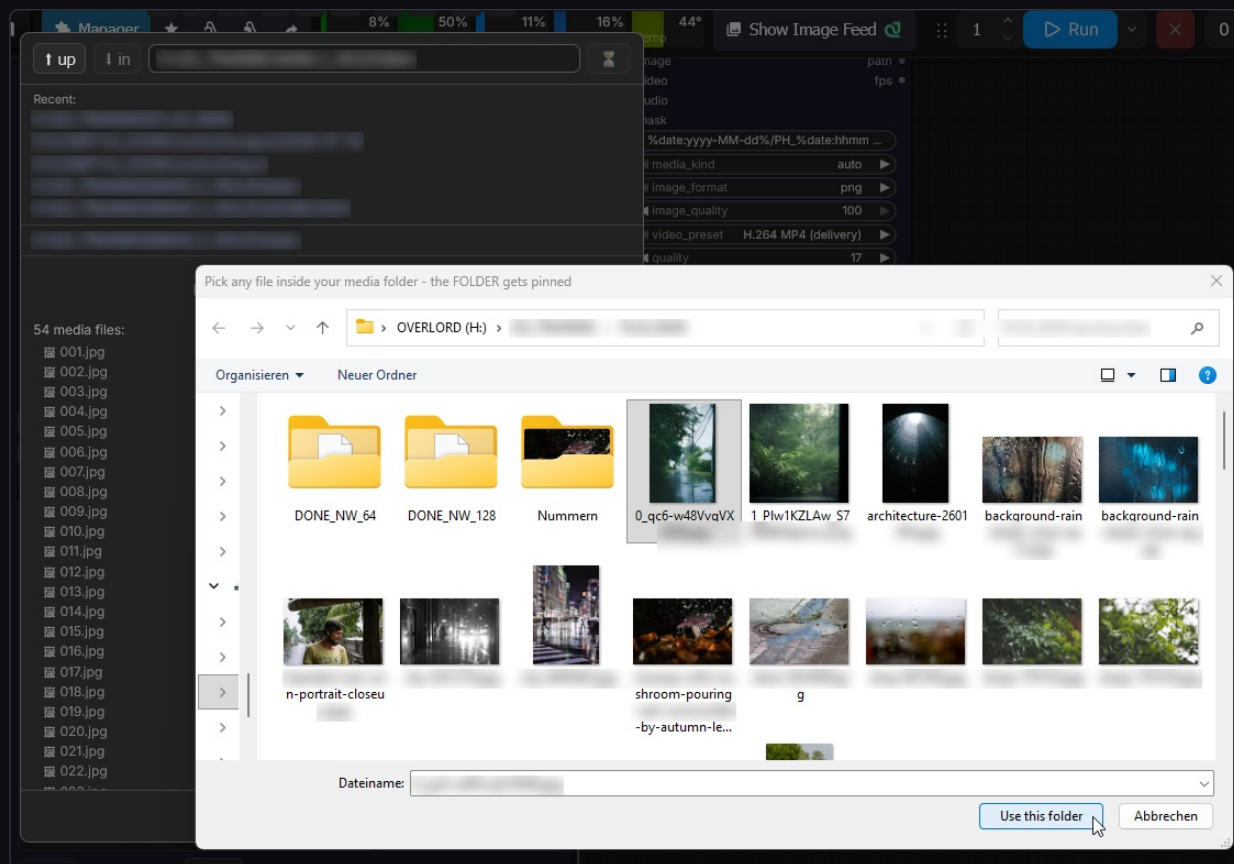


In-canvas picker: recents, subfolder rows and the media-file peek

When the current folder has **no subfolders**, the picker says so — **Choose** then pins the folder you are looking at:



- The picker **peeks into the current folder**: below the subfolders it lists the folder's own media files (count + sorted names with kind icons), so you can see what a pin would load before you commit.
- The **address bar is live**: type or paste a full path and hit **Choose** — the typed path is validated and pinned directly, no Enter-then-click detour. Unreadable paths say so and keep the picker open.
- The **yellow folder button** inside the picker opens the native Windows dialog — as a **file** dialog: you see the folder's contents like in Explorer, click **any file** inside your target folder, and that file's **folder** gets pinned and the file itself is selected and loaded right away. The dialog's OK is the confirmation; the pick applies immediately. (Windows cannot show files in its folder-pick mode — that's an OS limit, which is exactly why the dialog works file-first.)



Native Windows dialog in file mode — pick any file, its folder gets pinned

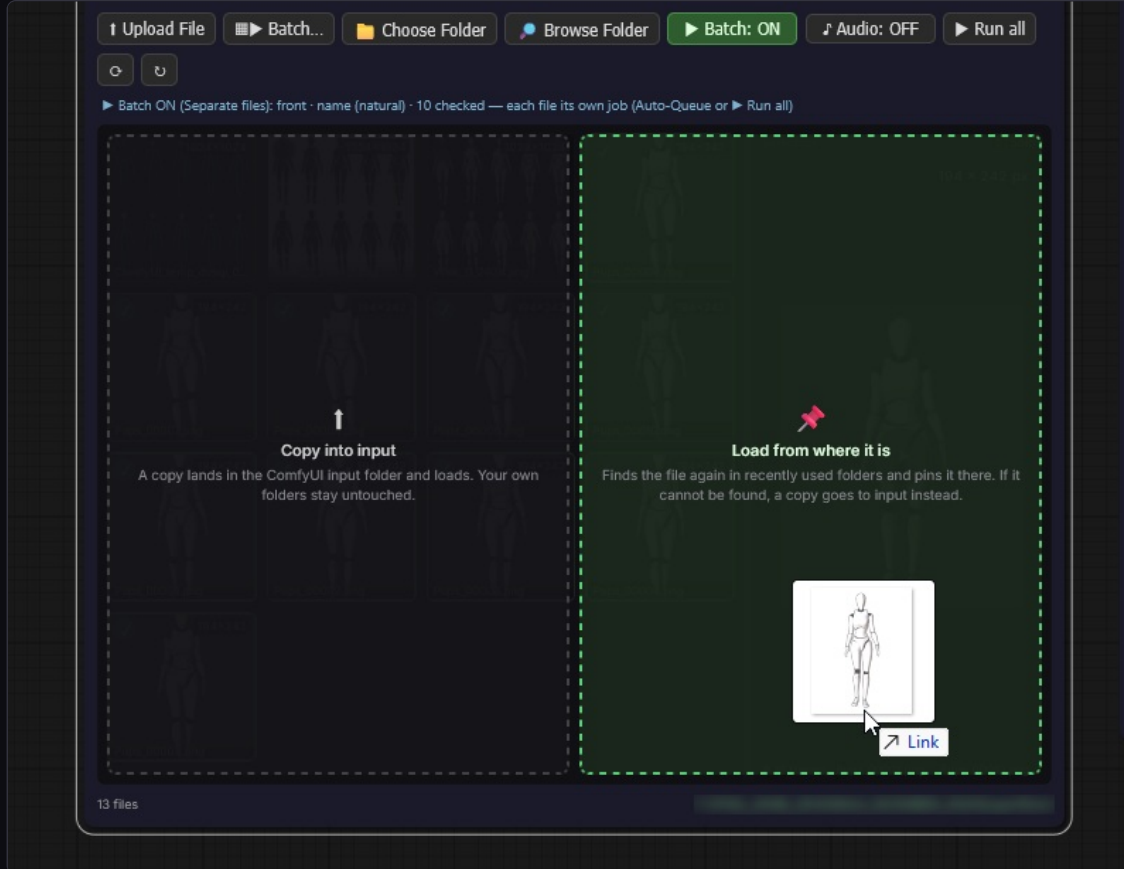
🔗 **Browse Folder** opens the large, filterable list view for visually hunting through long folders — see *Live preview on hover* in section 2.

Staying fresh

- 🔄 **Re-read** re-reads the pinned folder from disk — new files appear, your current page and checkmarks stay.
- **Silent auto-refresh**: when the ComfyUI window regains focus (a render finished, you copied files in from Explorer), the loader probes the folder in the background and re-renders only if something actually changed. No flicker, no clicks.
- 🔄 **Reset** jumps back to the ComfyUI input folder.

Getting files in

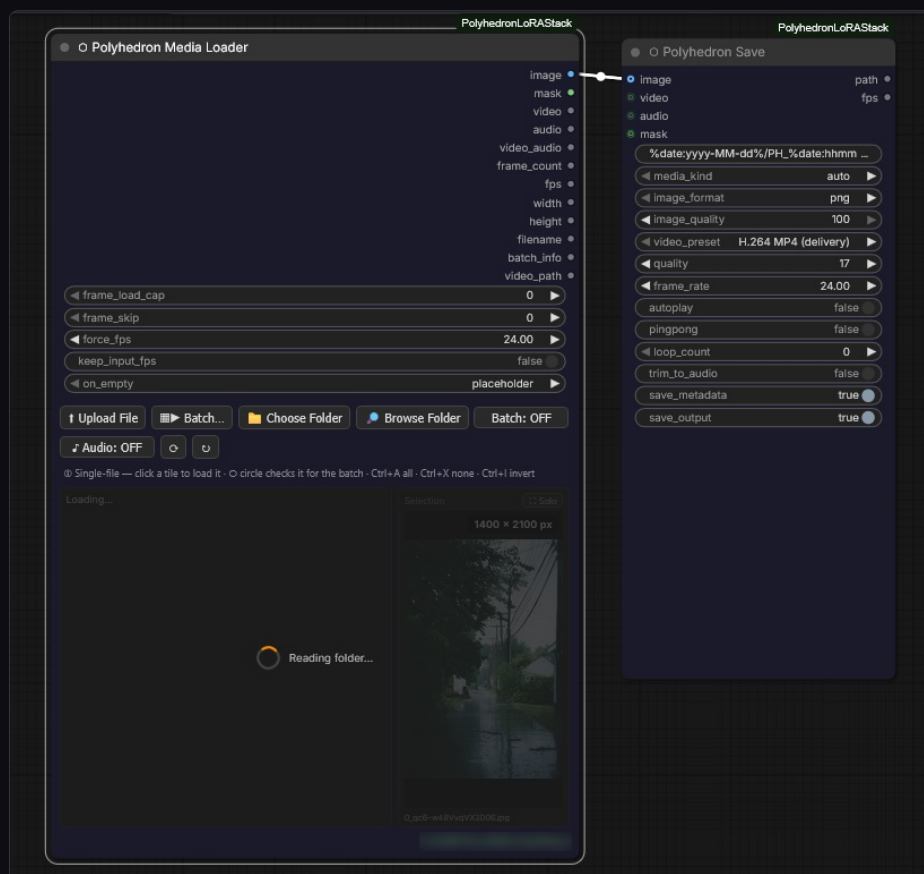
- ⬆ **Upload File** uploads into the ComfyUI input folder.
- **Drag & drop** raises **two drop zones** over the node — aiming at one is the choice, nothing needs to be known or held down:



The two drop zones during a drag — the aimed zone lights up

- **↑ Copy into input** (left): a copy lands in the ComfyUI input folder and loads. Your own folders are never written to.
- **✳ Load from where it is** (right): selects the file and pins the folder it already lives in — the whole folder appears in the grid, nothing is copied. When the drag carries the file's path (a dropped path text, a `file:///` URI, or the desktop client) that path is used directly; otherwise the file is **found again** by name, size and timestamp in recently used folders. If it cannot be found, a copy goes to input instead — and the node says so.

The zones exist only while a drag is in flight; the node gains no permanent buttons. The zone under the pointer lights up, so the aim is confirmed before release. A multi-file drop copies every file and loads the first — the status line names the count. - A **busy ring** with a phase label ("Uploading...", "Reading folder...") covers the node while slow operations run — visible in every view, including Solo.

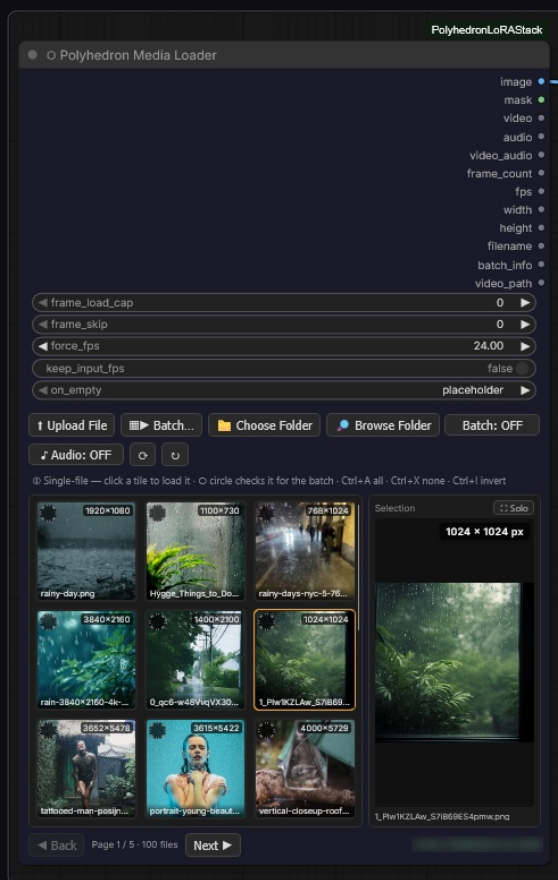


2 · Selection & views

The lower half of the node is the **tile grid**: one thumbnail per media file in the pinned folder, each with a **resolution badge** (width × height), a **kind icon** and the file name. Large folders are split into pages — the **pager** at the bottom shows the current page and the total file count, ◀ **Back** / **Next** ▶ flip through.

Single-file mode

Click a tile and that file is **loaded immediately** — it becomes the node's output and appears in the **Selection panel** on the right, with a large preview, the pixel dimensions and the file name. This is the everyday mode: one file in, one file out.



Tile grid with a loaded file in the Selection panel

Live preview on hover

Nothing has to be loaded to be judged — pointing at it is enough:

- **Images** pop a large preview over the node, with the pixel size as a badge. Move off the tile and it vanishes.

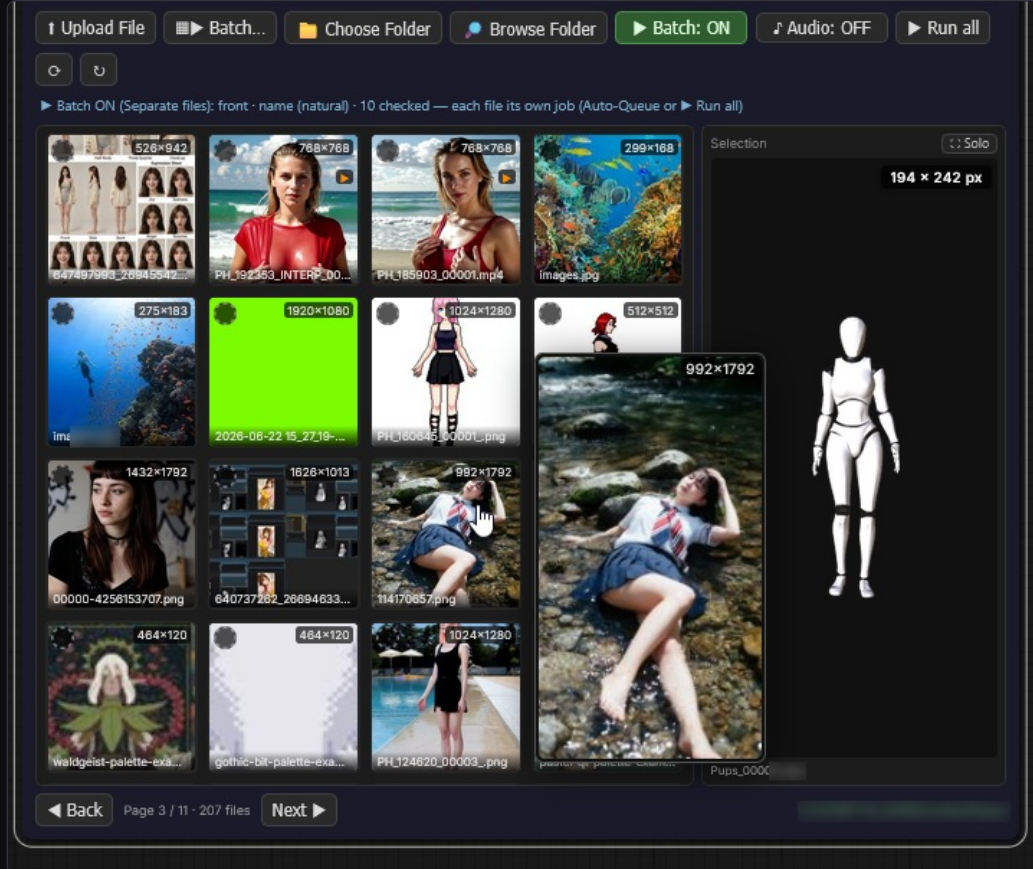
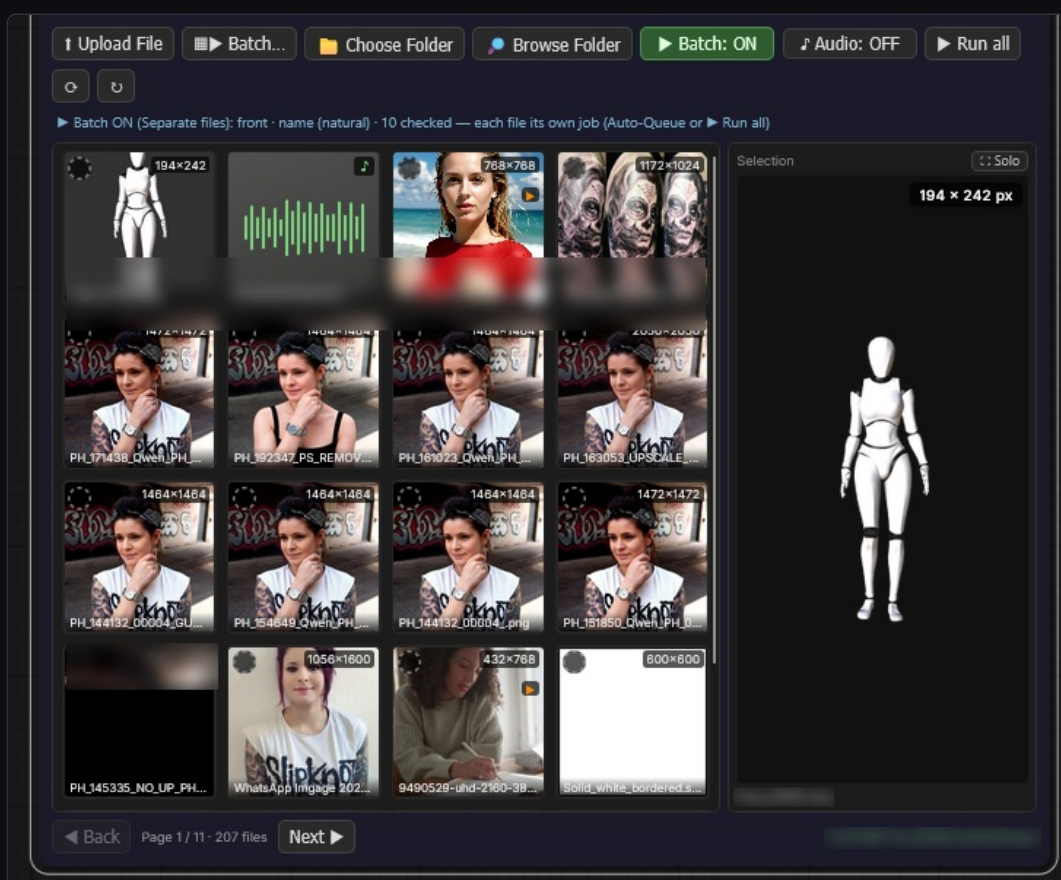


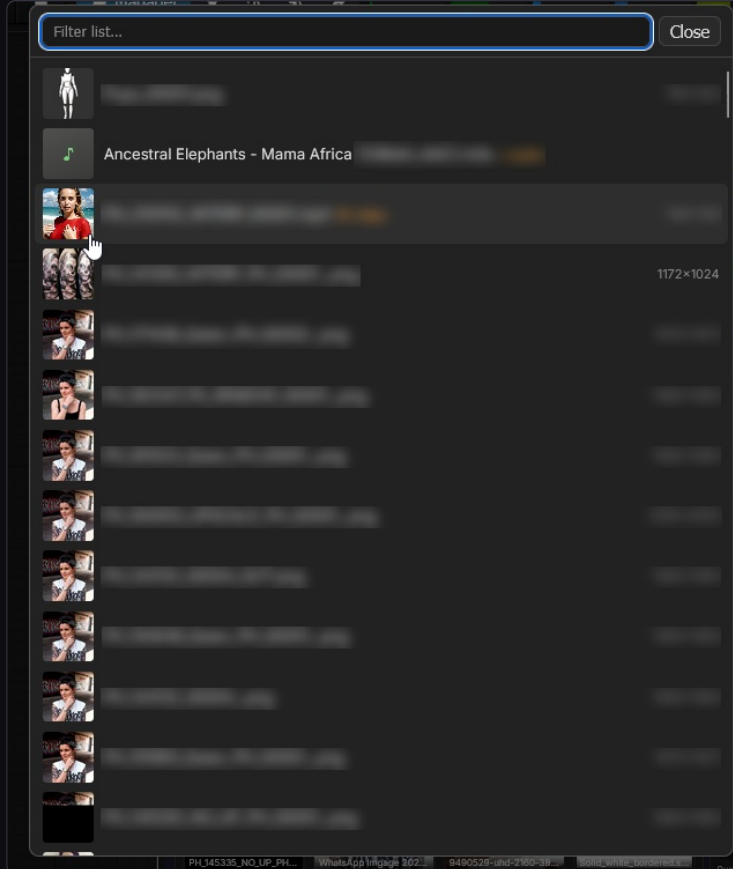
Image hover: a large pop-up with its own size badge

- **Videos play right inside their tile** — muted, looping, starting the moment the pointer arrives and stopping the moment it leaves. The ► badge marks them in the grid.
- **Audio plays on hover** — one clip at a time; a new hover, leaving the tile, or clicking anywhere stops the previous one. (The very first play on a fresh page may need one click first — that is the browser's autoplay policy, not the node.)



Mixed grid: audio waveform, a video tile and stills side by side

The same rules follow into **🔍 Browse Folder**, the large list view for hunting through long folders: every row carries a thumbnail, name, kind tag and pixel size, the **filter box** narrows the list as you type, and hovering a row animates its video or plays its audio just like the grid.

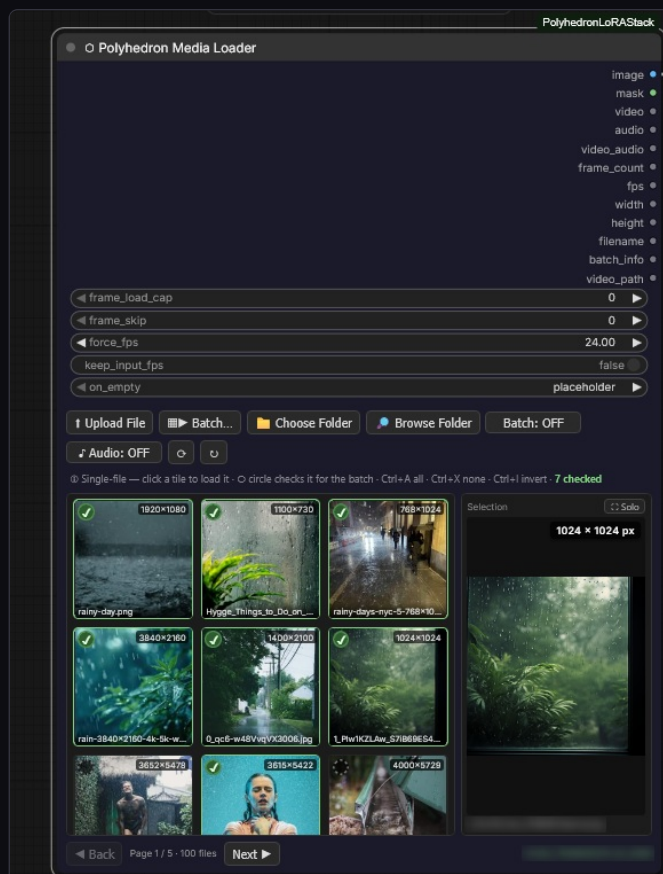


Browse Folder: filterable list with live thumbnails

Batch checkmarks

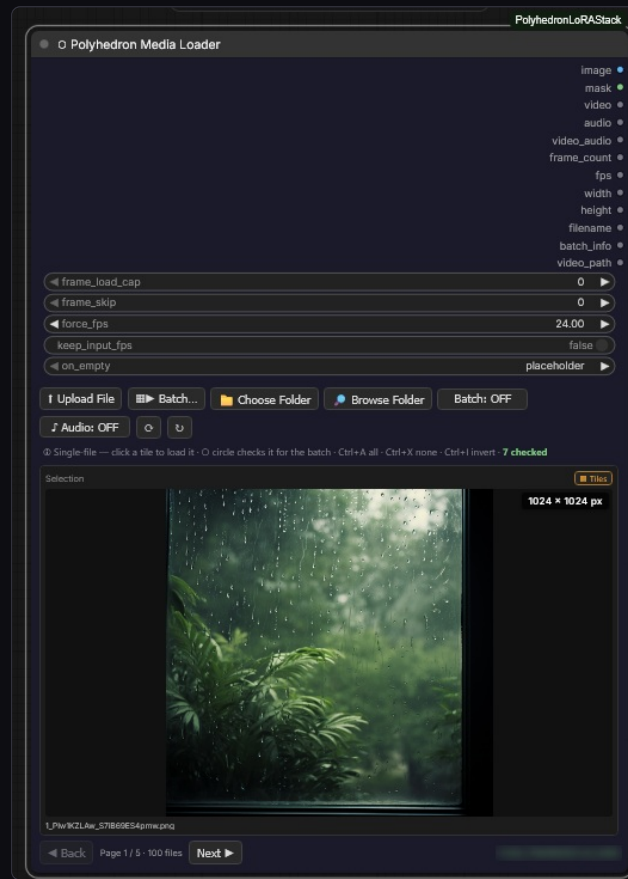
Every tile also carries a **circle checkmark**. Checking circles collects files for **batch mode** (see section 5) **without changing the currently loaded file** — the Selection panel stays where it was. The count is shown at the end of the hint line above the grid. Keyboard shortcuts work on the whole folder:

- **Ctrl+A** — check all
- **Ctrl+X** — uncheck all
- **Ctrl+I** — invert the checks



Seven tiles checked while the loaded file stays unchanged

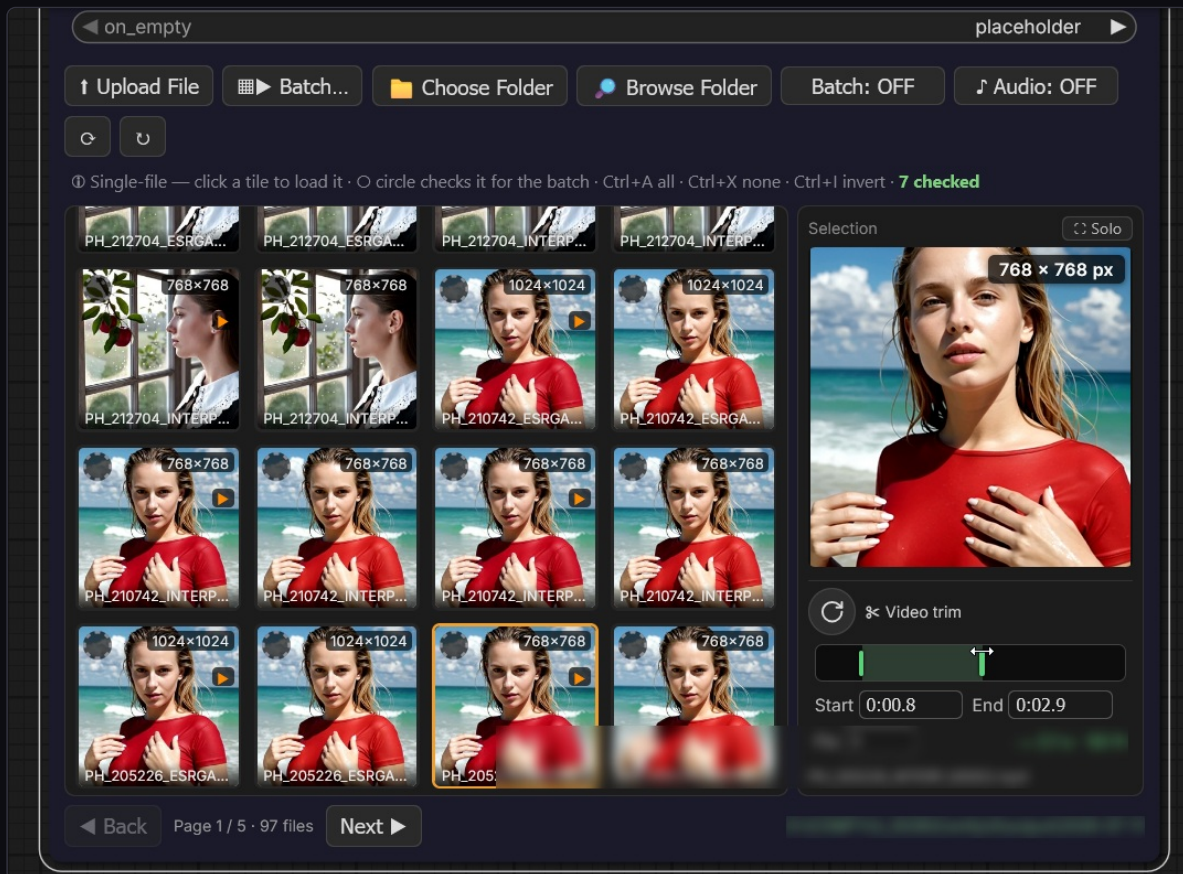
file properly before committing to it. The node size is remembered **per mode**, so you can keep Solo large and the tile view compact. The button turns into **Tiles** to bring the grid back.



Solo view: the selection fills the node, the grid is hidden

3 · Video

Pick a video tile and the Selection panel turns into a **video preview** with a trim pane underneath. The **video** output carries the trimmed clip; **frame_count**, **fps**, **width** and **height** describe what actually comes out.



Video selected: preview with the trim pane underneath

Trimming

The pane below the preview is headed ∞ **Video trim**:

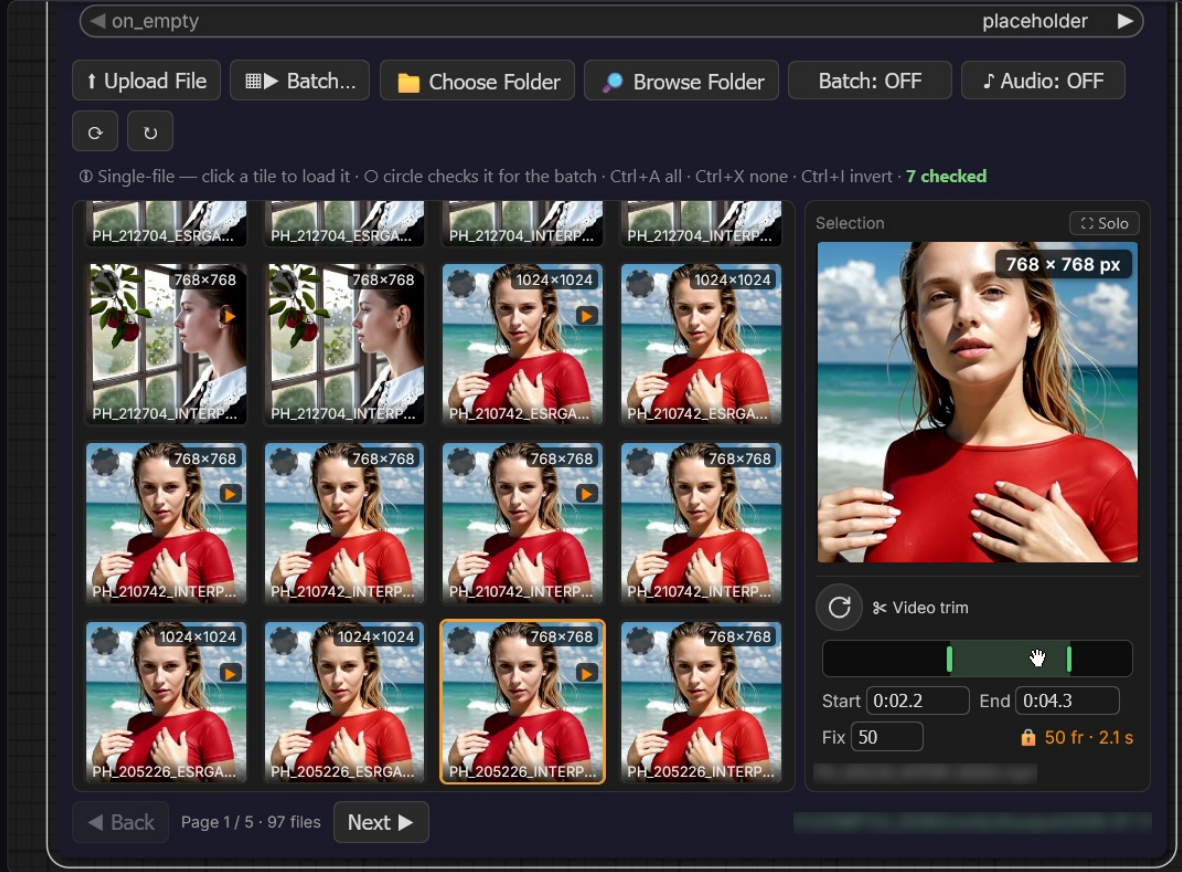
- The **track** represents the whole clip. Drag the two **handles** to set the kept range — the kept zone stays lit, the trimmed-away parts dim out.
- **Start** and **End** are editable timecode fields (**0:00.8**); typing and dragging drive each other, so you can dial an exact in-point and then nudge the out-point by hand.
- The **round button** left of the label plays the selection **on a loop**, following the handles live while you drag them.
- The **length readout** on the right shows the kept range in both currencies — seconds and frames (-> **2.1 s · 50 fr**).

Trim positions snap to real frame boundaries — the grid is 1/native fps, the very cut the backend makes, so the preview shows what the node hands on.

Fixed-length windows

Click the length readout and the window **locks**: the readout turns amber and shows a padlock. From then on **the length is the law** — dragging a handle or the kept zone **slides the window** through the clip instead of resizing it, and the window always stays fully inside the clip. The **Fix** field next to Start/End sets the locked length numerically, in frames (**fr**). Click the readout again to unlock.

This is the tool for cutting many clips to the exact same length: pin a folder, click through the videos, slide the fixed window to the good part of each.



Locked window: amber padlock readout, the window slid down the clip

Frame knobs

Four widgets on the node body decide how the video reaches the graph:

Widget	Effect
<code>frame_load_cap</code>	Video only: maximum number of frames to load. <code>0</code> = all (with a safety cap).
<code>frame_skip</code>	Video only: skip this many frames at the start.
<code>force_fps</code>	Output fps, default <code>24</code> . <code>0</code> = the file's native fps. Sets the play rate — and the frame count when a still is expanded to the length of trimmed audio.
<code>keep_input_fps</code>	On: ignore <code>force_fps</code> and use the source's native fps (a still or sequence falls back to 24). Off: <code>force_fps</code> applies.

When there is nothing to load

`on_empty` decides what happens when there is genuinely nothing to hand on — no selection, an empty or empty-filtered folder, a file that vanished from disk:

- `placeholder` (default) emits a built-in placeholder frame, so a half-built graph keeps running.
- `error` stops the run instead.

Misconfiguration and decode failures always error, in both settings.

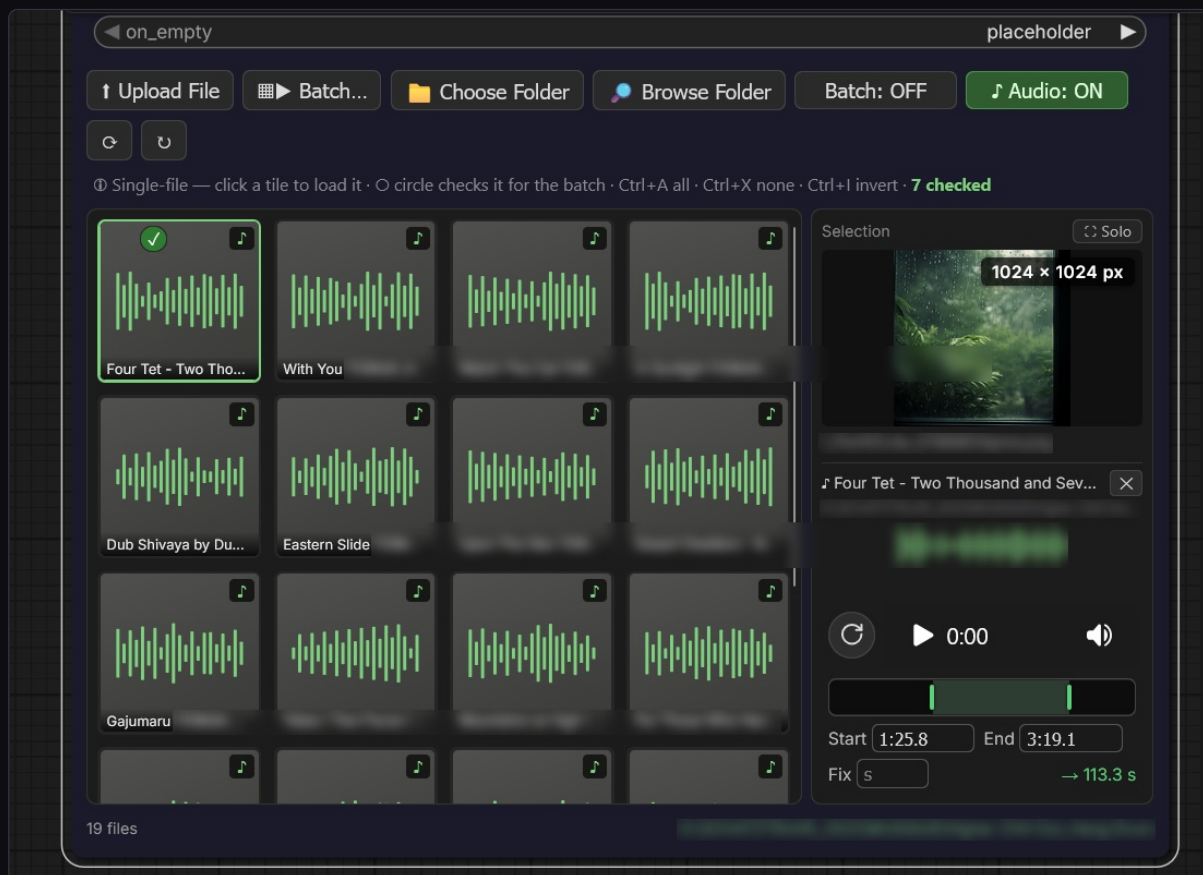
4 · Audio

Audio is a **third media kind** in the grid: audio files show up as tiles, are selectable and previewable, right next to images and videos.

Pairing

Clicking an audio tile **pairs** that file with your current visual and arms the **🎵 Audio** button. Audio is not an exclusive mode — the visual stays loaded, the audio rides along:

- **🎵 Audio: ON** reveals the audio companion pane in the Selection: the file name, the folder it came from, a **waveform**, a small transport (play button with the running time and a volume control) and the trim strip. An **×** button drops the pairing. Turning it on jumps the grid to the audio's folder, where it shows marked — the visual selection is marked in orange, the audio selection in green, so you can see both slots at once. Audio tiles carry a **waveform thumbnail** and a **🎵** badge, so a music folder reads at a glance.
- **🎵 Audio: OFF** sets the companion aside. The pairing is **remembered**, so you can hide it and bring it back without picking again.
- When the audio lives in a different folder than the one you are browsing, a **📁 button** in the pane jumps the grid over to it.
- Selecting a **video** arms its own **embedded track**, labelled **from video** in the pane. No separate file needed to keep a clip's sound.



Audio paired with a still: companion pane under the Selection

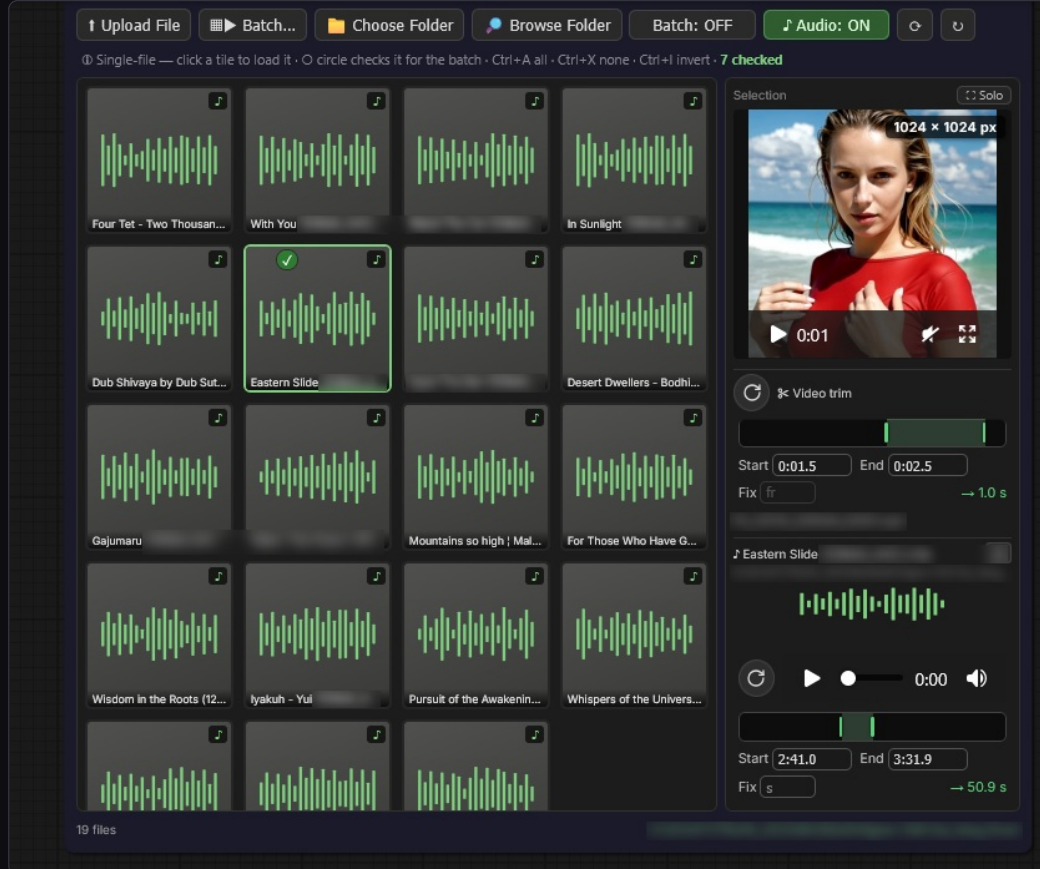
Trimming audio

The companion pane carries the same trim strip as the video pane: track, two handles, editable in/out fields, loop play and the length readout. Two differences:

- Audio snaps to **tenths of a second** instead of frame boundaries, and its **Fix** field is in seconds (**s**) where the video's is in frames (**fr**).
- The **fixed-length window** works exactly as it does for video — click the readout to lock, then slide the window through the track.

Pair audio with a **video** and both panes stack under the preview: the video trim on top, the audio companion below, each with its own handles, fields and readout. Trimming one does not disturb the other.

The video and audio trim handles follow the same window rules, so a clip and its sound stay in agreement.



Video and audio trimmed side by side, each with its own pane

Who has the upper hand

When a visual and an audio are paired, one of them sets the length. The rule is not "whichever is shorter" — it depends on what the visual is:

Pairing	Who wins	What happens
Visual only, no audio	—	Nothing is re-timed. A still stays one frame, a video keeps its frames.
Still + audio	the audio	The audio is the law: the still expands into a real clip of the trimmed audio's length. <code>force_fps</code> sets the rate, and the audio is fitted to the exact frame count so the two end together. A very long track is cut at a hard frame cap — the console says so and asks you to trim shorter.
Video + audio	the video	The video is the law: its frames are kept untouched and the audio is capped or silence-padded to the video's duration . A short track is padded with silence to the end, a long one is cut.

So a still bends to the sound, and sound bends to a real video. The picture never gets stretched or dropped to fit a track.

Which trim drives which output

The two trim panes are independent while you work, but they do not feed the outputs in the same way:

- audio** always carries **its own** trim window — the one you set in the audio pane. It stays independent no matter what the video does.
- video** carries the **video** trim. When a video is trimmed, the clip is re-encoded from the sliced frames and takes the file's **own** sound track, cut to the same window, so it stays self-contained and in sync.
- video_audio** (the muxed clip) follows the **video trim as master**: the paired audio is sliced to the *same* window the frames use and fitted to the trimmed duration. Your separate audio-trim handles do not shift the muxed clip out of sync. In every other case — still, untrimmed video, image batch — the muxed clip simply takes the trimmed paired audio.

Short version: **the video trim is the master lever for the muxed clip, and the audio trim owns the standalone audio output.**

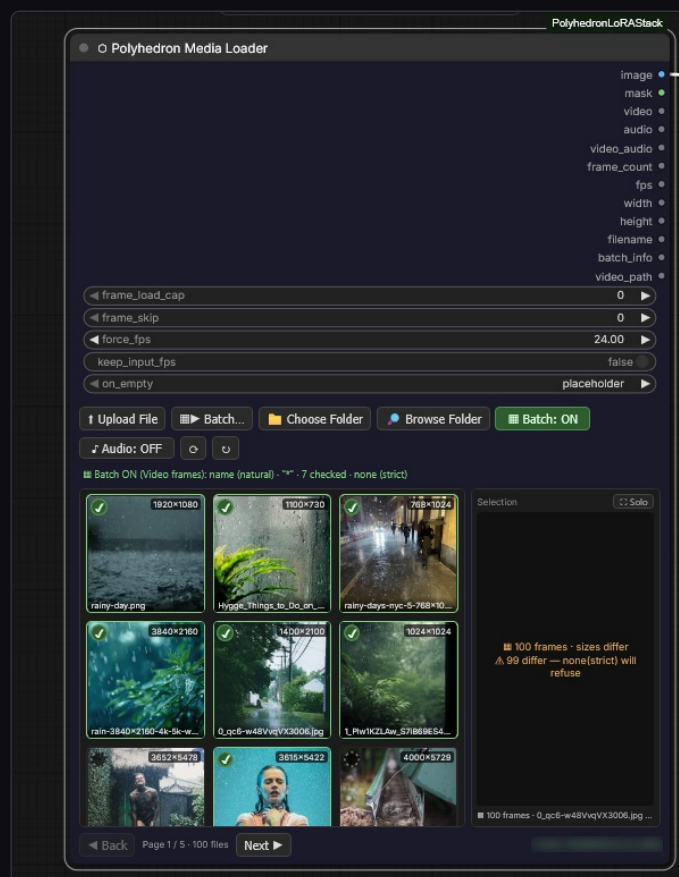
What comes out

- audio** hands on the decoded, trimmed audio. Decoding goes through PyAV — the same path ComfyUI's own audio loader uses — so m4a, aac and opus work without backend gaps.
- video_audio** hands on the visual's frames **muxed with the paired audio**: wire it into a video save node for an mp4 with a sound track. It is `None` when nothing is paired.

An **audio-only** selection still emits a small placeholder frame on the `image` output, so a graph that expects an image keeps

5 · Batch

Batch mode answers two questions in order: **which** files run, and **how** they run.



Batch armed: green Batch button, status line and batch info in the Selection

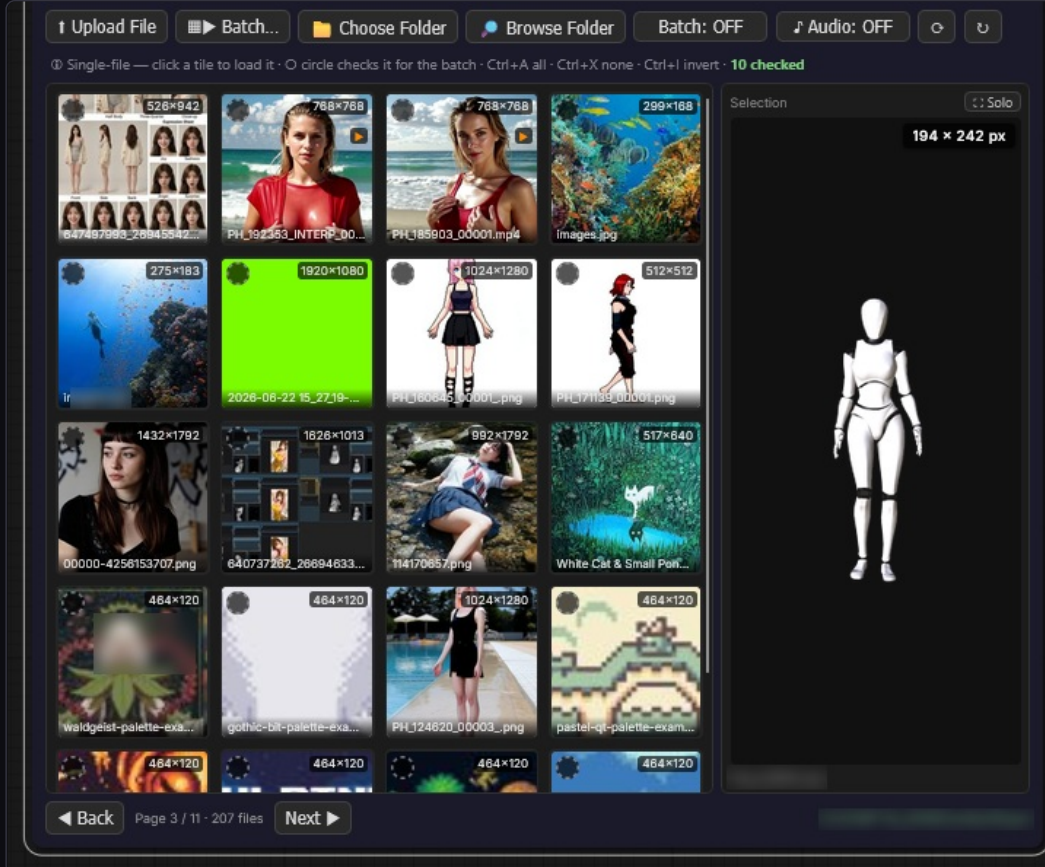
Which — the circle checkmarks from section 2. Checks are remembered in the workflow. **How** — the **Batch...** button opens the batch panel; **Apply** arms it, and the **Batch** button on the node turns it on and off afterwards without losing the checks. While it is armed, a status line above the grid spells the whole configuration out.

The batch remembers — even while you are somewhere else

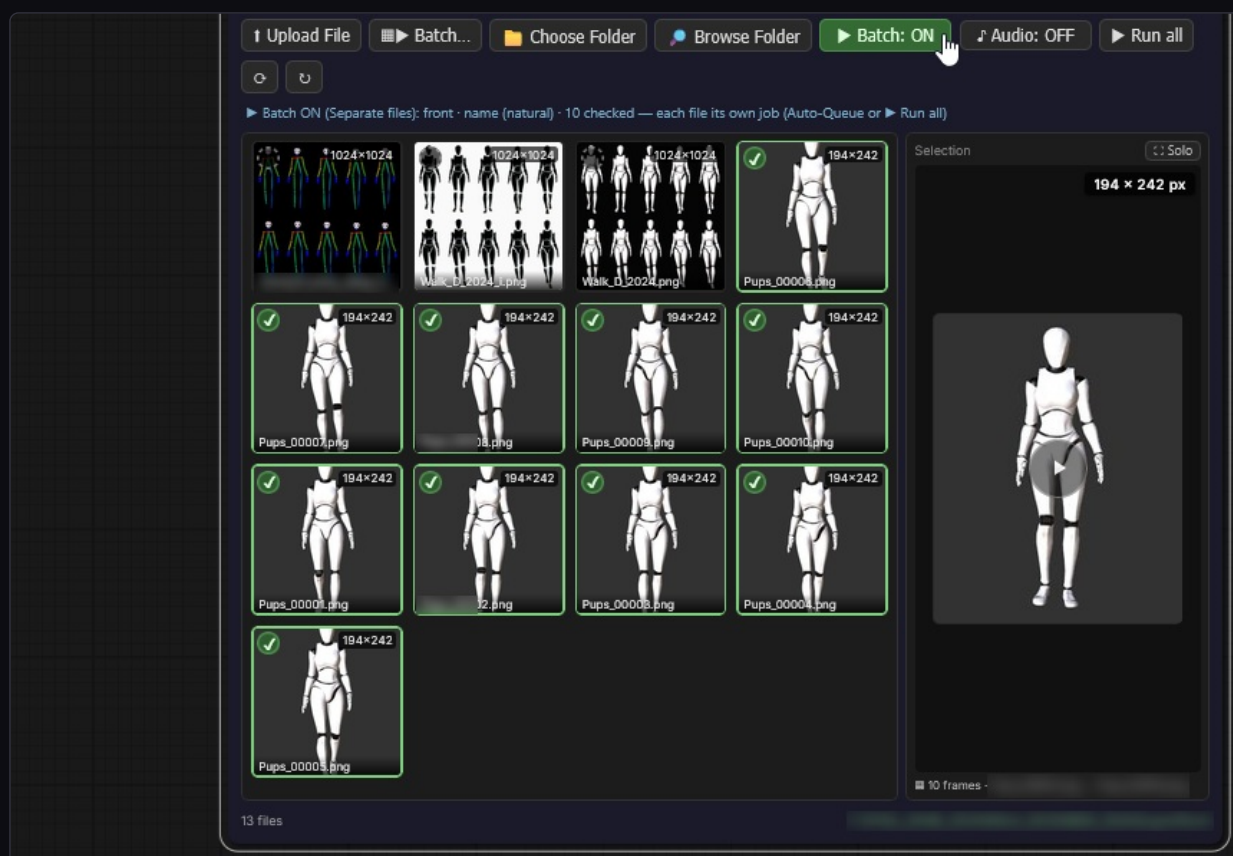
The checked set and the source folder are **not** tied to what you are currently browsing. You can turn the batch **OFF**, wander into a completely different folder, load other files — the hint line keeps showing the count (e.g. **10 checked**) the whole time, because those checks still exist and still belong to their source folder.

Click **Batch: ON** and the grid **jumps straight back to the source folder**, with the checked frames marked — you are looking at exactly what will run, no matter where you were a moment ago. Turning it off returns you to browsing without touching the checks.

So the mental model is: **the checks live with their folder, not with your current view**. The count in the hint line is a reminder that a prepared batch is waiting, and the Batch button is the way back to it.



Batch OFF in another folder: the count persists in the hint line

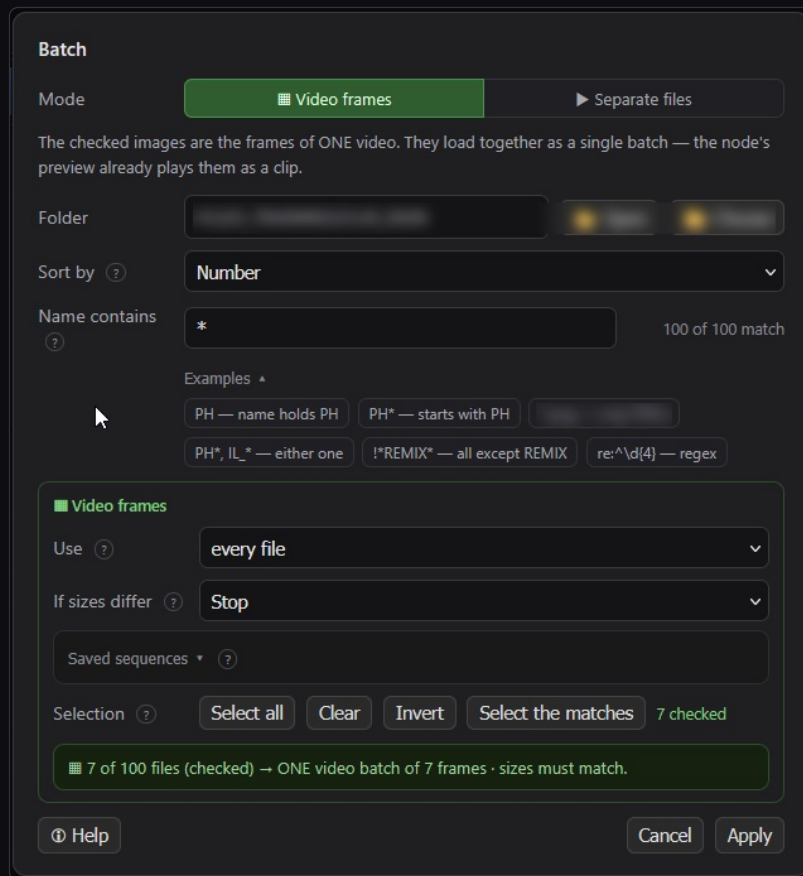


Batch ON: the grid jumped back to the source, checks marked

The two modes

Mode	What it does
Video frames	The checked images are the frames of one video and load as a single batch, after a size check.
Separate files	Each checked file — image or video — is its own job , fed in one per run. Auto-Queue or the Run all button sweeps through them.

Picking the set



The batch panel in Video-frames mode, examples row expanded

- **Sort by** — **Number** (natural order, so `img2` comes before `img10`), **A-Z** (literal character order), **Date modified** or **Date created**, oldest first.
- **Name contains** — plain text matches any name holding it, `*` means all files. The **Examples** row fills in the richer syntax: `PH*` (starts with), `*.png` (only PNGs), `PH*, IL_*` (either), `!*REMIX*` (exclude), `re:^(d{4})` (regex). **Select the matches** checks exactly the files the filter finds.

Video frames options

- **Use every Nth file** — thin the batch out: every 2nd halves it, every 3rd thirds it, or set a custom N.
- **If sizes differ** — what to do with mismatched frames: **Stop** (refuses to run and names the odd one out), **Stretch** (resize everything to the first frame's size), **Letterbox** (pad with bars, nothing squashed) or **Crop center**.
- **Saved sequences** — build the current batch into a renumbered copy under `output\PLS_sequences` and load it as the active batch. Your original files are never touched. Saved sequences can be re-loaded or deleted from the same row.

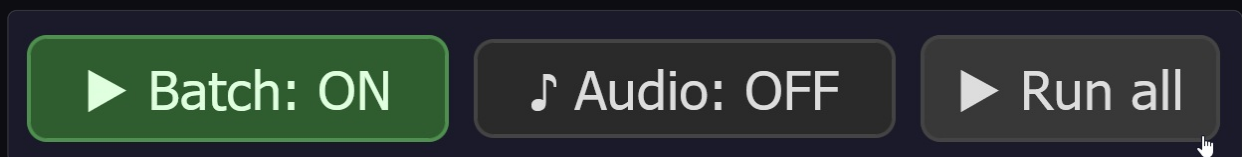
Separate files options

- **Begin at file** — which file of the set goes first; **Apply** sends the cursor back to it.
- **Start over after the last file** — on, the cursor wraps around to the first file; off, it stays parked on the last one.

Running the set — one press vs. ▶ Run all

The part that is easy to misread: in Separate-files mode **one queue press = one file**. Each normal ComfyUI queue run feeds exactly the file under the cursor into the graph and then advances the cursor — pressing queue once and seeing a single output is the mode working as designed, not a stall.

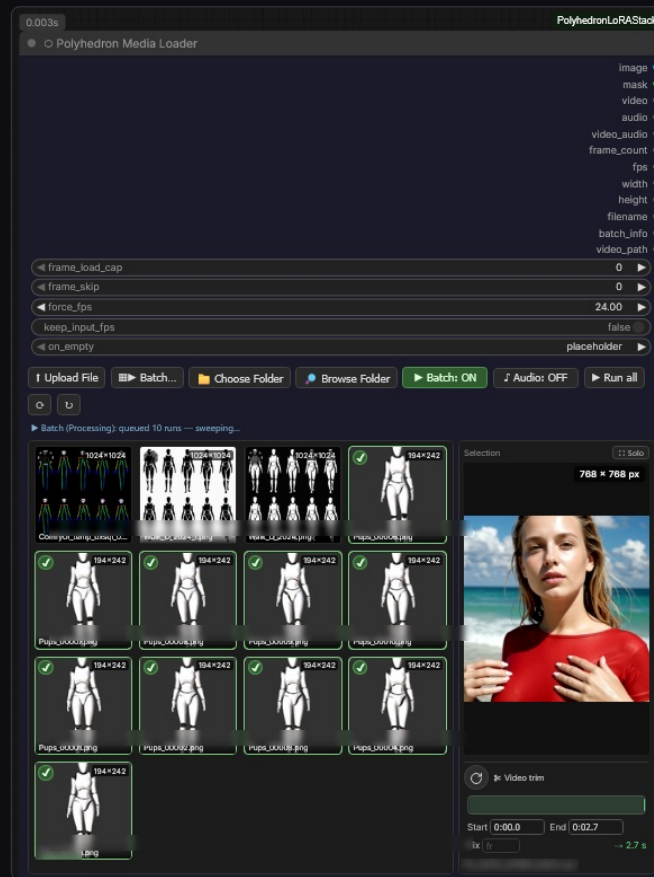
To work through the whole set in one go, use ▶ **Run all** — the button appears in the node's toolbar only while the batch is armed (Batch: ON). One press queues every remaining file as its own run.



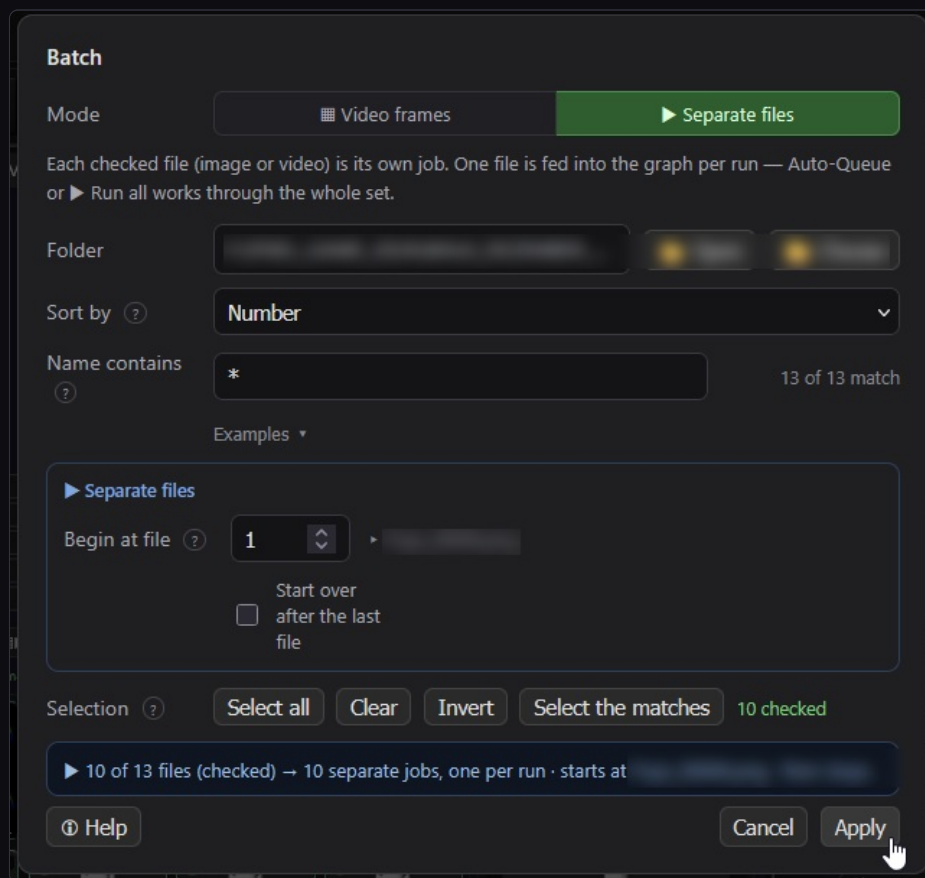
The ▶ Run all button lives on the node itself, next to the batch toggle

Note the difference between the two batch modes here: a **Video frames** batch is ONE clip and runs through ComfyUI's own Run/Queue button like any other graph execution — there is nothing extra to press on the node. The Separate-files sweep is **not** started from ComfyUI's Run button; it is started from the node's own ▶ **Run all**. While the sweep is going, the node's hint line counts along:

That line is the confirmation that the set is actually being worked through. When the sweep finishes, the cursor parks after the last file — or wraps back to the first, if *Start over* is on.



► Run all mid-sweep — the hint line counts the queued runs



The batch panel in Separate-files mode

What the graph sees

- **batch_info** is a live counter for a wired text node: `Batch 0023 / 1334` per firing in Separate-files mode, `Batch 1334 frames (one pass)` for an image batch, and `Single: <name>` outside batch mode.
- **video_path** hands on the full path of the source file — the Batch Pipeline Source node consumes it. It is empty in modes without a single source file (image batch, sequence).

6 · Outputs & masks

The full pin table lives up front in *At a glance* — *every output pin*. This section covers the part that is genuinely this pack's own: the mask conventions and the round trip.

The mask rules

- **Alpha convention** (matching core LoadImage): **opaque** → **0**, **transparent** → **1**. A still with a real alpha channel carries it as the mask.
- **A grayscale still IS a mask**: a mode-L image without alpha carries its **luminance** as the mask, **white** → **1**. That is deliberately the inverse reading — a saved mask is white-on-black, and this rule makes it survive the round trip (below).
- **Video**: a plain video gets a blank (all-zero) mask. An **alpha-bearing video** — VP9-alpha WEBM, ProRes 4444, transparent GIF — is decoded through PyAV so the alpha becomes a **real per-frame mask**; without PyAV the node says clearly what to install instead of failing obscurely.

The Save → Load round trip

The Save node closes the loop:

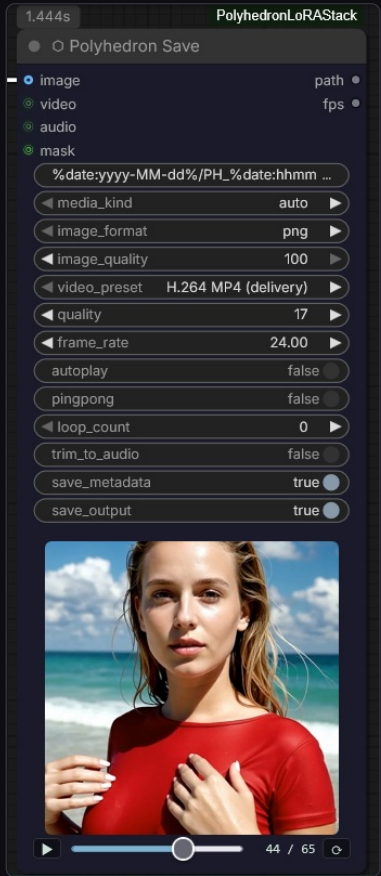
- Wire **only a mask** into Save and the mask itself is written as **grayscale stills** (mode L, white = mask on) — one file per batch entry, with the same naming, metadata and preview machinery as normal images.
- Load such a file back and the grayscale rule above turns it into the identical MASK again. **What Save writes, the Loader reads back** — masks included.

The two-loader idiom

Because a saved mask is just a file, guided background removal becomes a folder workflow: one loader carries the **images**, a second loader carries the **masks** (the grayscale stills Save wrote), and the removal node takes both. Edit a mask in any painting tool, drop it back into the mask folder, ↺ re-read — the correction flows through without any in-graph mask surgery.

7 · Polyhedron Save

One output node for stills, image sequences, videos (with muxed audio) and masks. It picks the writer from what is actually wired to it, embeds the workflow so the file can be dragged back into ComfyUI — and when it writes a clip it drops a **still frame beside it as a control image**, because an .mp4 cannot carry a workflow but a PNG can.



Polyhedron Save: inputs, widgets and the in-node player

Inputs and outputs

Pin	Direction	Carries
image	in	Still(s) or video frames [N,H,W,3\ 4] .
video	in	A native VIDEO — frames, audio and fps ride through. Takes the loader's video or video_audio output alike: a video_audio wire carries its trimmed companion track along, muxed into the written clip.
audio	in	Optional: muxed with image frames, or replaces a wired video's audio.
mask	in	Optional alpha [N,H,W] (opaque → 0) for png/webp stills and alpha-capable presets (ProRes 4444). Wired alone , the mask itself is saved as grayscale stills (section 6).
path	out	The full path of the written file.
fps	out	The rate the clip was written at.

What gets written — media_kind

auto infers from the wiring: a VIDEO input → video; IMAGE + audio (or an fps) → video; IMAGE alone → image; AUDIO alone → audio; MASK alone → grayscale mask stills. Or force image / video .

Naming files and folders — the token table

filename_prefix is a path, not just a name: every / creates a subfolder under ComfyUI/output/ , and tokens are expanded before the path is built. All the pieces:

Piece	Writes	Example
plain text	itself	Polyhedron → Polyhedron_00001_.png
/	a subfolder level	renders/test → output/renders/test_00001_.png
%date:yyyy%	4-digit year	2026
%date:yy%	2-digit year	26
%date:MM%	month (01-12)	07
%date:dd%	day (01-31)	19
%date:hh%	hour (00-23)	10
%date:mm%	minute	29
%date:ss%	second	36
%width% / %height%	frame size in pixels	1024
%Node.widget%	another node's widget value (expanded by ComfyUI core)	%KSampler.seed%

Date letters combine freely inside one token — %date:yyyy-MM-dd% gives 2026-07-19 , %date:hhmm% gives 1029 . **Uppercase MM is the month, lowercase mm the minute.** A counter (_00001_) is always appended, so files never overwrite each other. Put together, patterns like these sort themselves:

Pattern	Result
%date:yyyy-MM-dd%/PH_%date:hhmmss%	one folder per day, files stamped to the second: output/2026-07-19/PH_102936_00001_.png
runs/%width%x%height%/frame	grouped by resolution: output/runs/1024x1024/frame_00001_.png
%date:yyyy%/%date:MM%/clip	year/month tree: output/2026/07/clip_00001_.mp4

Illegal path characters are stripped per segment (a stray : can never break the path on Windows), traversal segments (..) are dropped, and an empty result falls back to Polyhedron .

Stills

- image_format — **png** (lossless, embedded workflow, drag back to reload), **webp** (smaller; image_quality 100 = lossless), **jpg** (no alpha).
- image_quality — webp/jpg only; png is always losslessly compressed.

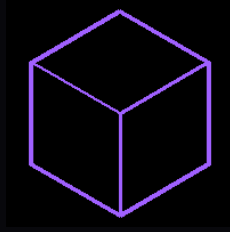
Video presets

Preset	Container	For
H.264 MP4 (delivery)	.mp4	Small, universal — ComfyUI's own tested writer.
H.265 MP4 (delivery, grain)	.mp4	10-bit HEVC, grain-tuned so particles and gradients are not smeared by HEVC's default in-loop filters.
ProRes 422 HQ (master)	.mov	10-bit edit/archive master, intra (every frame a keyframe).
ProRes 4444 (master + alpha)	.mov	The alpha-capable master — the wired mask rides along.
FFV1 (lossless archive)	.mkv	Mathematically lossless, FLAC audio.
WebM VP9	.webm	Web delivery, Opus audio.
Animated WebP / GIF	.webp / .gif	Loops; loop_count (0 = forever).

The masters encode through bundled PyAV with a 16-bit intermediate, so the diffusion float precision reaches the encoder — less banding. quality is crf-style for the lossy presets (~17-20 is visually lossless); ProRes/FFV1 ignore it.

The remaining levers

Widget	Effect
frame_rate	fps when a video is built from an IMAGE batch; a wired VIDEO keeps its own rate. Convertible to an input — wire Interpolate's fps output here.
autoplay	In-node preview: off = first frame with play controls; on = plays automatically, muted.
pingpong	Appends the reversed frames for a seamless boomerang loop.
loop_count	GIF / animated-WebP loops, 0 = forever.
trim_to_audio	Cuts the longer of clip and audio so both end together.
save_metadata	Embeds workflow + prompt (PNG text / MP4 metadata).
save_output	On: write to ComfyUI/output/ . Off: temp dir — preview only, keeps output/ clean while iterating.

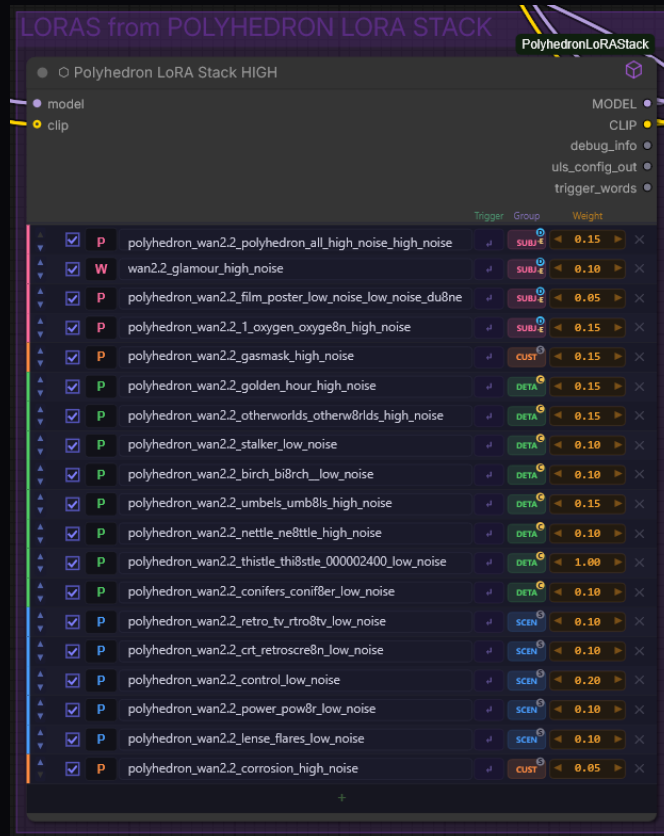


Polyhedron LoRA Stack

User Documentation · v313

by Polyhedron

A custom ComfyUI node for managing and applying large collections of LoRAs with group-based apply modes, cleanup switches, thumbnails, trigger words, an Accelerator engine and a Merge Analyzer.



Both Stack nodes side by side — HIGH and LOW noise paths in a WAN 2.2 dual-noise workflow.

© Polyhedron · All rights reserved

civitai.com/user/Polyhedron_AI · patreon.com/c/polyhedron_ai

1 · Overview

The Polyhedron LoRA Stack is a replacement for the standard Power LoRA Loader node. It is designed for workflows that use many LoRAs simultaneously — such as WAN 2.2 dual-noise image generation — where managing individual LoRA nodes becomes impractical.

The pack currently ships **thirteen nodes**. The three core nodes:

- **Polyhedron LoRA Stack** — the main node. Holds any number of LoRAs, organises them into groups, and applies them in a defined order.
- **Polyhedron LoRA Engine** — a companion node for acceleration LoRAs (LightX2V, FusionX, CausVid, ...). Flat list, single global apply mode, placed before the Stack in the pipeline.
- **Polyhedron LoRA Inspector** — a passive consistency-check node. Compares the active LoRAs from the Stack against the actual prompt text and reports which trigger words are present and which are missing.

New since this documentation was last updated:

- **Polyhedron Merge Analyzer** — a passive measurement node for the Combined / Smooth Mix merge path. Shows what the Stack is about to merge and, on request, measures how faithful the merge is (chapter 14).
- **Cleanup switches — Trim & Resolve** — two per-group refinement switches for stacks with many concurrent LoRAs (section 3.7).

The companion nodes — Token Counter, Select Model Switch, WAN Bridge, Frame Inflate / Pick Frame and the Sigma nodes — are covered in chapters 9–13.

Design principle

One node = one model. For dual-noise workflows, use two Stack nodes side by side (HIGH and LOW), each receiving its own model. The nodes are fully independent.

The backend is model-agnostic: FLUX, WAN 2.1, WAN 2.2, SDXL, SD 1.5 all work identically.

2 · Installation

Copy the folder into your ComfyUI custom nodes directory:

```
ComfyUI/custom_nodes/ComfyUI-PolyhedronLoRAStack/
```

Then install the dependencies. For standard Python environments:

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

For ComfyUI Portable (Windows):

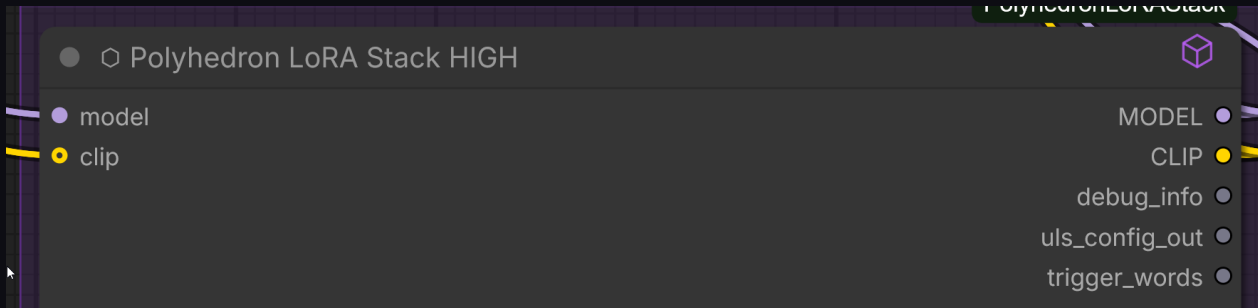
```
D:\ComfyUI\python_embeded\python.exe -m pip install -r requirements.txt
```

Dependencies: Pillow >= 9.0.0, requests >= 2.28.0

3 · Polyhedron LoRA Stack

The Stack node accepts a MODEL and optionally a CLIP input, applies all active LoRAs in group order, and outputs the patched model along with debug information and collected trigger words.

3.1 · Node header

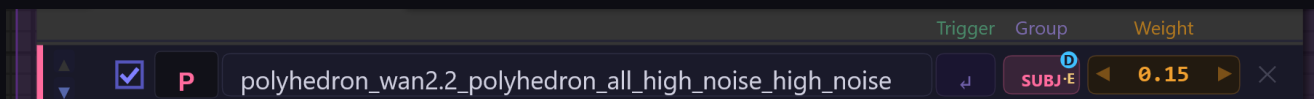


Stack node header — inputs on the left, outputs on the right.

The header contains:

- **model** (input) — the diffusion model to patch.
- **clip** (input, optional) — the text encoder. Not required for WAN/video models.
- **MODEL** (output) — the patched model, ready for the sampler.
- **CLIP** (output) — the patched text encoder.
- **debug_info** (output) — lists every applied LoRA, weight, group and mode.
- **uls_config_out** (output) — internal configuration passthrough for chaining and analysis.
- **trigger_words** (output) — all trigger words from active LoRAs, comma-separated.

3.2 · Row anatomy

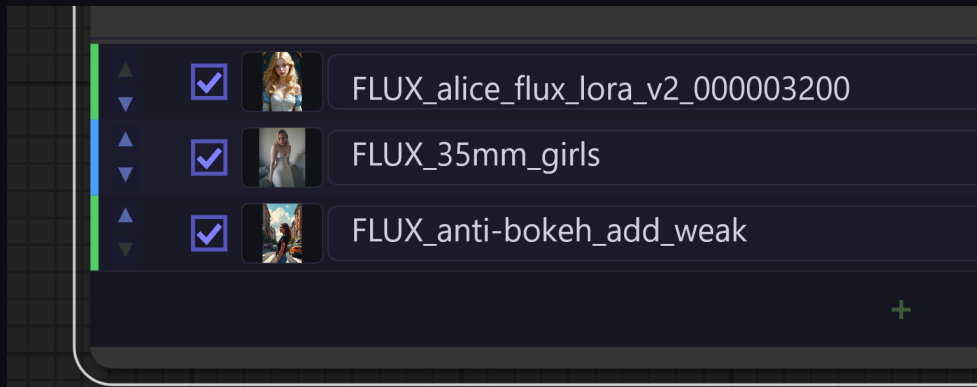


A single LoRA row — all interactive elements visible.

Each row represents one LoRA. From left to right:

- ▲ ▼ — move the row up or down in the list.
- **Checkbox** — enable or disable this LoRA without removing it.
- **Thumbnail** — preview image if available. Click to open the detail overlay.
- **Name button** — displays the filename. Click to open the LoRA selector with search.
- **Insert** — inserts trigger word(s) at the cursor in the active text node.
- **Group pill** — shows the group (SUBJ, DETA, SCEN ...). Left-click cycles groups. Right-click opens the apply mode panel.
- **Weight** — LoRA strength. Click arrows to step by 0.05, or click the number to type directly.
- **x** — remove this row.

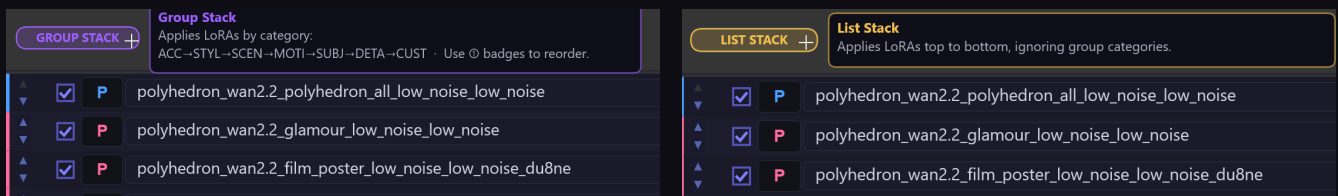
3.3 · Adding and reordering rows



▲▼ reorder buttons and the + button to add new rows.

Click + at the bottom of the list to add an empty row. Use ▲ / ▼ to reorder, or drag from the far-left handle.

3.4 · Stack mode and group order



GROUP STACK — LoRAs applied by category in a fixed order. · LIST STACK — LoRAs applied top to bottom, ignoring groups.

The **GROUP STACK** / **LIST STACK** pill in the header controls how LoRAs are ordered when applied. Click it to toggle between the two modes.

- **GROUP STACK** (purple) — LoRAs are applied in group order: ACC → STYL → SCEN → MOTI → SUBJ → DETA → CUST. This is the default and recommended mode.
- **LIST STACK** (gold) — LoRAs are applied exactly in the order they appear in the list, top to bottom. Groups are ignored. Useful for quick testing or when list order matters more than categories.

Custom group order

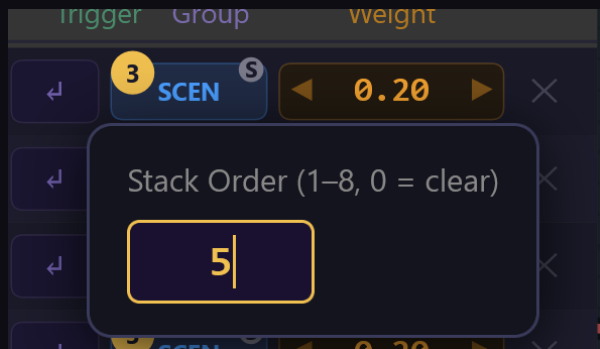


Gold badge = custom order assigned. · Dashed circle = no order set (runs after numbered groups).

In **GROUP STACK** mode, each group can be assigned a custom stack position using the **gold order badge** in the top-left corner of the Group pill.

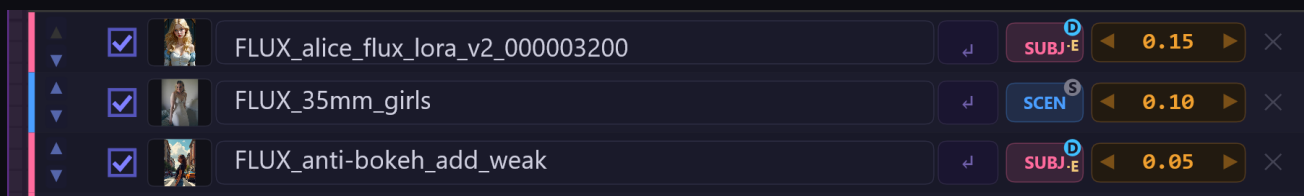
- Gold filled badge with a number — this group runs at that position.

- Dashed circle — no position set, runs after all numbered groups.
- **Click the badge** to open the Stack Order input. Enter 1–8 to assign a position, 0 to clear.
- Each number can only be used once — a duplicate entry is rejected and the conflicting badge flashes red briefly.



Stack Order input — integer 1–8, enter 0 to clear.

3.5 · Groups



Group pills: SUBJ·E = Element Drop, SCEN·S = Sequential, DETA·C = Channel Drop.

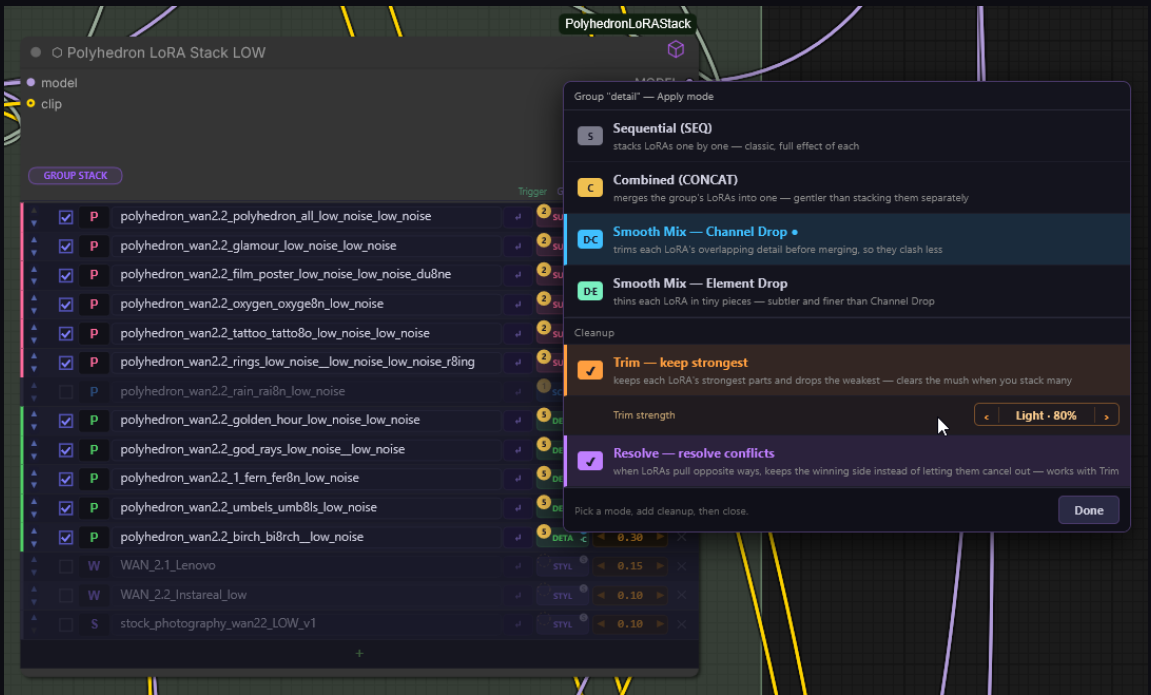
Every LoRA belongs to a group. Groups control the *application order* and the *apply mode* for that batch. Default group order:

ACC → STYL → SCEN → MOTI → SUBJ → DETA → CUST → (ungrouped)

Group	Badge	Intended content	Default mode
—	—	Ungrouped / unassigned LoRAs	SEQ
acc	ACC	Acceleration / distillation LoRAs (LightX2V, FusionX, CausVid ...)	SEQ
style	STYL	Film look, colour grading, artist styles	SEQ
scene	SCEN	Setting, atmosphere, vegetation, objects	SEQ
motion	MOTI	WAN motion LoRAs	SEQ
subject	SUBJ	Main subject, clothing, accessories	SEQ
detail	DETA	Skin, eyes, hands, anatomy corrections	SEQ
custom	CUST	Anything else	SEQ

Note: Group assignments persist in `uls_groups.json` next to your LoRAs folder — they survive workflow reloads and ComfyUI restarts.

3.6 · Apply modes



Right-click on any Group pill to open the apply-mode panel — base modes on top, Cleanup switches below.

Each group has its own apply mode. **Right-click the Group pill** to open the apply-mode panel. The mode applies to all rows in that group.

The panel **stays open** while you click, so you can pick a base mode and layer the Cleanup switches (section 3.7) in one go. Close it with **Done**, a click outside, or **Escape**.

Mode	Badge	What it does	Best for
Sequential	S (grey)	Applies each LoRA one by one — classic, full effect of each.	Default. Always safe.
Combined	C (gold)	Merges the group's LoRAs into one patch — gentler than stacking them separately.	Experimental comparison with SEQ.
Smooth Mix — Channel Drop	D·C (cyan/ green)	Like Combined, but trims each LoRA's overlapping detail before merging — drops whole rank-channels, LoRA-aware.	Groups with many LoRAs targeting the same area.
Smooth Mix — Element Drop	D·E (cyan/ gold)	Like Combined, but thins each LoRA in tiny pieces — drops individual tensor elements (classic DARE paper).	Alternative to Channel Drop — finer-grained.

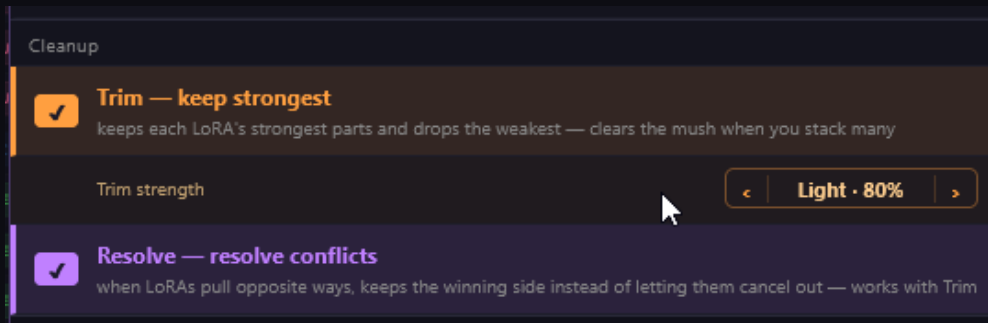
DARE density — automatic scaling

$$\text{density} = \max(0.5, 1 - 0.05 \times (n-1)) \quad | \quad 2 \text{ LoRAs: } 0.95 \quad | \quad 5 \text{ LoRAs: } 0.80 \quad | \quad 10 \text{ LoRAs: } 0.55$$

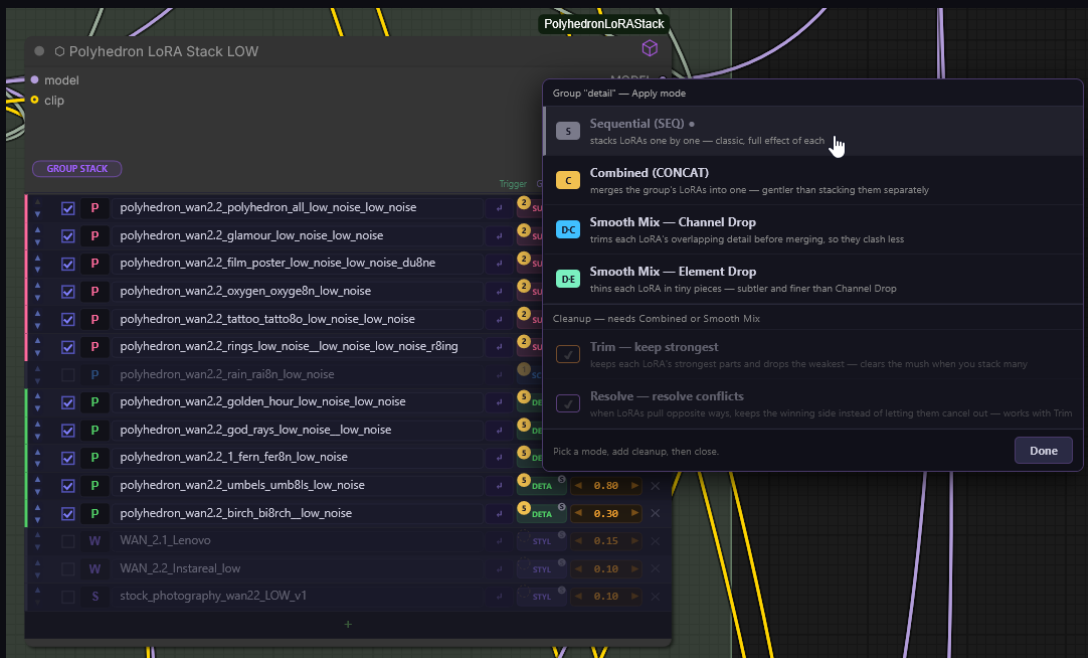
The DARE seed is deterministic (SHA-1 of names + weights) — same config = same result every session.

3.7 · Cleanup switches — Trim & Resolve

Below the four base modes, the panel has a **Cleanup** section with two optional switches. They refine the **Combined** / **Smooth Mix** merge for groups with many concurrent LoRAs. In **Sequential** mode the section is greyed out ("Cleanup — needs Combined or Smooth Mix") and the backend ignores both switches.



The Cleanup section — Trim with its strength stepper, and Resolve.



On a Sequential group the Cleanup section is greyed out — switch states are kept, but ignored.

Trim — keep strongest

Keeps each LoRA's strongest parts and drops the weakest — clears the multi-LoRA interference that builds up when many LoRAs are stacked at once. The trim is **deterministic** (no randomness): for each LoRA, the weakest rank-channels are removed by magnitude *before* the group is merged.

Trim strength — a stepper under the Trim switch (visible while Trim is on) sets how much of each LoRA is kept:

Step	Kept share	Character
Auto	group-scaled	recommended starting point
Gentle	90%	barely noticeable
Light	80%	safe everyday value
Medium	70%	noticeable cleanup
Strong	60%	aggressive
Max	50%	extreme stacks only — see warning

◀ steps toward Auto (gentler), ▶ toward Max (stronger).

Auto scaling

The more LoRAs are active in the group, the more of the weak tail is pruned: 2 LoRAs keep ~95%, 5 keep ~88%, 10 keep ~73%, 15+ hit the floor of 60%.

⚠ Too much Trim

Max · 50% can cut away the very channels that make a concept distinctive: the subject collapses to a silhouette and the base model fills the gap with unprompted content. A high LoRA weight does **not** protect against this — the trim ranking runs inside each LoRA. Use Max only for extreme stacks (15+ LoRAs) and always check results visually. If renders look washed out with Trim on, lower the Trim strength first. The recommended working range is **70-80%** (Medium/Light); the Merge Analyzer (chapter 14) warns about this in its report.

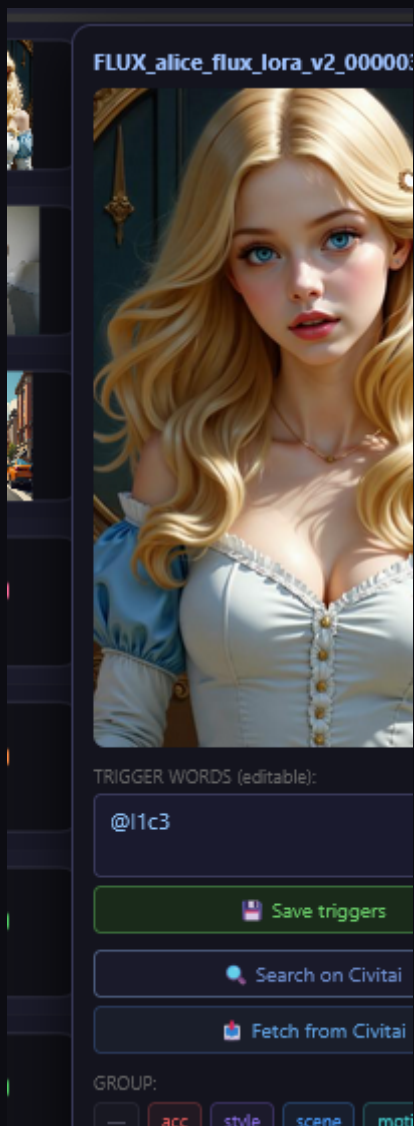
Resolve — resolve conflicts

When LoRAs pull in opposite directions, Resolve keeps the winning side instead of letting them cancel each other out — and it works together with Trim. Under the hood it runs a sign election across the group's competing LoRAs and then re-packs the result so it stays as light as a normal LoRA patch.

Good to know

- **Persistent** — the switch states *and* the Trim strength are saved into the workflow and restored on load. They survive ComfyUI restarts and never reset to defaults on their own. (Group *assignments* persist separately in `uls_groups.json`, see section 3.5.)
- **Visible in debug_info** — each group is tagged with its full mode, e.g. `DARE [CHAN] +TRIM +RESOLVE`.
- **Console feedback** — long Combined / Smooth Mix merges print a live progress log to the console (`[PLS]` lines, plus a per-phase merge-timing summary at the end), so you can see what a long merge is doing.
- **Cancel works** — ComfyUI's red × aborts a running merge promptly instead of letting it finish in the background.

3.9 · Thumbnail overlay



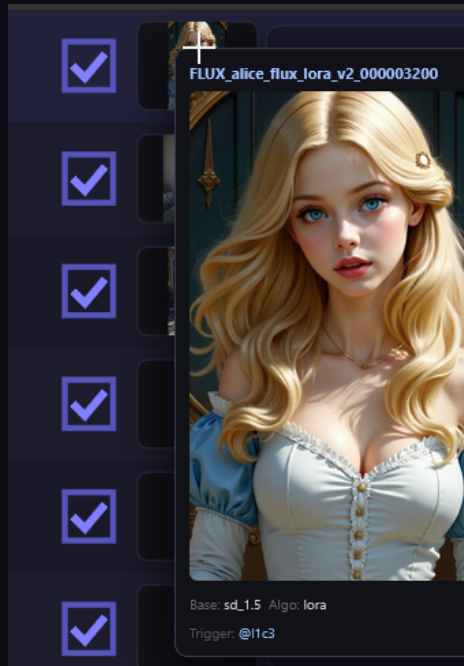
Thumbnail overlay — preview, trigger words, Civitai, group selector, Smooth Mix variant.

Click a thumbnail to open the detail overlay:

- **Preview image** — LoRA preview if available.
- **TRIGGER WORDS** — editable. Click Save triggers to store.
- **Search on Civitai / Fetch from Civitai** — Civitai integration.
- **GROUP** — reassign the LoRA to a different group.
- **DARE VARIANT** — only visible when the group uses Smooth Mix. Select CHAN or ELEM.

Note: Trigger .txt files are read-only. Edits are stored in .uls-meta.json.

3.10 · Hover preview



Hovering over a thumbnail shows a quick preview popup.

Hovering over a thumbnail shows a popup with preview image, base model type, and trigger words — no click needed.

3.11 · Trigger word insertion

The ↵ button inserts trigger word(s) at the cursor in the active text node:

- **Single trigger** — inserted immediately with a toast confirmation.
- **Multiple triggers** — a selection popup lists all options. Click one to insert.



Trigger selection popup. · Confirmation toast + trigger in the prompt node.

3.12 · Setting weights

Three ways to change a LoRA's weight:

- ◀/▶ — step by 0.05.
- **Click the number** — type any value, press Enter. Range: -10.0 to 10.0.



Direct weight input field.

3.13 · Per-row CLIP strength

By default a row's CLIP (text-encoder) strength follows its model weight — exactly as in every previous version. Each row can decouple the two and carry a separate CLIP strength, on both the LoRA Stack and the LoRA Engine.



Shift-click on a weight opens the blue CLIP Strength input. The first row is decoupled (0.30 / c 0.35).

- **Shift-click** the weight value and enter a separate CLIP strength.
- **Two-line cell** — decoupled rows show the amber model weight on top and the blue CLIP strength (c 0.35) below.
- **Shift + ◀▶** steps the CLIP strength; a plain click / ◀▶ still edits the model weight.
- **Re-link** — entering the model weight again returns the row to the classic single value.

Weight / CLIP Strength

Works on both the LoRA Stack and the LoRA Engine. Rows without a decoupled value behave exactly as before.

- ① **Shift+Click the weight value**
to enter a separate CLIP (text-encoder) strength.
- ② **Decoupled rows show two lines**
amber model weight on top, blue c 0.40 below.
- ③ **Shift + ◀▶ steps the CLIP value**
plain click / ◀▶ still edit the model weight.
- ④ **Re-enter the model weight to re-link**
the row falls back to the classic single value.

The interaction at a glance — header, decoupled cell, stepping and re-linking.

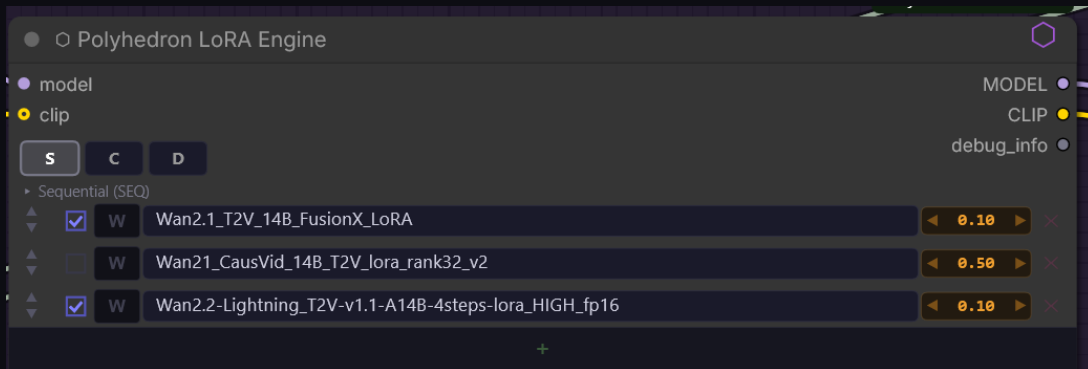
Behaviour and compatibility

Hover the Weight / CLIP Strength column header for an in-node reminder of these shortcuts. In Sequential mode the CLIP strength is passed straight to the loader; in Combined / Smooth Mix the text-encoder layers are scaled with the CLIP strength inside the same one-pass merge. Smooth Mix seeds stay derived from the model weights only, so existing results are bit-identical until a row is actually decoupled. The feature is only relevant for models whose LoRAs carry text-encoder keys (SDXL, SD 1.5, FLUX, ...) — WAN LoRAs have none, so WAN workflows are unaffected.

4 · Polyhedron LoRA Engine

The Engine node is designed for **acceleration LoRAs** — LoRAs that modify *how* the model generates (inference trajectory, step reduction) rather than *what* it depicts. Examples: LightX2V, FusionX, CausVid, WAN 2.1 Lightning.

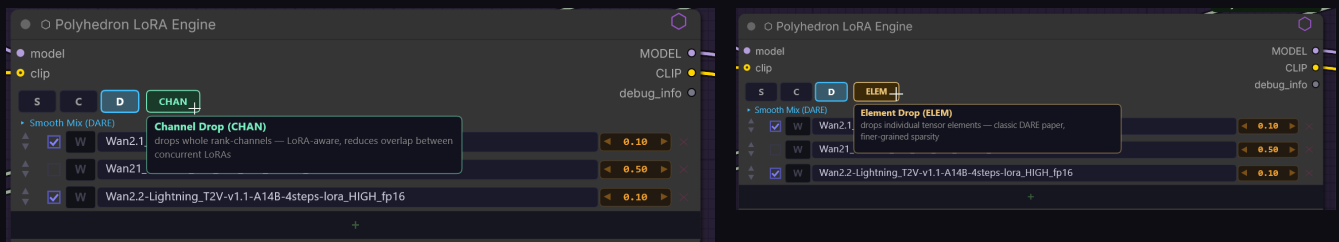
It uses a flat list with a single global apply mode. Place it *before* the Stack node — connect MODEL and CLIP outputs of the Engine into the Stack inputs.



Engine node — S mode active, three acceleration LoRAs loaded.

4.1 · Mode buttons and DARE variant

Buttons **S**, **C**, **D** select the apply mode for all Engine LoRAs. When **D** is active, a **CHAN** or **ELEM** pill appears to its right. Click to toggle. Hover for a tooltip.



CHAN — Channel Drop tooltip. · ELEM — Element Drop tooltip.

The Cleanup switches (section 3.7) are currently **Stack-only**; the Engine offers the plain S / C / D modes.

Engine vs Stack — when to use Smooth Mix

For acceleration LoRAs (typically 2-3), SEQ is the recommended default.

Smooth Mix is most useful in the Stack when a group has 5+ LoRAs targeting the same model areas.

5 · Polyhedron LoRA Inspector

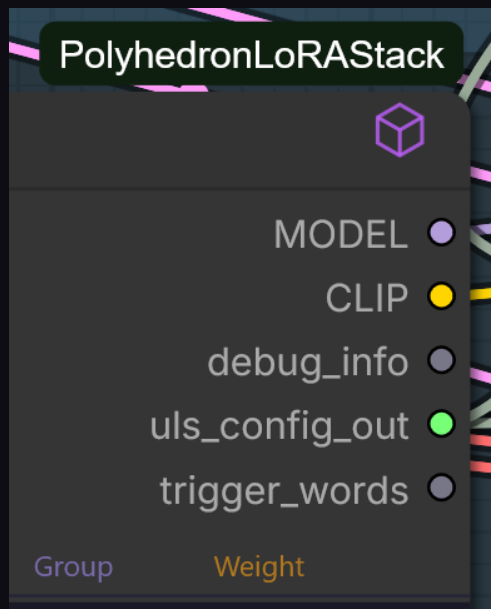
The Inspector node is a **passive consistency-check tool**. It compares the active LoRAs from the Stack node against the actual text in your prompt and reports which trigger words are present and which are missing — without modifying anything.

This is useful when working with many LoRAs at small weights: trigger words are easy to forget, and a missing trigger silently weakens the LoRA's effect. The Inspector makes this visible at a glance.

5.1 · Inputs and output

The Inspector has two inputs and one output:

- **uls_config_out** (input, green link) — connect from the Stack's *uls_config_out* output. Carries the list of active LoRAs with their weights, groups, and trigger words.
- **prompt** (input, STRING) — connect the same string that goes into your CLIPTextEncode / prompt encoder.
- **inspector_report** (output, STRING) — formatted report. Connect to a Show Text node to display it.



*Stack node — green *uls_config_out* output pin connects to the Inspector.*

5.2 · How matching works

The Inspector checks each active LoRA's trigger words against the prompt:

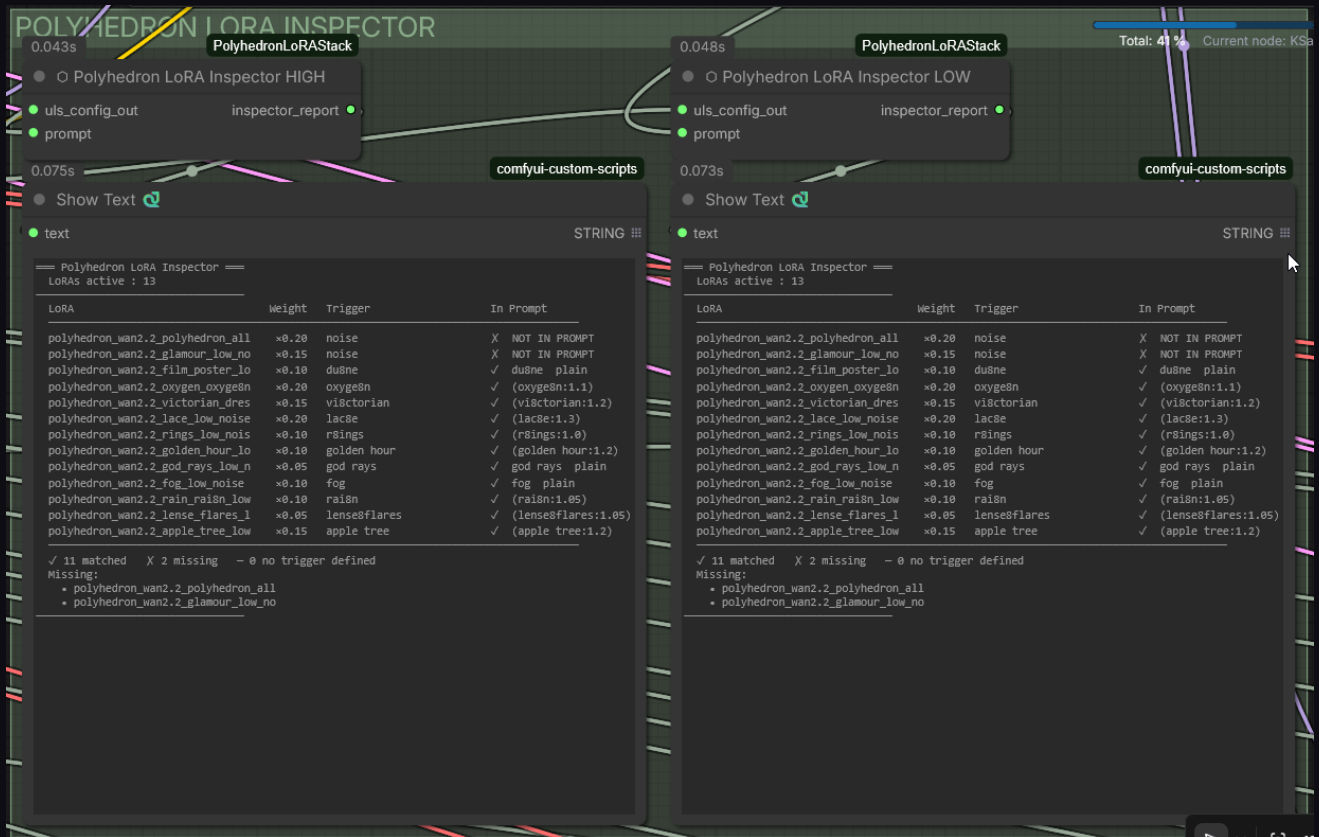
- **Weighted syntax** — (trigger:1.2), (trigger:0.8) etc. is recognised and shown verbatim in the report.
- **Plain keywords** — bare trigger words are matched with word boundaries, so "ring" inside "stinging" is not a false positive.
- **No trigger defined** — LoRAs without trigger words are listed separately and don't count as missing.

Trigger word resolution priority

1. .uls-meta.json (user-edited triggers from the thumbnail overlay)
2. .txt file next to the .safetensors (read-only)
3. Embedded safetensors header metadata
4. Filename fallback

The Inspector uses the same priority as the Stack — what you see in the overlay is what the Inspector matches against.

5.3 • Report format



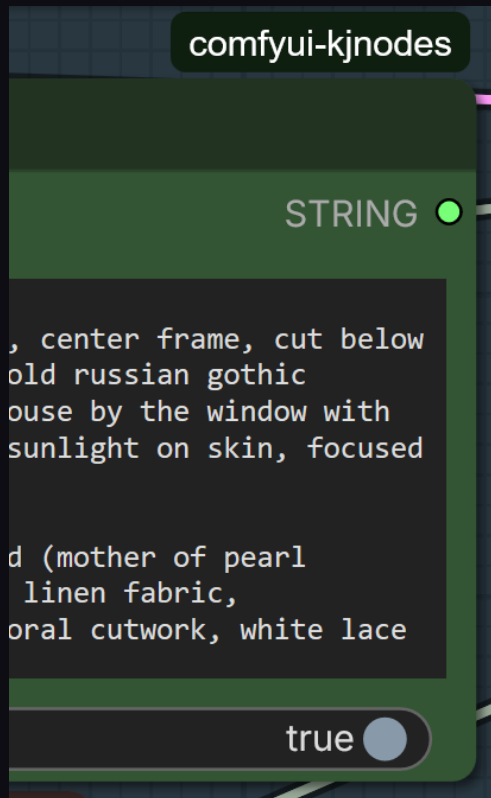
Two Inspector nodes for HIGH and LOW Stacks — same prompt input, different uls_config_out links.

The report is a fixed-width text table:

- **LoRA** — short filename
- **Weight** — current weight in the Stack
- **Trigger** — the trigger word(s) defined for this LoRA
- **In Prompt** — ✓ if found (with the actual matched syntax shown), X if missing, or 'plain' if found as a plain keyword without weight syntax

A summary line at the bottom shows totals: matched, missing, no trigger defined. Missing LoRAs are listed by name so they're easy to fix.

5.4 · Dual-noise pattern



The prompt String node — feeds both Inspector nodes and the CLIPTextEncode.

For dual-noise workflows (WAN 2.2): use **two Inspector nodes**, one connected to the HIGH Stack's *uls_config_out* and one to the LOW Stack. Both receive the same prompt. This way you can verify that all triggers for both noise paths are present in the prompt.

Inspector is passive

The Inspector never modifies your prompt or your LoRA configuration. It only reads and reports. Disconnect it any time without consequences.

The Show Text node connected to *inspector_report* only runs when needed — the Inspector itself doesn't trigger workflow execution.

6 · Connecting the outputs

Stack node outputs and how to connect them:

- **MODEL** → sampler (KSampler, KSamplerAdvanced, etc.)
- **CLIP** → CLIPTextEncode nodes
- **debug_info** → any text display node (optional but useful during setup)
- **uls_config_out** → Polyhedron LoRA Inspector (chapter 5) and/or Merge Analyzer (chapter 14); also used when chaining Stack nodes. Otherwise leave unconnected.
- **trigger_words** → JoinStrings or directly into your positive prompt node

Trigger words output

The trigger_words output collects trigger words from all active LoRAs automatically. Connect it to JoinStrings together with your manual prompt. No manual trigger word management needed as LoRAs are added or removed.

7 · Tips and recommended settings

7.1 · Weight strategy

The Polyhedron workflow uses many LoRAs at small weights (0.05–0.20) rather than few LoRAs at full strength:

- Averages across multiple training domains — reduces LoRA burn
- Allows fine-grained control over each element's contribution
- Makes Smooth Mix subtle at low weights — acts as insurance rather than a strong effect

7.2 · Dual-noise workflow (WAN 2.2)

Use two Stack nodes side by side — one for HIGH noise, one for LOW noise. Each receives its own separate WAN 2.2 model. Configure both identically.

7.3 · Group mode recommendations

Group	LoRAs	Recommended mode	Reason
subject (SUBJ)	3–5	SEQ	Full individual effect needed
detail (DETA)	5–8	Smooth Mix · CHAN + Trim + Resolve	Many overlapping LoRAs — classic conflict case
scene (SCEN)	3–5	SEQ	Small enough that conflicts are minimal
acc (ACC)	2–3	SEQ	Acceleration LoRAs rarely conflict

For Trim, start with **Auto** and stay in the **70–80%** corridor when overriding — see the warning in section 3.7 before going lower.

8 · Preview images and trigger words

Place files alongside your .safetensors files:

- **Preview image:** same base name, .jpg extension
- **Trigger words:** same base name, .txt extension. Read-only — never modified.
- Edits made in the overlay are stored in .uls-meta.json (auto-created). These take priority over the .txt file if both exist.
- Trigger words also read from the safetensors header as fallback.

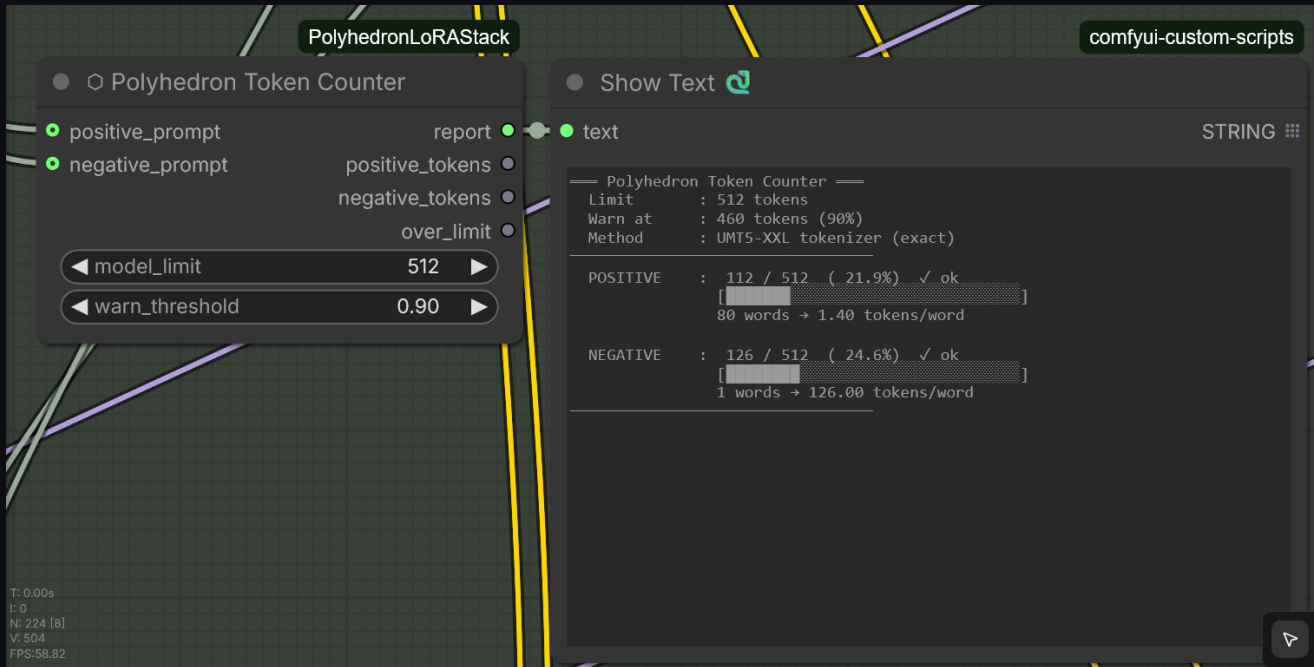
Resolution priority (highest first): `.uls-meta.json` → `.txt` → `safetensors header` → `filename`

Note: Civitai preview images and trigger words can be fetched automatically via "Fetch from Civitai" in the thumbnail overlay.

Note: Civitai preview selection is SFW-strict — only gallery images rated None/Soft are used as previews. If a LoRA's Civitai gallery contains exclusively Mature/X-rated images, no preview is stored.

9 · Polyhedron Token Counter

The Token Counter is a companion node for the Stack. It reads both the positive and negative prompt strings and counts how many tokens each consumes — measured against the limit of the active text encoder. When working with large LoRA collections, trigger words from 15–20 active LoRAs can silently push the prompt toward the encoder limit, degrading output quality without any warning in ComfyUI itself.



Token Counter node connected to a Show Text node. The report shows POSITIVE and NEGATIVE prompts separately with usage bars, percentages, and tokens-per-word ratios.

9.1 · Inputs, outputs and widgets

Name	Type	Description
positive_prompt	STRING (input)	Connect from your positive prompt node or from trigger_words.
negative_prompt	STRING (input)	Connect from your negative prompt node.
report	STRING (output)	Full formatted report — connect to a Show Text node.
positive_tokens	INT (output)	Raw token count for the positive prompt.
negative_tokens	INT (output)	Raw token count for the negative prompt.
over_limit	BOOLEAN (output)	True when either prompt exceeds warn_threshold × model_limit.
model_limit	INT (widget)	Token limit of the active encoder. Default 512 (UMT5 / WAN).
warn_threshold	FLOAT (widget)	Fraction of model_limit at which to warn. Default 0.90 = warn at 90%.

9.2 · Report output

The report shows POSITIVE and NEGATIVE separately, each with a usage bar, percentage, word count, and a tokens-per-word ratio. A checkmark (ok) appears when under the warn threshold; an exclamation mark appears when the threshold is exceeded.

Typical placement

Connect `positive_prompt` from the same string node that feeds your `CLIPTextEncode`. Connect `negative_prompt` from the negative string. The node runs passively — it never modifies data, only counts.

9.3 · Token limits by model

Different text encoders have different token limits. The table below covers all models commonly used in ComfyUI workflows. The WAN row (highlighted) is the primary target of this plugin.

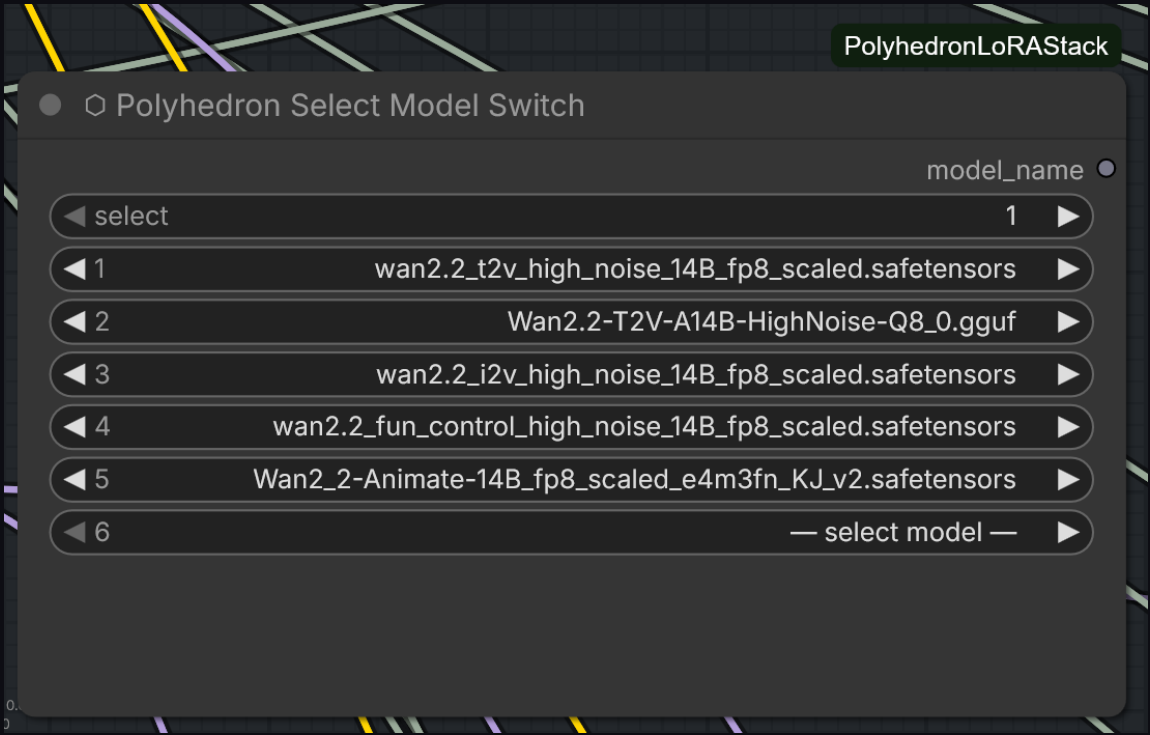
Model	Text Encoder(s)	Hard Limit	Practical	Notes
SD 1.5	CLIP-L	77	75	One chunk = 75 usable tokens. ComfyUI can chain chunks.
SDXL	CLIP-L + CLIP-G	77 each	75 each	Two encoders in parallel. Long-prompt extensions available.
SD 3 / 3.5	CLIP-L + CLIP-G + T5-XXL	77/77/512	75/75/512	Triple encoder. T5 carries most semantic load.
FLUX.1 dev / schnell	CLIP-L + T5-XXL	77 / 512	75 / 512	CLIP-L: style anchor. T5-XXL: full prompt detail.
FLUX.1 Kontext	CLIP-L + T5-XXL	77 / 512	75 / 512	Same encoders as FLUX.1. Image tokens add to budget.
FLUX.2 / Lumina	CLIP-L + Mistral-3 24B	77 / ~4096	75 / ~2000	Mistral-3 replaces T5. Quality gain plateaus early.
Qwen-Image	Qwen2.5-VL 7B	No hard limit	~500	LLM-based encoder. Practically limited by VRAM and quality.
WAN 2.1 / 2.2	UMT5-XXL	512	~256-300	Hard limit 512. Quality degrades above ~320-350: slower motion, grid artefacts.
HunyuanVideo / 1.5	LLaMA-3 + CLIP-L	1000 / 77	~400 / 75	LLaMA-3 is primary (1000 limit, ~400 recommended). CLIP-L for visual anchor.
CogVideoX	T5-XXL	226	200	T5 capped at 226 in official implementation.
Mochi / LTX-Video	T5-XXL	256	200	T5-XXL at 256 tokens.

WAN / UMT5 practical limit

Although UMT5 supports up to 512 tokens, quality degrades noticeably above approximately 320–350 tokens: motion slows down and a grid pattern artefact can appear in the output video. The default `warn_threshold` of 0.90 (= 460 tokens) is conservative — consider lowering it to 0.60–0.65 (~300–330 tokens) for WAN workflows.

10 · Polyhedron Select Model Switch

The Select Model Switch node provides six model slots and outputs the filename of the active model as a string. It is designed to sit next to your model loader nodes, allowing fast switching between model variants without rewiring the graph.



The Polyhedron Select Model Switch node with all six slots populated. Slots 1 and 2 are wire inputs (green dots on the left); slots 3-6 are dropdown widgets built directly into the node.

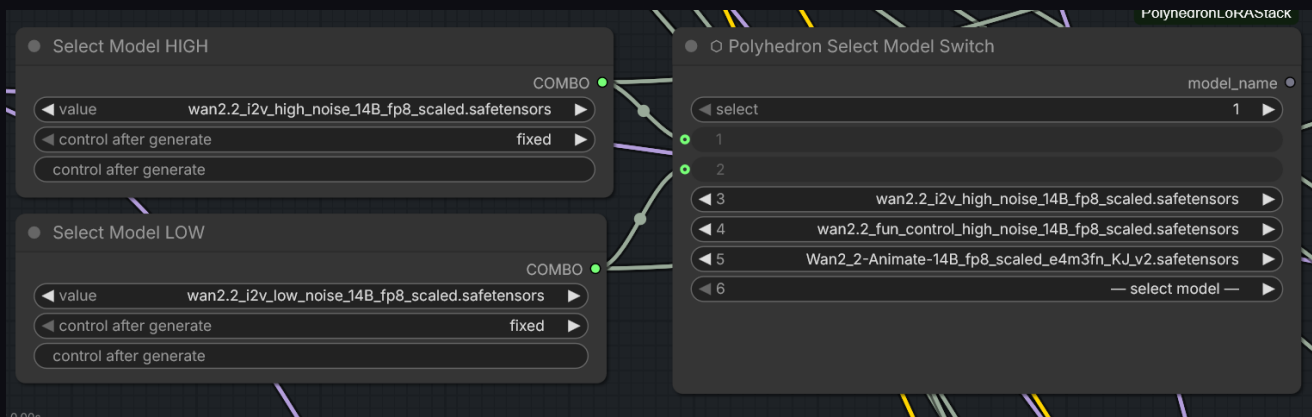
Slots 1 and 2 are **wire inputs** (green dots) — connect a MODEL output from any loader node. Slots 3-6 are **dropdown widgets** built directly into the node — click the arrow to open a searchable list of all models in your `diffusion_models` folder.

10.1 · Inputs, outputs and widgets

Name	Type	Description
1, 2	MODEL (input)	Wire inputs. Connect any loader node output directly.
3, 4, 5, 6	COMBO (widget)	Dropdown selectors. Choose a model filename from the list.
select	INT (widget)	Active slot index (1-6). Only this slot's model is used.
model_name	STRING (output)	Filename of the active model. Connect to a model loader's path input.

10.2 · Wire-input usage

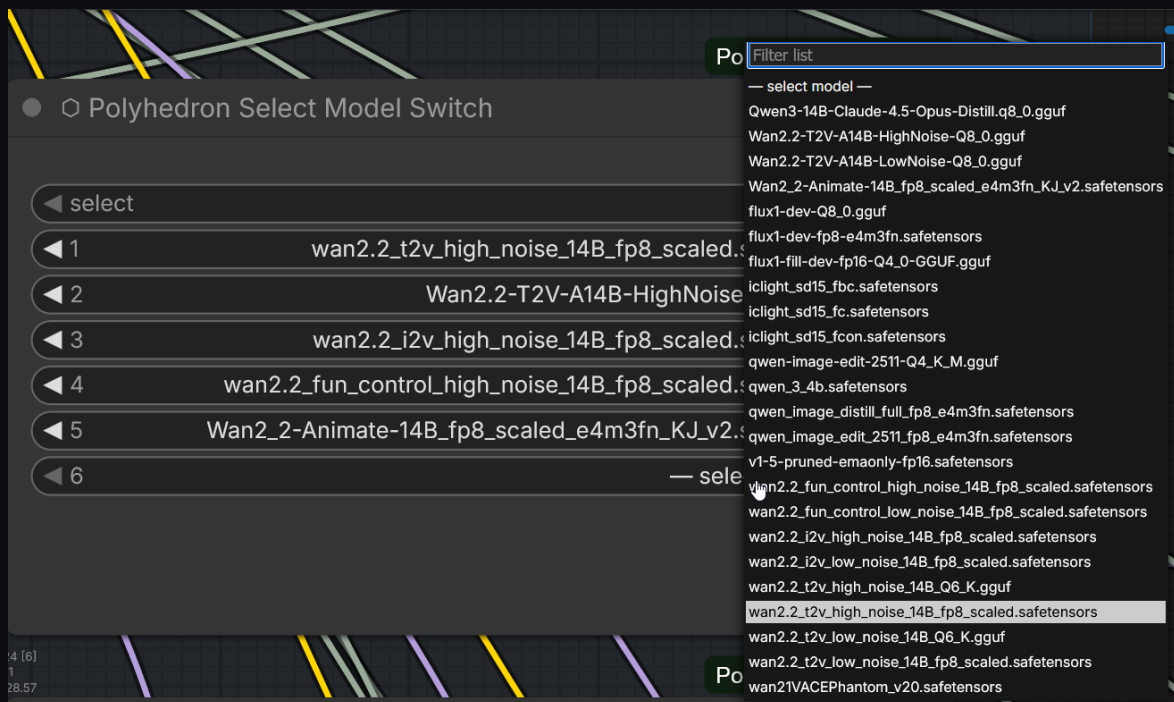
Slots 1 and 2 accept any MODEL connection. A typical use is to wire two model loader nodes — one HIGH and one LOW — directly into the Switch node, with the slot dropdowns 3-6 holding additional variants for quick comparison without rewiring.



Two Select Model nodes (left) wired into a Polyhedron Select Model Switch (right). The Switch holds additional variants in slots 3–5; slot 6 is left as the placeholder fallback.

10.3 · Searchable dropdown

Each dropdown slot (3–6) opens a filterable list of all models in the `diffusion_models` folder. Start typing in the filter field at the top to narrow the list — particularly useful when you have dozens of model variants installed.



The dropdown selector with a filter field at the top. All models in the `diffusion_models` folder are listed and can be searched.

10.4 · Slot 6 placeholder

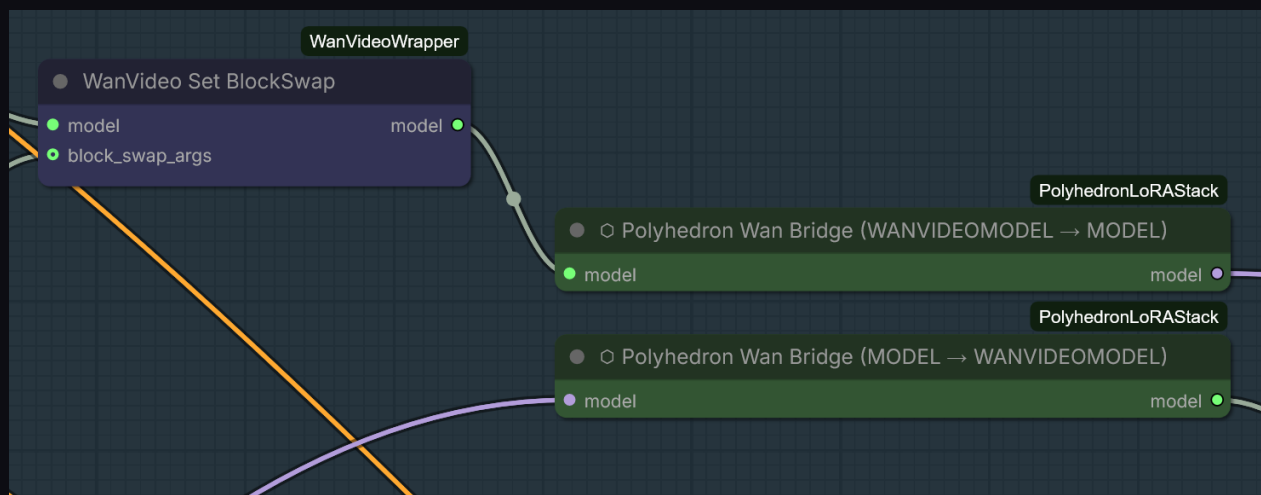
Slot 6 defaults to **— select model —**. Leave it as-is to use it as an intentionally empty fallback, or assign a model to use all six slots.

10.5 · Recommended use in dual-noise workflows

In a WAN 2.2 dual-noise setup, place one Switch node on the HIGH path and one on the LOW path. Wire or select all your HIGH model variants in the first Switch and all LOW variants in the second. Change the select index on both to swap the full model pair in a single edit.

11 · Polyhedron WAN Bridge

ComfyUI uses two distinct model types for WAN workflows: **WANVIDEOMODEL** (the native type used by WanVideoWrapper nodes) and **MODEL** (the standard ComfyUI type used by KSampler, LoRA loaders, and most other nodes). The two Bridge nodes convert between these types so that both ecosystems can be used in the same pipeline.



Both Bridge variants side by side. Top: WANVIDEOMODEL → MODEL (green input dot, lilac output dot). Bottom: MODEL → WANVIDEOMODEL (lilac input dot, green output dot). The colour of the dots indicates the type at that connection point.

Polyhedron Wan Bridge (WANVIDEOMODEL → MODEL)

Takes a WANVIDEOMODEL input and outputs a standard MODEL. Use this when you want to pass a WanVideoWrapper-loaded model into nodes that expect the standard MODEL type — for example the Polyhedron LoRA Stack or a KSampler.

Polyhedron Wan Bridge (MODEL → WANVIDEOMODEL)

Takes a standard MODEL input and outputs a WANVIDEOMODEL. Use this to pass a model that has been processed by standard ComfyUI nodes (e.g. after LoRA patching) back into WanVideoWrapper nodes.

11.1 · Inputs and outputs (both nodes)

Name	Type	Description
model	WANVIDEOMODEL or MODEL (input)	The model to convert. Type depends on which Bridge variant is used.
model	MODEL or WANVIDEOMODEL (output)	The converted model. Passed downstream to the next node.

11.2 · Typical pipeline position

In the standard dual-noise workflow:

- WanVideo model loader → **Bridge (WANVIDEOMODEL → MODEL)** → LoRA Stack → WanMoeKSampler
- LoRA Stack (MODEL output) → **Bridge (MODEL → WANVIDEOMODEL)** → WanVideoWrapper sampler

Stability note

Both Bridge nodes are pure type-cast operations — they do not modify model weights or patches. They have been stable since v232 and are not modified during active development.

12 · Polyhedron Wan Frame Inflate (T2I LoRA fix)

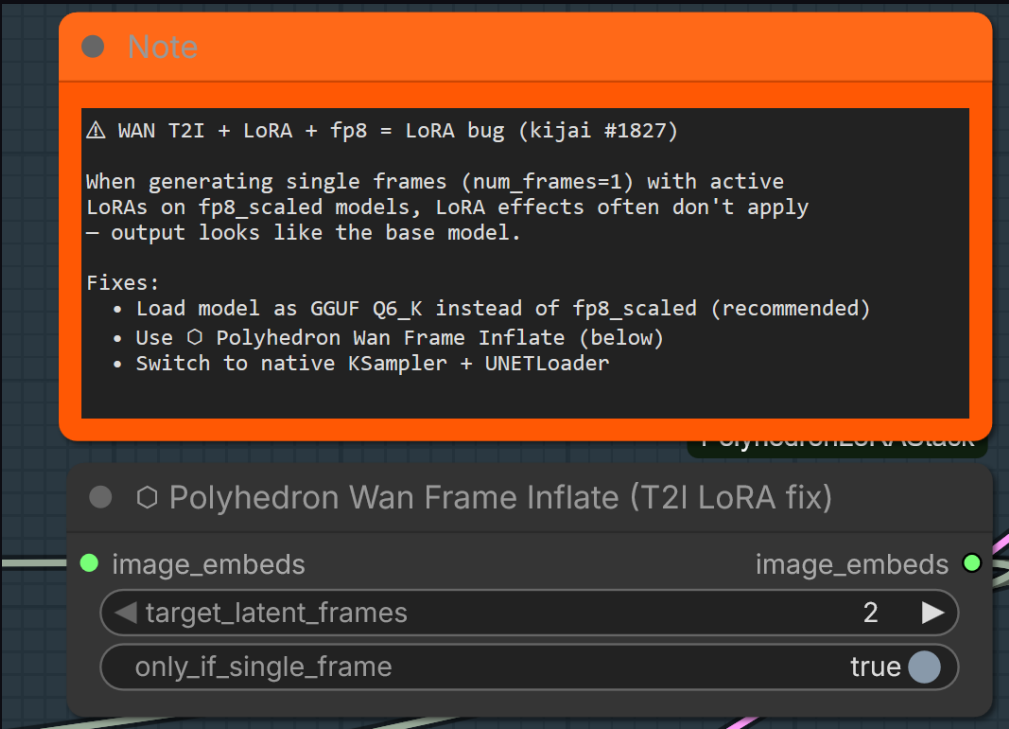
WAN T2I + LoRA + fp8 = LoRA bug (kijai #1827)

When generating single frames (num_frames=1) with active LoRAs on fp8_scaled models, LoRA effects often do not apply — the output looks like the base model without any LoRA influence.

Available fixes:

- Load model as GGUF Q6_K instead of fp8_scaled (**recommended**)
- Use the Polyhedron Wan Frame Inflate node (below)
- Switch to native KSampler + UNETLoader

The Frame Inflate node works by temporarily inflating the **image_embeds** tensor to a multi-frame sequence before sampling, which forces the fp8 model to activate its LoRA-aware code path. After sampling, the extra frames are discarded.



The Frame Inflate node together with the in-workflow note that documents the underlying bug and the three available workarounds.

12.1 · Inputs, outputs and widgets

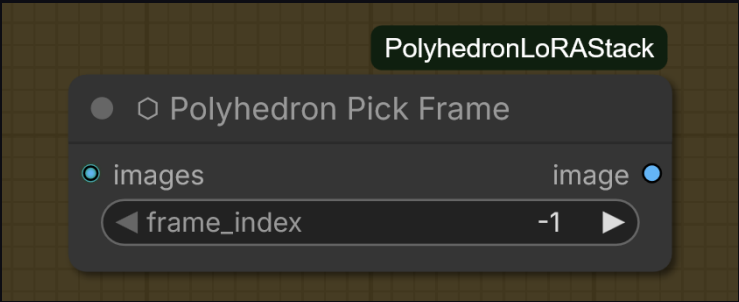
Name	Type	Description
image_embeds	IMAGE_EMBEDS (input)	Connect from the image embedding node upstream of the sampler.
target_latent_frames	INT (widget)	Number of frames to inflate to. Default: 2.
only_if_single_frame	BOOLEAN (widget)	When true, only inflates if the input is a single frame. Default: true.
image_embeds	IMAGE_EMBEDS (output)	Inflated tensor, ready for the sampler.

12.2 · When to use

- You are generating single still frames (T2I mode) with WAN.
 - You are using fp8_scaled model weights.
 - LoRA effects are visibly absent from the output.
- If you have already switched to GGUF Q6_K weights, this node is not needed. It can remain in the pipeline with only_if_single_frame=true without affecting multi-frame video generation.

12.3 · Polyhedron Pick Frame

A small companion to Frame Inflate: it picks a **single frame** out of a decoded image batch — typically the batch that Frame Inflate produced for a T2I run.



The Pick Frame node — frame_index -1 selects the middle frame.

Name	Type	Description
images	IMAGE (input)	The decoded image batch (e.g. from VAE Decode).
frame_index	INT (widget)	Which frame to pick (0-based). Default: -1 = middle frame.
image	IMAGE (output)	The selected single frame.

Why the middle frame? In inflated T2I runs the sampler converges most cleanly on the central frame — the first frame can carry anchor artifacts and the last can be slightly blurred by motion continuity. Out-of-range indexes are clamped; an empty batch passes through unchanged with a console warning.

Typical wiring: `Frame Inflate` → `sampler` → `VAE Decode` → `Pick Frame` → `Save Image`

13 · Sigma Nodes

13.1 · What are sigmas? A plain-language explanation

Diffusion models generate images by starting from pure random noise and gradually removing noise step by step until a clean image emerges. **Sigmas** are simply the noise levels at each step — they tell the model how much noise is still in the image at any given point.

Think of it like developing a photograph in a darkroom: you start in complete darkness (high sigma = lots of noise) and slowly bring up the lights (low sigma = little noise) until the image is fully visible. The schedule is the plan for how fast you bring up the lights.

13.2 · The scheduler

The scheduler decides the **shape** of the sigma curve — how quickly the noise level drops across the steps. Different schedulers produce different curves, which leads to different visual results even with the same model and prompt:

karras

The most widely used schedule. Spends more steps in the mid-noise range, which tends to produce well-structured images with good detail. A safe default for most workflows.

exponential

Drops noise faster in the early steps, slower in the late steps. Can produce sharper results but is more sensitive to CFG and step count.

simple / linear

Equal noise reduction per step. Fast and predictable, but often less detailed than karras. Used in some Lightning/distilled models.

bong_tangent / res_multistep and others

Community schedulers with custom curve shapes. `bong_tangent` front-loads the high-noise steps; `res_multistep` uses residual logic for smoother transitions. Results vary by model — worth experimenting with.

13.3 · The sampler

If the scheduler decides what noise levels to use at each step, the **sampler** decides how to move from one noise level to the next. It is the mathematical solver that performs the actual denoising calculation.

A practical analogy: the scheduler is the map (where you go), the sampler is the vehicle (how you get there).

euler

The simplest solver. Fast, consistent, slightly soft at low step counts. Good general-purpose default.

euler_a (ancestral)

Adds a small amount of random noise at each step, producing more varied and sometimes more creative results. Not fully deterministic — the same seed gives slightly different results each run.

dpm++ / dpm++_2m

Higher-order solvers that take larger, smarter steps. Generally produces sharper results than euler at the same step count. dpm++_2m uses two-step momentum — good balance of speed and quality.

uni_pc

Unified Predictor-Corrector. Fast convergence, good for low step counts. The default in WanMoeKSampler workflows.

lcm / ddim

Designed for distilled / Lightning models. lcm is optimised for 4-8 step generation. ddim is deterministic and good for inpainting workflows.

Scheduler + Sampler together

The two settings are independent but interact. A karras schedule with dpm++_2m is a classic high-quality pairing. A simple schedule with lcm is the standard Lightning setup. For WAN 2.2 with WanMoeKSampler the recommended defaults are simple + uni_pc for Lightning LoRAs, or karras + dpm++ for full-step generation.

13.4 · Rho (ρ) — the shape parameter

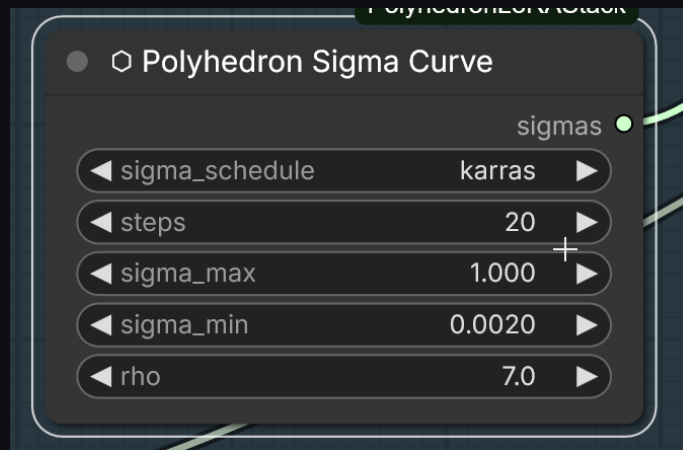
The karras schedule has one additional parameter: **rho**. It controls how curved the sigma schedule is.

- **rho = 1.0** — linear. Equal noise reduction per step.
- **rho = 3.0-5.0** — moderate curve. More steps in the mid-noise range.
- **rho = 7.0** — default. Strong curve, front-loads structure formation.
- **rho > 10.0** — very steep. Most steps are spent at low noise (detail), structure forms quickly.

Higher rho = more detail steps, but less time for large-scale structure. Lower rho = more balanced. The default of 7.0 works well for most cases.

13.5 · Polyhedron Sigma Curve

A standalone sigma schedule generator. It produces a single sigma sequence using a configurable schedule and rho value, replacing the standard BasicScheduler node. Use this in single-model workflows where you want precise control over the schedule shape.



The Polyhedron Sigma Curve node with default values: karras schedule, 20 steps, sigma_max 1.000, sigma_min 0.0020, rho 7.0.

Name	Type	Description
sigma_schedule	COMBO (widget)	Schedule type: karras, exponential, simple, and others.
steps	INT (widget)	Total number of denoising steps. Default: 20.
sigma_max	FLOAT (widget)	Starting noise level. Default: 1.000 (full noise).
sigma_min	FLOAT (widget)	Ending noise level. Default: 0.0020 (near-clean).
rho	FLOAT (widget)	Curve shape. Default: 7.0. See section 13.4.
sigmas	SIGMAS (output)	The computed sigma sequence, ready for the sampler.

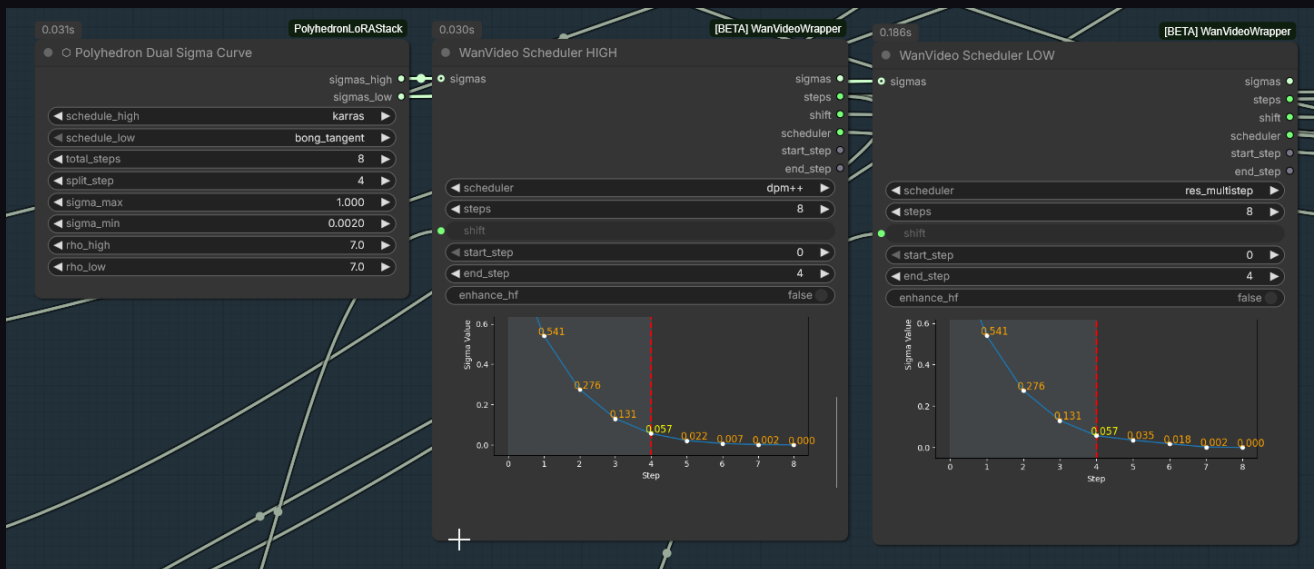
13.6 · Polyhedron Dual Sigma Curve

An extended sigma generator for WAN 2.2 dual-noise workflows. It produces **two separate sigma sequences** — one for the HIGH noise model and one for the LOW noise model — split at a configurable step. Each half uses its own schedule and rho value, giving independent control over the structure-forming phase and the detail-refinement phase.

Name	Type	Description
schedule_high	COMBO (widget)	Schedule for the HIGH noise segment (e.g. karras).
schedule_low	COMBO (widget)	Schedule for the LOW noise segment (e.g. bong_tangent).
total_steps	INT (widget)	Total steps across both segments combined.
split_step	INT (widget)	Step at which HIGH ends and LOW begins. Should match WanMoeKSampler boundary.
sigma_max	FLOAT (widget)	Starting noise level. Default: 1.000.
sigma_min	FLOAT (widget)	Ending noise level. Default: 0.0020.
rho_high	FLOAT (widget)	Curve shape for HIGH segment. Default: 7.0.
rho_low	FLOAT (widget)	Curve shape for LOW segment. Default: 7.0.
sigmas_high	SIGMAS (output)	Sigma sequence for the HIGH noise model.
sigmas_low	SIGMAS (output)	Sigma sequence for the LOW noise model.

13.7 · Connecting to WanVideo Scheduler nodes

In a WAN 2.2 dual-noise pipeline with WanVideoWrapper, the Dual Sigma Curve outputs connect to separate **WanVideo Scheduler HIGH** and **WanVideo Scheduler LOW** nodes. Each Scheduler node displays its sigma curve as a graph with the step values annotated — the red vertical line marks the split point.



Dual Sigma Curve (left) feeding two WanVideo Scheduler nodes — HIGH (centre) and LOW (right). Each Scheduler renders its segment as an annotated sigma graph; the red vertical line marks the split step.

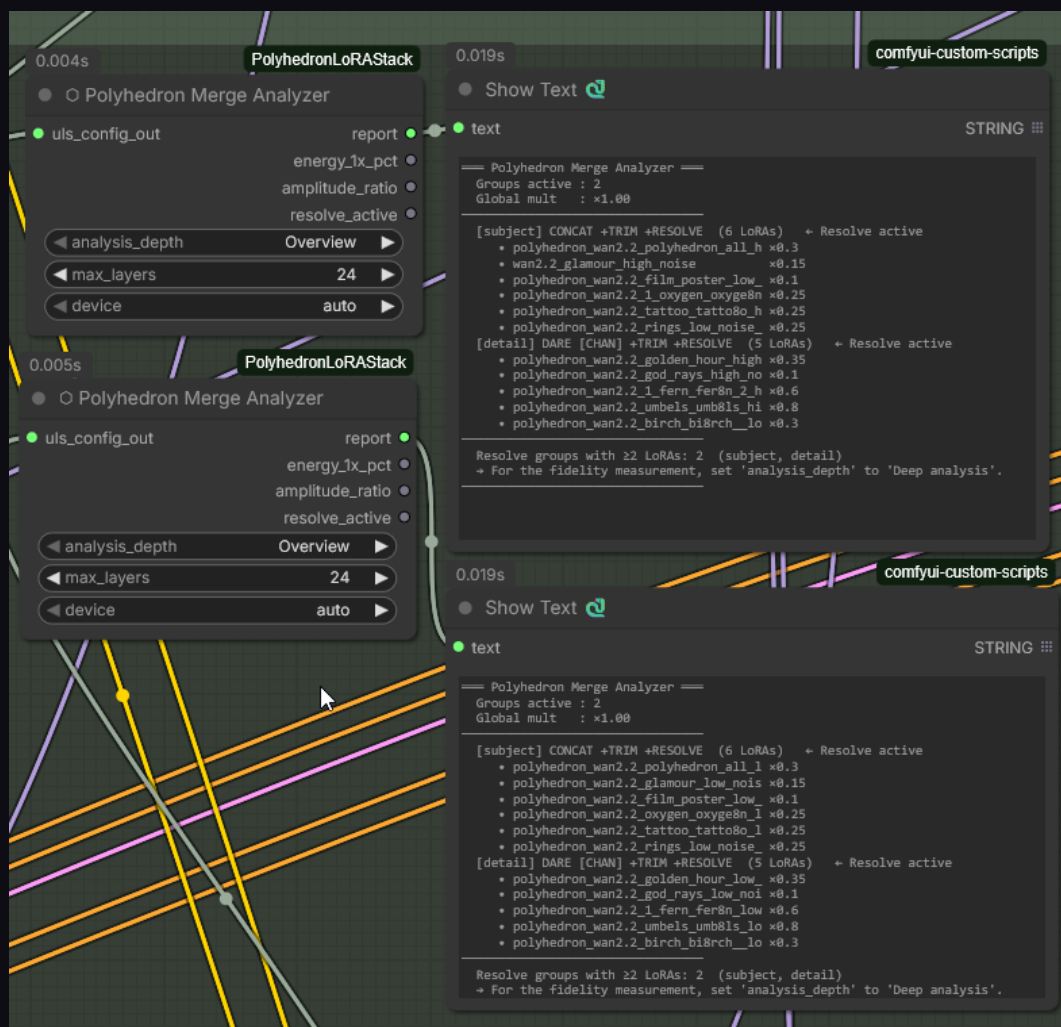
Known issue — LOW sigma chain

In the current release, the LOW sigma segment does not always produce visible refinement under certain scheduler combinations (e.g. `res_multistep` on the LOW path). This is an upstream compatibility issue under investigation. The node is safe to use — it will not corrupt output — but its effect on the LOW path may be neutral until a fix is published.

14 · Polyhedron Merge Analyzer

The Merge Analyzer is a **passive measurement node** for the Stack's Combined / Smooth Mix / Resolve merge path. It reads the Stack's `uls_config_out`, never patches the model, and answers two questions: *what* is the Stack about to merge, and *how faithful* is the Resolve re-pack for the groups that use it.

Use **one Analyzer per Stack** — in a dual-noise workflow that means one for HIGH and one for LOW.



One Merge Analyzer per Stack (HIGH and LOW), each in Overview mode — Stack's `uls_config_out` in, report into a Show Text node.

14.1 · Inputs, outputs and widgets

Name	Type	Description
uls_config_out	STRING (input)	Connect from the Stack's uls_config_out output.
analysis_depth	COMBO (widget)	Overview = instant (selection/modes only). Deep analysis = loads the LoRAs and measures (slower).
max_layers	INT (widget)	Deep analysis: how many of the largest-contribution layers are fully measured. Default: 24.
device	COMBO (widget)	auto = GPU if free (like the real Resolve path), else CPU. cpu forces CPU.
report	STRING (output)	Full formatted report — connect to a Show Text node.
energy_1x_pct	FLOAT (output)	Energy retained at the current rank, aggregated (%).
amplitude_ratio	FLOAT (output)	Amplitude of the re-pack vs. the true sum (1.0 = identical).
resolve_active	BOOLEAN (output)	True when at least one Resolve group with ≥ 2 LoRAs is active.

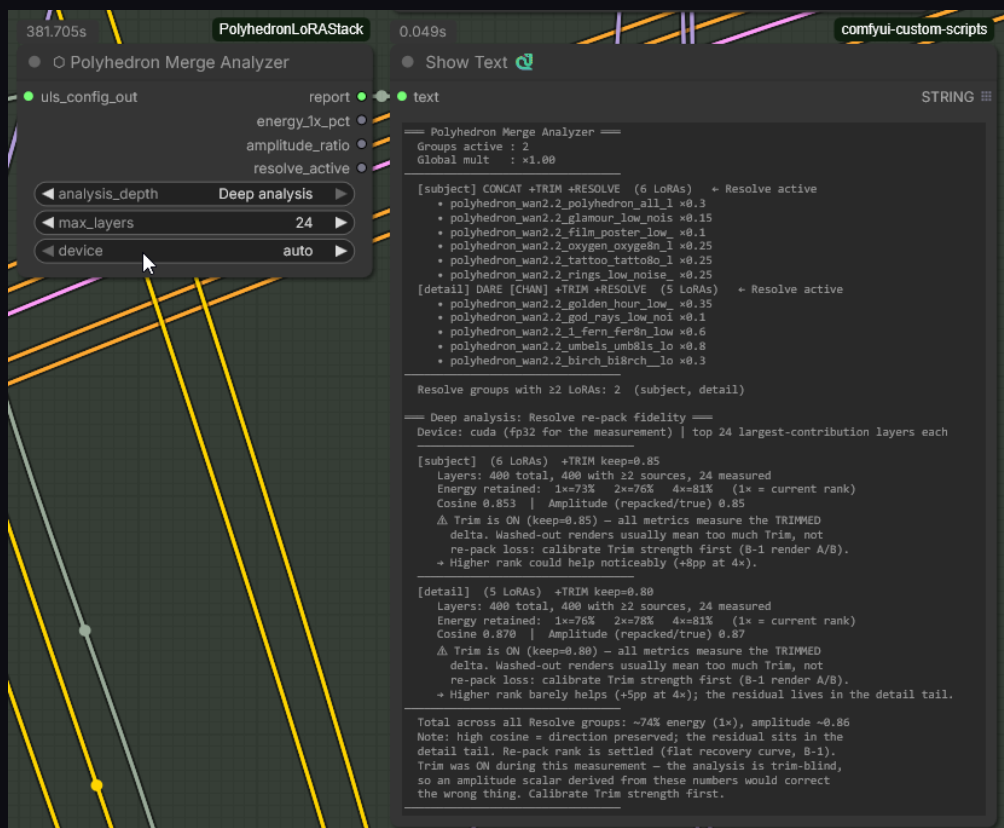
14.2 · Overview mode

Overview is instant. The report lists every active group with its full mode tag (**SEQ** / **CONCAT** / **DARE** [CHAN/ELEM] plus **+TRIM** / **+RESOLVE**), the LoRAs and their weights, and flags groups where Resolve will actually do work ("← Resolve active", i.e. Resolve enabled and ≥ 2 LoRAs). The bottom line tells you which groups can be measured — switch **analysis_depth** to **Deep analysis** for that.

14.3 · Deep analysis

Deep analysis loads the group's LoRAs and measures the Resolve re-pack against the true merged result, per group:

- **Layers** — how many layers exist, how many have ≥ 2 competing sources, how many were measured (capped by **max_layers**, ranked by contribution).
- **Energy retained at 1× / 2× / 4×** — how much of the merged signal survives at the current rank (1×) and what a bigger re-pack would recover (2×/4×).
- **Cosine** — direction match (≈ 0.9 means the merge points the right way).
- **Amplitude (repacked/true)** — overall loudness of the re-pack ($0.90 = \sim 10\%$ quieter).
- A short per-group **verdict** plus a total across all Resolve groups.



Deep analysis report — energy curve, cosine, amplitude, and the Trim warning.

Reading the numbers

A *flat* energy curve (only a few points gained from 1× to 4×) means a higher re-pack rank is not worth it: the residual lives in the fine-detail tail. High cosine = the direction is preserved.

⚠ Trim and the metrics

With Trim ON, all metrics measure the *trimmed* delta (the report warns inline). Washed-out renders are usually **too much Trim**, not re-pack loss — calibrate the Trim strength first (section 3.7).

14.4 · Performance, caching and cancel

- The result is **cached** — the analysis re-runs only when its inputs change, not on every queue.
- Deep analysis prints **live progress** to the console while it works: `[PLS] ANALYZE [group] layer i/N dev layer=...s cum=...s`.
- ComfyUI's red × **cancels** a running deep analysis promptly.
- `device auto` uses the GPU exactly like the real Resolve path; `cpu` forces CPU (slower, but VRAM-free).

```
[PLS] ANALYZE [subject] layer 1/24 cuda layer=8.06s cum=8.1s
[PLS] ANALYZE [subject] layer 5/24 cuda layer=8.13s cum=39.9s
[PLS] ANALYZE [subject] layer 10/24 cuda layer=8.11s cum=80.2s
[PLS] ANALYZE [subject] layer 15/24 cuda layer=7.79s cum=119.3s
[PLS] ANALYZE [subject] layer 20/24 cuda layer=7.77s cum=158.3s
[PLS] ANALYZE [subject] layer 24/24 cuda layer=7.83s cum=189.7s
[PLS] ANALYZE [detail] layer 1/24 cuda layer=7.75s cum=7.7s
[PLS] ANALYZE [detail] layer 5/24 cuda layer=7.93s cum=39.0s
[PLS] ANALYZE [detail] layer 10/24 cuda layer=7.88s cum=78.6s
[PLS] ANALYZE [detail] layer 15/24 cuda layer=7.79s cum=118.2s
[PLS] ANALYZE [detail] layer 20/24 cuda layer=7.98s cum=157.3s
[PLS] ANALYZE [detail] layer 24/24 cuda layer=7.70s cum=188.7s
```

Live [PLS] ANALYZE progress lines in the console during a deep analysis.

© Polyhedron · All rights reserved

Civitai: civitai.com/user/Polyhedron_AI · Patreon: patreon.com/c/polyhedron_ai

Part III — CLIP Text Encode

Polyhedron Suite · User Documentation

One node in place of a small chain. The positive prompt is assembled from several fields instead of one wall of text, so a trigger block, the subject and a style block can be edited without hunting through a paragraph. A negative prompt sits in the same node and can be switched off. Both are encoded through ComfyUI's own encoder — reused, never re-implemented — and the composed strings come back out as text, so what the encoder saw is what you can read.

- 1 · Overview
- 2 · Segments — the prompt in parts
- 3 · Comments — headings that cost nothing
- 4 · The negative
- 5 · External text
- 6 · The footer — words, characters, tokens

1 · Overview

Where a normal setup needs a text node, two encoders and a bit of wiring, this is one node with everything in view: the prompt in parts, the negative beside it, the cleanup switches on top and a live count at the bottom.



Two positive segments, the negative below them, the cleanup switches on top and the live footer at the bottom.

Inputs and outputs

Pin	Direction	Carries
clip	in	The CLIP / text encoder — the same pin as the core node.
pos_external	in	Optional text from elsewhere: a captioner, a Stack’s trigger words, any STRING source. Folded into the positive prompt (section 5).
neg_external	in	Optional external text for the negative prompt.
positive	out	The encoded positive conditioning.
negative	out	The encoded negative conditioning — valid even when the negative is switched off.
positive_text	out	The composed positive prompt, exactly as encoded.
negative_text	out	The composed negative prompt, exactly as encoded.
full_text	out	Both prompts in one string — for logging, or a Save node’s metadata.

The controls at a glance

segments sets how many positive fields take part. **separator** decides how they are joined into one prompt. **strip_comments** with **comment_markers** removes your section headings before encoding; **strip_newlines** flattens the remaining line breaks. **use_negative** turns the negative box on and off. **external_mode** decides whether wired-in text is appended, prepended, or replaces the fields entirely.

The footer keeps count while you type — words and characters — and adds the token figure from the last run.

2 · Segments — the prompt in parts

`segments` decides how many positive fields take part. Turn it up and another empty field appears; turn it down and the field disappears from the node.

Hiding is not deleting. The text of a hidden segment stays with the node, survives saving and comes back untouched when you turn the count up again. But while it is hidden it is also **out of the prompt** — the encoder only ever sees the fields that are showing.

That makes the control useful for more than tidiness: park a whole block of styling in segment 3, turn the count down to 2, and you have switched it off without losing a word. The footer confirms it immediately, because the word count drops by exactly that block.

Polyhedron CLIP Text Encode

- clip positive
- pos_external negative
- neg_external positive_text
- negative_text
- full_text

external_mode append

strip_comments On

strip_newlines On

separator comma

comment_markers //

segments 1

```
// SCENE & CAMERA
quiet daylight studio, 50mm lens, shallow depth of field, eye-level shot
// BACKGROUND
plain warm grey wall, soft falloff to the corners
// CLOSING
natural colour, fine film grain, no heavy retouching
```

use_negative On

```
// EYES & GAZE
crossed eyes, mismatched pupils, dead stare, glassy highlights
// ANATOMY & BODY
extra limbs, fused shoulders, elongated neck, broken proportions
// FACE
melted features, asymmetric jaw, waxy skin, plastic sheen
// MOTION ERRORS
motion smear, ghosting, duplicated silhouette, frozen mid-blur
// QUALITY & TECHNICAL
low resolution, jpeg artefacts, banding, oversharpening, heavy noise
// STYLE
cartoon, anime, 3d render, oil painting, illustration
// SUBJECT
multiple people, cropped head, back turned to camera
// COMPOSITION & ARTIFACTS
watermark, signature, text overlay, border, frame
```

pos 28 words · neg 60 words · 177 chars · run to count tokens

The same node with `segments` turned down to 1: the second block is out of sight and out of the prompt. The footer drops from 99 words to 28.

The boxes grow with the text

A prompt field is never a peephole. Each visible box sizes itself to what is in it — type another line and the box gets a line taller, delete one and it shrinks back — and the node follows, so the whole prompt is readable without scrolling inside a two-line window.

An empty field still keeps about two lines of height, so it is easy to hit with the mouse. Growth stops at a generous ceiling: paste something monstrous and the box scrolls at that point instead of turning the node into a mile-high column. Between those two limits the rule is simply that what you typed is what you see.

Joining the parts

`separator` decides how the fields are joined into the single prompt that gets encoded. `comma` puts `,` between them — the usual choice. `newline` keeps them on separate lines, `space` runs them together, `none` concatenates them with nothing in between.

The separator applies *between* segments, not inside them; whatever you typed within one field stays as it is.

3 · Comments — headings that cost nothing

Long prompts get unreadable fast. `strip_comments` lets you keep section headings in the box that never reach the model: everything from a comment marker to the end of that line is removed before encoding.

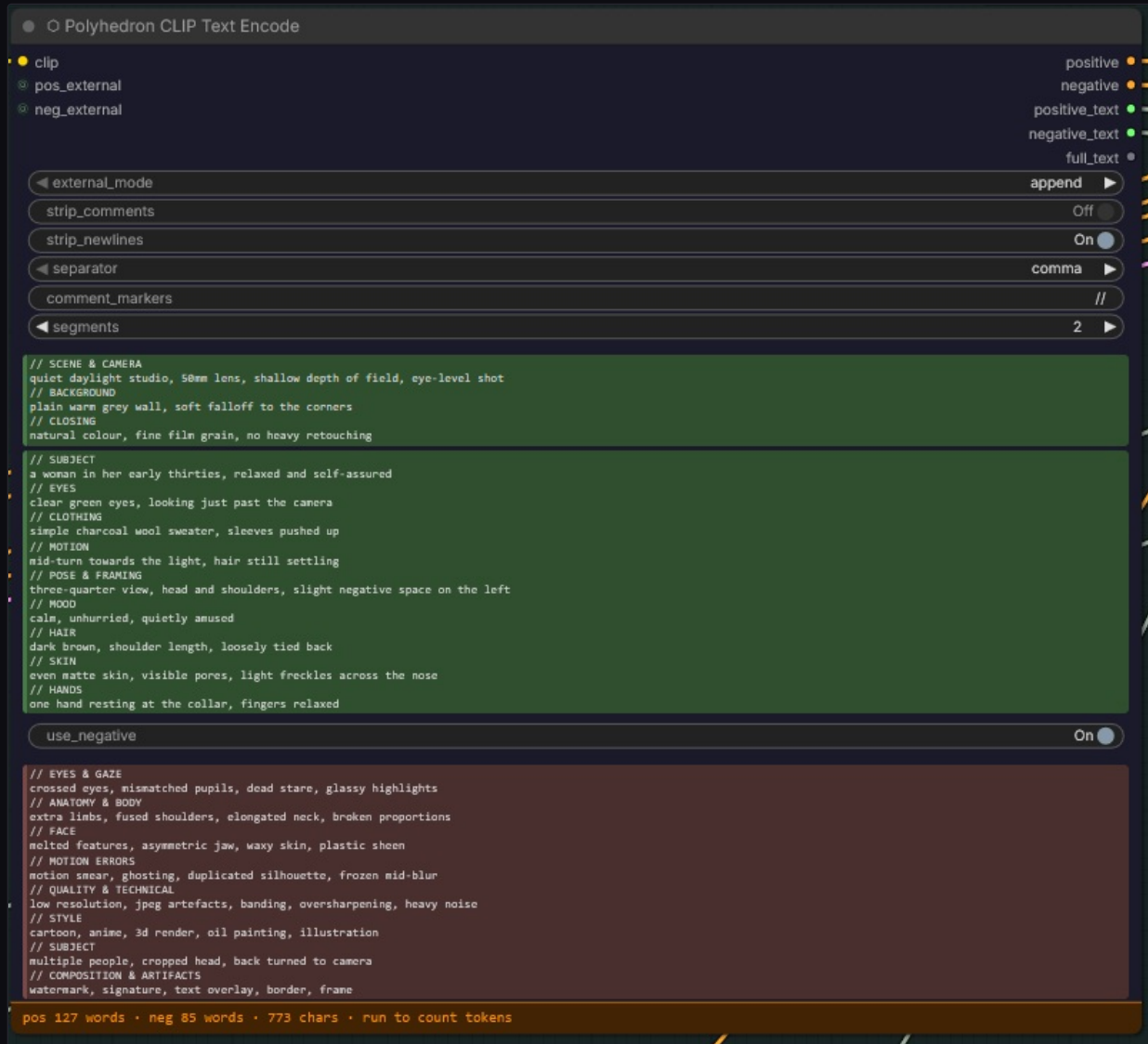
That is what makes a structured prompt possible. `// SCENE & CAMERA`, `// SUBJECT`, `// HANDS` — they organise the prompt for you and cost nothing at the encoder. Switch the control off and they become part of the text like any other words; the footer shows it at once.

```
// SCENE & CAMERA
quiet daylight studio, 50mm lens,
shallow depth of field, eye-level shot
// BACKGROUND
plain warm grey wall, soft falloff to the corners
// CLOSING
natural colour, fine film grain, no heavy retouching
```

What you write. The `//` lines are yours — headings, reminders, anything you want beside the prompt rather than in it.

```
quiet daylight studio, 50mm lens,
shallow depth of field, eye-level shot,
plain warm grey wall, soft falloff to the corners,
natural colour, fine film grain, no heavy retouching
```

What the encoder receives. Every marked line is gone and what is left is joined by the separator. Nothing else was touched — and nothing you wrote for yourself became a token.



The same prompt with `strip_comments` switched off. The headings now count: 127 words instead of 99, and 773 characters instead of 649.

What comments are for

Two things, and both of them matter once a prompt grows past a few lines.

Structure. A long prompt is a list of decisions — scene, camera, subject, clothing, mood, hands. Written as one paragraph it becomes unsearchable within a week; you end up re-reading the whole thing to change the lens. With headings the prompt gets a table of contents: you scroll to `// CLOTHING`, change one line, and leave everything else alone. The prompt in this manual is built exactly that way, and none of those headings reaches the encoder.

Notes to yourself. The second use is the one people discover later. Anything you want to keep *next to* the prompt but not *in* it goes behind a marker: the seed that worked, the LoRA this wording was written for, a line you took out but might want back, a reminder that the model ignores a certain word. It travels with the workflow, it is there when you open the file in three months, and it costs nothing at generation time.

Why this is not a small thing. Every word in the prompt competes for the encoder's attention, and every text encoder has a limit. Notes and headings left in a prompt do not merely take up room — they are read as content, and a heading like `SUBJECT` or a stray `TODO: try 0.8` is exactly the kind of stray noun that shows up in the picture. Comments let you keep the documentation and pay nothing for it.

Choosing the markers

`comment_markers` decides what counts as a marker. The default is `//`, borrowed from code, because it is rare in ordinary prompt text. Enter a **space-separated list** to use several at once — `// # ***` gives you three, and any of them opens a comment.

Use whatever suits your habits. `#` if you come from Python or YAML, `--` from SQL, `;` or `***` if you want something that never occurs in your prompts. Several markers are useful when they mean different things to you: `//` for section headings and `#` for private notes, say, so you can find the notes again with a search. The node does not distinguish between them — that meaning is yours — it strips them all alike.



The tooltip states the rule: a space-separated list, a marker counts at the line start or after a space, and an empty field strips nothing.

Situation	What happens
// SUBJECT	Line start — the whole line is a comment and goes.
//SUBJECT	The same. There is no leading-space rule to remember.
red dress // maybe blue	The marker follows a space, so the note goes and red dress stays. Comments do not have to sit on their own line.
https://example.com	Survives. The // here follows a colon, neither a line start nor a space — which is precisely why the rule is written that way.
empty field	Nothing is stripped at all, whatever strip_comments says.

A comment always runs to the end of its line — there is no closing marker and no block comment. That is deliberate: it keeps the rule small enough to hold in your head while you write.

Why comments go first

Comments are removed line by line, and only afterwards does strip_newlines flatten the remaining line breaks into spaces. The order matters: flatten first and a heading like // SUBJECT would swallow the whole line that follows it, because after flattening there is no line end left to stop at. The node does it the safe way round, so you can write headings above their content without thinking about it.

strip_newlines itself is the tidy-up afterwards: your prompt keeps its readable shape in the box, while the encoder gets one continuous line.

4 · The negative — there, or honestly empty

`use_negative` switches the negative prompt off. The box disappears from the node and the footer drops to zero words.

What does not disappear is the `negative` output. It still carries a valid conditioning — an encoding of an empty prompt — so anything wired to it keeps working. Nothing to unplug, nothing to re-route, no red link when you change your mind mid-workflow. Switch it back on and your text is still in the box; turning the negative off hides it the same way a hidden segment is hidden, and it comes back untouched.

This matters for models that are meant to run without a negative. Some distilled and Lightning setups do their best work with an empty negative and CFG at 1.0; with a normal node you either leave an empty box lying around or rebuild the wiring. Here it is one switch, and the graph does not notice.

The screenshot shows the 'Polyhedron CLIP Text Encode' node in a workflow. On the right, a vertical list of outputs includes 'positive', 'negative', 'positive_text', 'negative_text', and 'full_text'. The 'negative' output is highlighted with an orange dot. Below the node, a large green box contains a detailed prompt. At the bottom, a 'use_negative' toggle switch is set to 'Off'. The footer bar displays 'pos 99 words · neg 0 words · 649 chars · run to count tokens'.

external_mode append
strip_comments On
strip_newlines On
separator comma
comment_markers //
segments 2

```
// SCENE & CAMERA
quiet daylight studio, 50mm lens, shallow depth of field, eye-level shot
// BACKGROUND
plain warm grey wall, soft falloff to the corners
// CLOSING
natural colour, fine film grain, no heavy retouching

// SUBJECT
a woman in her early thirties, relaxed and self-assured
// EYES
clear green eyes, looking just past the camera
// CLOTHING
simple charcoal wool sweater, sleeves pushed up
// MOTION
mid-turn towards the light, hair still settling
// POSE & FRAMING
three-quarter view, head and shoulders, slight negative space on the left
// MOOD
calm, unhurried, quietly amused
// HAIR
dark brown, shoulder length, loosely tied back
// SKIN
even matte skin, visible pores, light freckles across the nose
// HANDS
one hand resting at the collar, fingers relaxed
```

use_negative Off

pos 99 words · neg 0 words · 649 chars · run to count tokens

`use_negative` off: the box is gone and the footer reads `neg 0 words`. The `negative` output still delivers a valid, empty conditioning.

5 · External text — prompts from elsewhere

Two optional inputs take text from other nodes: `pos_external` feeds the positive prompt, `neg_external` the negative. Anything with a `STRING` output fits — a captioner, a Stack handing over its trigger words, or a plain text node.

Mode	Result
append	The external text goes behind your segments. The default — your own wording leads, the incoming text follows.
prepend	It goes in front. Useful for trigger words that a LoRA wants to see early in the prompt.
replace	It takes over entirely. The segments stay in the node, untouched and saved, but the prompt is the external text alone.

The mode applies to both sides at once — there is no separate setting for the negative.

Where that text comes from

The obvious source is another prompt fragment you keep somewhere else. The more interesting one is a node that **writes the text for you**.

Vision models do exactly that. A captioner such as **Florence2** looks at an image and returns a description of it as a `STRING` — the subject, the setting, the lighting, in as much detail as you ask for. Wire that output into `pos_external` and the description becomes part of the prompt without you typing a word of it. The same works for any interrogator or tagger that hands back text: WD14-style taggers, an LLM node that rewrites a sketch of an idea into a full prompt, a node that reads the caption sidecar of a training image.

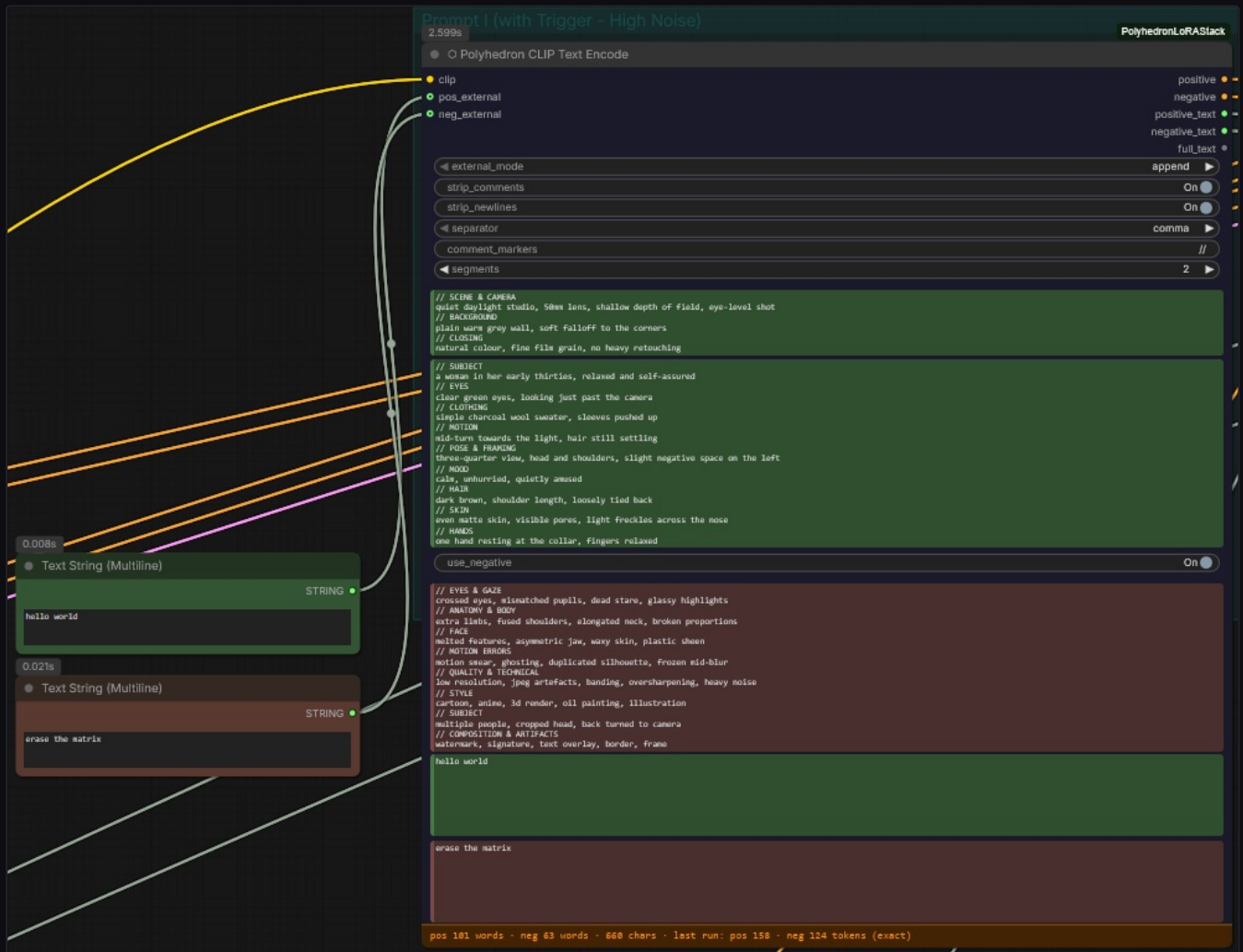
That turns the three modes into three quite different workflows:

With a captioner on <code>pos_external</code>	Result
append	Your segments set the frame — camera, style, quality — and the description of the reference image follows. The usual choice for image-to-image and for style transfer: you keep control of the look, the picture supplies the content.
prepend	The description leads and your own wording refines it afterwards.
replace	The caption <i>is</i> the prompt. Useful for batch work: point a captioner at a folder and every image is re-generated from its own description, while your segments sit in the node untouched, ready for when you switch the mode back.

Because the resolved text appears in the node after the run, you can always see what the captioner actually said — and the footer counts it, so a chatty vision model that quietly eats half your token budget shows up as a number instead of a mystery.

Seeing what arrived

Wire an input and a field for it appears in the node. Until the first run it shows a placeholder, because the text only exists once the graph has actually delivered it; run once and the resolved text is there to read. From then on it is counted with everything else in the footer, so the word and token figures tell the truth about the whole prompt, not just the parts you typed.



Two text nodes feeding `pos_external` and `neg_external` in `append` mode. After the run their text shows in the node and counts in the footer: 101 words instead of 99, 63 instead of 60.

One thing to know. `neg_external` only works while the negative is switched on. With `use_negative` off, the negative prompt is not assembled at all, and a wired external negative is silently ignored. If a negative input seems to do nothing, that switch is the first place to look.

6 · The footer — words, characters, tokens

The strip along the bottom of the node is the honest account of what will be encoded. It reads, for example: `pos 99 words · neg 60 words · 649 chars · last run: pos 155 · neg 119 tokens (exact) .`

Two clocks, not one

Words and characters are counted **as you type**, and they already respect your cleanup settings — with `strip_comments` on, your section headings are not in the figure. Resolved external text is included too, so the count describes the whole prompt.

The token figure comes from the **last run**. Tokens can only be counted by the tokenizer that belongs to your text encoder, and that lives in the backend; so after every edit the field falls back to `run to count tokens` until the graph has run again. That is not a glitch — it is the difference between a guess and a measurement.

Reading	Means
<code>exact</code>	Counted with the encoder's own tokenizer.
<code>heuristic</code>	The tokenizer was not reachable; the figure is a calibrated estimate.
<code>run to count tokens</code>	The text changed since the last run.

The count comes from the same function behind the Polyhedron Token Counter, so the two never disagree. This node only ever *displays* the number; the Token Counter remains the place for limits and warnings.

The text outputs

`positive_text` and `negative_text` hand back exactly the strings that were encoded — after the comment pass, after flattening, with the segments joined and any external text folded in. Wire one into a Token Counter to enforce a limit, into a Show-Any node to read it, or into the Save node so the prompt that actually ran is written beside the picture. `full_text` carries both at once for logging.

Why this exists. A prompt that goes through several nodes before it reaches the encoder is easy to lose track of. These outputs close that gap: what the encoder saw is what you can read, character for character.

Part IV — Sampler

Polyhedron Suite · User Documentation

One node for both sampling architectures. In **Single** it is a superset of KSampler and KSampler (Advanced) — a real denoise field beside the step window, the full sampler and scheduler lists, add_noise. Switch the mode pill and the same node runs a noise-split **two-expert** denoise, the architecture Wan 2.2 ships. Nothing is rebuilt in between: what does not apply goes dim, everything else stays where your eye expects it. And the picture inside the node is a live preview that can animate the whole clip while it denoises.

- 1 · Overview
- 2 · Single — the everyday case
- 3 · High + Low — two experts, one node
- 4 · Live preview — watching it work
- 5 · Putting it together

1 · Overview

If you have used the KSampler, you already know most of this node. What it adds is everything that normally costs a second node or a different one: the advanced step window, a second expert, and a preview worth looking at.

Pin	Direction	Carries
model	in	The model. In <code>High + Low</code> this is the high-noise expert.
positive / negative	in	Conditioning to include and to exclude.
latent_image	in	The latent to denoise — image or video.
model_low	in	Optional. The low-noise expert; connect it and switch the mode pill to <code>High + Low</code> .
sigmas · sigmas_high · sigmas_low	in	Optional external schedules. When connected they override the built-in schedule — not covered in this edition.
noise	in	Optional external noise source.
LATENT	out	The denoised latent.



The node in `High + Low`. The Single-only controls near the bottom are dimmed, not hidden — you always see the whole node and read from the grey what does not apply right now.

2 · Single — the everyday case

`seed`, `steps`, `cfg`, `sampler_name`, `scheduler` and `denoise` work exactly as you know them.

The three controls that usually cost a second node are here too. `add_noise` decides whether the run starts by adding noise at all — off when you are continuing a latent that already carries it. `start_at_step` and `end_at_step` cut a window out of the schedule, so a run can begin partway in or stop early. `return_with_leftover_noise` hands the latent on **unfinished** on purpose, for a second sampler to complete — the classic two-stage refiner setup, without swapping in the Advanced variant.

In `High + Low` these three go dim: the split is then decided by the Handoff, not by hand.

Sigma shift

`sigma_shift` applies a flow-matching shift to the model before the schedule is built. `0.00` means **off** — the node leaves the model's sampling untouched, which is correct when an upstream `ModelSamplingSD3` node or the model itself already provides one.

Wan 2.2 needs a shift. Without it a short run cannot converge and leaves residual noise that shows up as fine RGB speckle in flat areas. `8.0` matches the Wan MoE KSampler; `5.0` is the older Wan 2.1 convention and is too low for 2.2. A positive value here **overrides** rather than adds, so it takes precedence over anything set upstream.

The low expert's own shift

Directly underneath sits `sigma_shift_low`, the low expert's twin. Its default of `-1` reads as *same as high*: the low expert simply follows `sigma_shift`, which is how the node behaved before this widget existed — an older workflow loads and runs exactly as it did. `0` switches the shift off for the low expert alone, even while the high expert carries one. Any positive value is its own shift.

The case this was built for: Wan 2.2 with `8.0` on the high expert and `5.0` on the low one. That split used to need two separate `ModelSamplingSD3` nodes wired in front of the sampler — now it is two fields, and those nodes can come out of the graph.

Where it bites. Like `scheduler_low`, this widget only takes effect in *Wan MoE parity*, where the low segment's schedule is built from the low expert. In *Continuous* both experts share **one** schedule, built from the high expert, so the field greys out — and if you set a value there anyway, the console says it is inert rather than letting you believe otherwise. In *Single* it greys out too, and an external `SIGMAS` curve switches it off along with the rest of the schedule controls.

3 · High + Low — two experts, one node

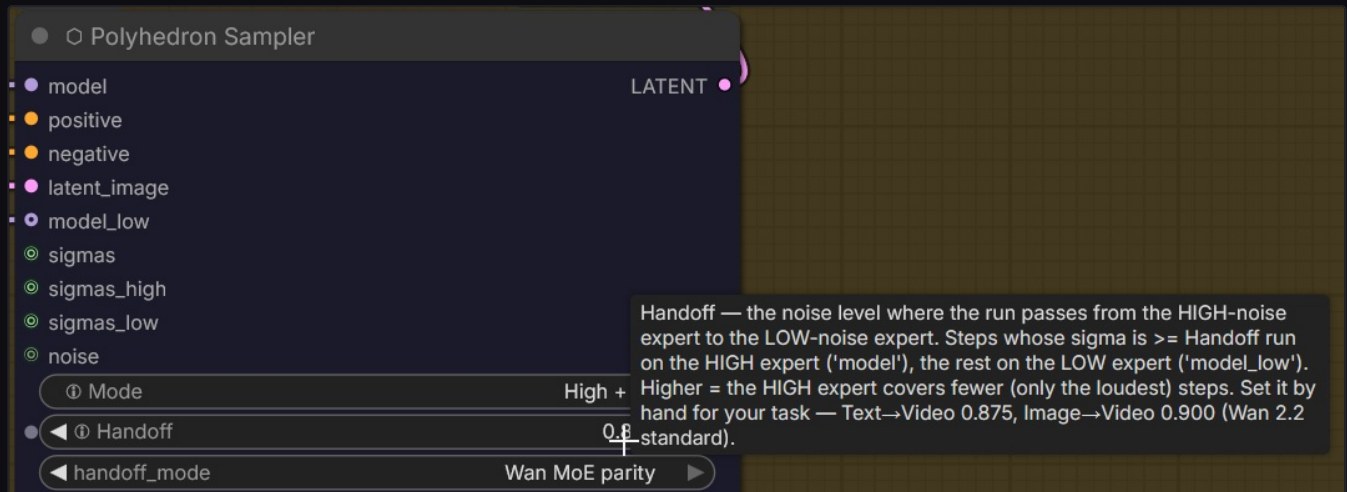
Some model families ship as a **pair**: one expert trained for the loud, early part of the denoise, one for the quiet, detailed end. Wan 2.2 is the prominent example.

The `Mode` pill switches the node from `Single` to `High + Low`, `model` becomes the high-noise expert, and `model_low` takes the second one. Everything else stays where it is — you do not rebuild the graph to change sampling architecture.

The Handoff

This is the one number that matters in this mode. It is a **noise level**, not a step count: every step whose sigma is at or above the Handoff runs on the high-noise expert, the rest on the low-noise expert. Raise it and the high expert covers fewer steps — only the loudest ones; lower it and it keeps working further into the run.

There is no automatic value, and that is deliberate: the right split depends on the task, not on the file. The Wan 2.2 conventions are **0.875 for text-to-video** and **0.900 for image-to-video**. Start there and move it if the result tells you to — too much high expert and fine detail stays mushy, too little and structure or motion never settles.



The Handoff is a noise level, not a step: everything at or above it runs on the high expert.

`cfg_low` gives the second expert its own guidance value, because the two halves rarely want the same one; distilled and Lightning LoRAs typically run the low pass at 1.0. `sampler_low` and `scheduler_low` do the same for sampler and schedule and default to *same as high*, so you only touch them when you actually want a different one.

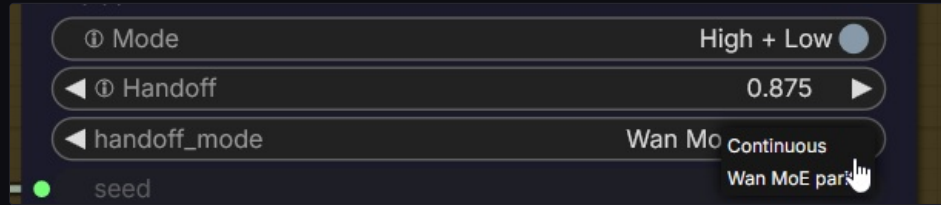
3 · High + Low (continued)

Two ways to hand over

`handoff_mode` decides what happens at the seam.

Continuous runs both experts as **one unbroken denoise**. The high expert stops while leftover noise is still present and the low expert picks it straight up — mathematically the clean continuation, and this node's original behaviour.

Wan MoE parity reproduces the Wan MoE KSampler exactly. The switch happens **one step earlier**, the high expert finishes on a clean estimate of the final image, and the low expert re-noises that estimate with the *same* seed noise before denoising to the end. Use it when you want results that match the Wan reference workflow rather than the mathematically tidier path.



Two seams: the unbroken continuation, or exact parity with the Wan MoE KSampler.

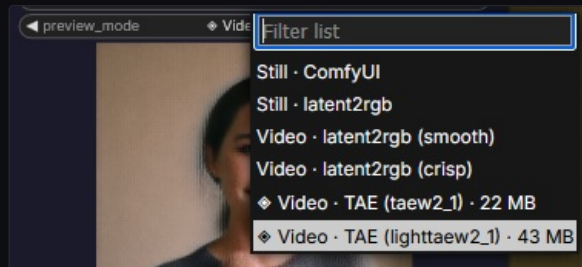
Changing the mode re-plans the run. Both the mode pill and `handoff_mode` recompute what is active the moment you touch them, so the greyed-out controls always tell the truth about the run you are about to start.

4 · Live preview — watching it work

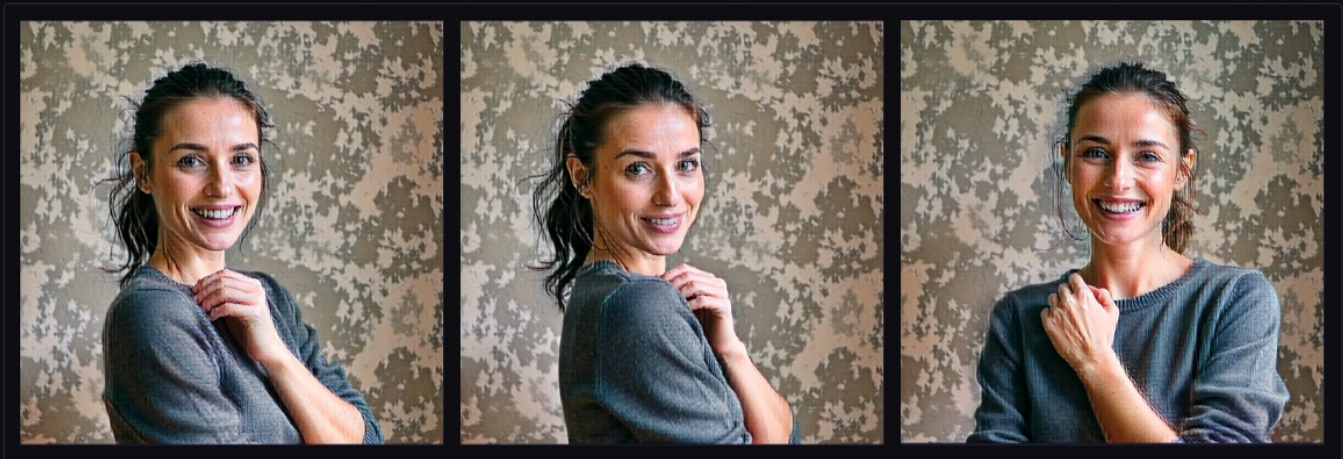
`preview_mode` picks the decoder for the picture inside the node. It is visualisation only: whatever you choose, the latent that leaves the node is identical.

Still · shows a single frame that refreshes each step — the behaviour you know from the KSampler. `Still · ComfyUI` hands the job to ComfyUI's own previewer and honours your `--preview-method` setting; `Still · latent2rgb` forces the model-free view, which costs nothing and needs no files.

Video · animates *every frame* of the predicted latent. On a still image there is one frame and nothing moves; on a video the preview plays the whole clip while it is being denoised, so you see the motion long before the run ends. `Video · latent2rgb` comes in a smooth and a crisp variant. The two `Video · TAE` modes decode through a small approximate decoder instead: slower per step, but far closer to the finished picture.



Six decoders. The two TAE entries carry their download size, because they are files that have to be on disk. The list filters as you type.



Three moments of the same run in a `Video ·` mode: the preview is not one frame refreshing — the whole clip plays while it denoises.

The TAE decoders

Pick a TAE mode whose file is missing and the node says so with a small amber notice instead of failing silently, and offers to fetch it. The download goes through the pack's one download door: staged to a temporary file, checksum-verified, then moved into place, so an interrupted download never leaves a broken decoder behind.

Switching mid-run

The mode can be changed **while a render is running**. The change is picked up on the next step — useful when a long video run has started and you decide you want the better decoder after all. Again: preview only, the result does not change.

5 · Putting it together

A plain image

Model, conditioning and an empty latent in, mode on `Single`, sampler and scheduler to taste. Nothing else needs touching.

Wan 2.2 video

Both experts in — the high one on `model`, the low one on `model_low` — mode on `High + Low`, Handoff at `0.875` for text-to-video or `0.900` for image-to-video, `sigma_shift` at `8.0` unless it comes from upstream. Distilled or Lightning LoRAs usually want `cfg` and `cfg_low` at `1.0`. Pick a `Video` · preview mode and you can watch the motion settle instead of guessing.

A two-stage refine

First sampler in `Single` with `end_at_step` set and `return_with_leftover_noise` on; the second picks the latent up with `add_noise` off and `start_at_step` where the first one stopped.

With a Polyhedron CLIP Text Encode in front

Its `positive` and `negative` go straight into this node's conditioning inputs. Because that node also hands back the composed prompt as text (Part III, section 6), you can wire `full_text` into a Save node and keep the exact prompt beside the picture that came out of this sampler.

Not covered here. The `sigmas`, `sigmas_high` and `sigmas_low` inputs accept an external schedule and override the built-in one entirely. They are deliberately left out of this edition.

Part V — Upscale & Interpolate

Polyhedron Suite · User Documentation

Three nodes for what happens **after** the sampler. **Fast Upscale** resizes — image, video and mask together, so the mask never drifts out of register. **Power Upscale** is the slow, thorough one: it cuts the frame into overlapping tiles and re-samples each of them, with a second expert if you want one. **Interpolate** invents the frames between your frames. All three report what they are doing while they do it, because on a long clip the difference between working and hanging should never be a guess.

- 1 · Fast Upscale — size without invention
- 2 · Power Upscale — the thorough one
- 3 · Power Upscale — watching it work
- 4 · Interpolate — the frames in between
- 5 · Choosing between them

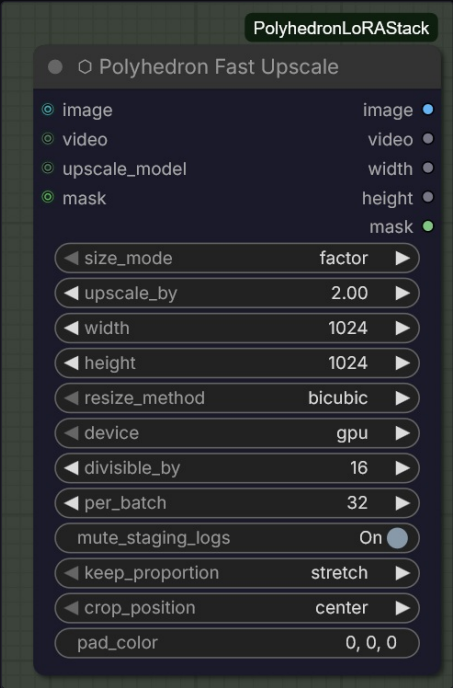
1 · Fast Upscale — size without invention

A resize, optionally through an upscale model, and nothing else. No diffusion, no sampler, no seed. A 65-frame clip is done in seconds rather than minutes, and the result is exactly as detailed as what you put in — only bigger.

Pin	Direction	Carries
image	in	Frames. One for a still, many for a clip.
video	in	A video object, as an alternative to loose frames.
upscale_model	in	Optional. An ESRGAN-family model. Without it the node is a pure resize.
mask	in	Optional. Scaled with the image, by the same rule.
image · video · mask	out	The scaled result, in all three shapes.
width · height	out	The dimensions actually produced — after <code>divisible_by</code> has had its say.

The controls

Control	What it does
size_mode	<code>factor</code> multiplies by <code>upscale_by</code> ; the other modes use <code>width</code> and <code>height</code> . The fields that do not apply stay visible, so you can always read what the node would do in the other mode.
resize_method	The filter. Most run on the GPU; lanczos (cpu) is honestly labelled because it is a PIL path and cannot run anywhere else. Ask it for a GPU device and it refuses out loud instead of quietly doing something different.
device	Where the resize runs. On the GPU it is dramatically faster.
divisible_by	Snaps the result to a multiple — 8, 16, 32. Samplers downstream expect it, and it is easier to snap here than to discover it there.
per_batch	How many frames are worked at once. A long clip does not have to fit in VRAM in one piece.
keep_proportion · crop_position · pad_color	What happens when the target does not share the source's aspect ratio: stretch it, crop it (and from where), or pad it (and in what colour).



Fast Upscale, whole. Four pins in, five out — and the mask comes back scaled by exactly the rule the image was scaled by.

2 · Power Upscale — the thorough one

This one does not just make pixels bigger, it re-samples them. The frame is cut into overlapping tiles; every tile gets a partial denoise at the larger size and is blended back. Detail that was never in the source can appear — which is the point, and also the risk.

Pin	Direction	Carries
image · video	in	What to enlarge.
model	in	The model for stage H.
model_low	in	Optional. The model for stage L — connect it and switch the Mode pill to <code>High + Low</code> .
positive / negative	in	Conditioning for the refine passes.
vae	in	For encoding and decoding around each pass.
upscale_model · upscale_model_low	in	Optional ESRGAN-family models — one per stage , and they may differ.
image · video	out	The finished result.

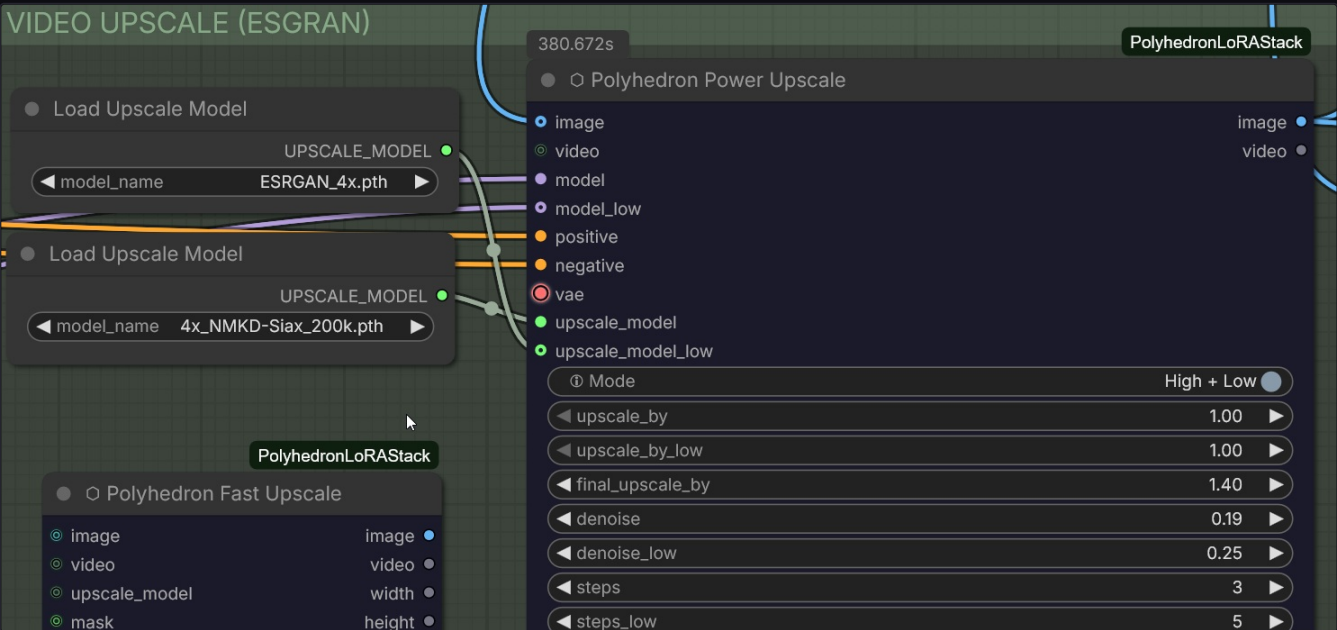
Two stages, not two halves. Stage L does not continue stage H's schedule the way the Sampler's two experts do. It starts from the **finished, decoded, re-upscaled** image of stage H and runs its own fresh partial denoise with its own steps and denoise. There is no seam to keep continuous — which is exactly why `sampler_low` and `scheduler_low` are honoured unconditionally here, while in the Sampler they only bite in one handoff mode.



The head of the node in `High + Low`. Every stage parameter appears twice — the plain name drives stage H, the `_low` twin drives stage L.

2 · Power Upscale — the controls

Control	What it does
Mode	The pill. High + Low turns on the second stage; every _low control below belongs to it.
upscale_by · upscale_by_low	How much each stage enlarges before it refines. Both at 1.00 means the stage refines at the size it was given.
final_upscale_by	A last plain resize after the refining is done. This is where a modest overall enlargement is cheapest — no tile pays for it.
denoise · denoise_low	How far each stage is allowed to reinvent. Low values (0.15–0.30) sharpen and add texture; high values redraw. This is the dial that decides whether the result still shows your picture.
steps · steps_low	Steps per stage. They multiply with the tile count, so a small number here is a large number of tile-steps.
sampler_low · scheduler_low	Stage L's own algorithm and schedule. Default same as high. Ancestral and SDE samplers (*_a , *_sde) inject fresh noise every step — at refine strengths that changes the grain.
tile_size · tile_overlap	The tiling itself. Larger tiles mean fewer seams and more VRAM; the overlap is the band in which neighbours are blended. The blend weights are proved to sum to exactly 1, so no seam is brighter or darker than its surroundings.
pixel_stage	Where the upscale model does its work in the chain.
vae_tiling	Tiles the VAE too, for when decoding is what runs out of memory.
per_batch	Frames per chunk in the pixel stage — the 9 chunks you see in the process view come from this.



One upscale model per stage. Here a general ESRGAN drives stage H and a different model drives stage L — two ordinary core loaders, nothing bespoke.

3 · Power Upscale — watching it work

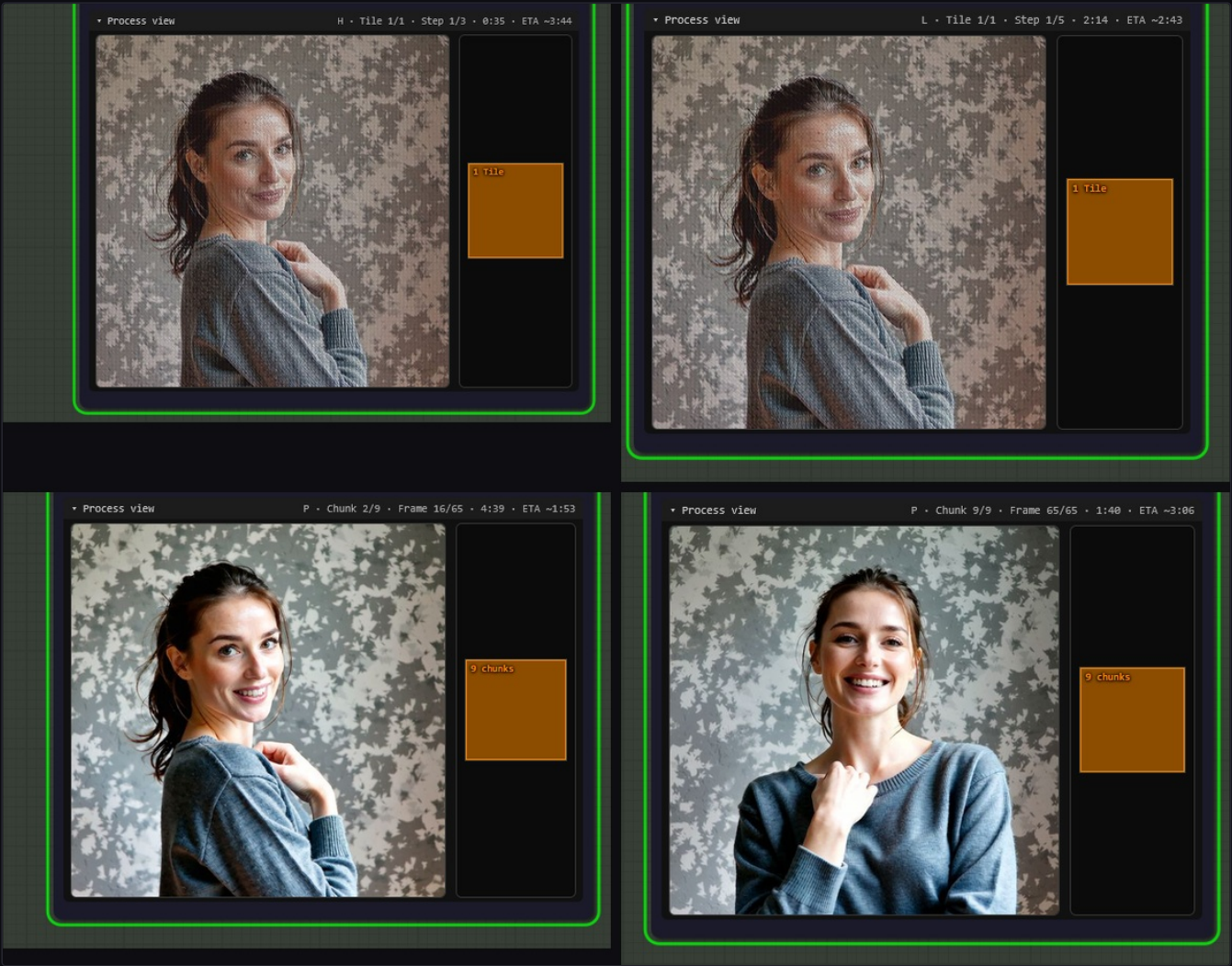
A tiled two-stage upscale of a clip is a long job. The process view exists so that a long job never looks like a hung one: it names the stage, counts the work, and gives a running estimate.

Reading the header line

Letter	Stage	What it counts
H	high	Tile n/m and Step n/m — the first refine, tile by tile.
L	low	The same counters for the second refine, with its own step count.
P	pixel	Chunk n/m and Frame n/m — the pixel stage, working through the clip in chunks.

After the counters comes the **elapsed** time and then the estimate. The estimate is deliberately marked `ETA ~`: it is measured from what has actually run so far, so it moves as the job goes on — and it moves most when the stage changes, because a tile-step and a frame are not the same unit of work.

The panel on the right is the **map**. It shows what the current stage is divided into — `1 Tile` while a stage refines a frame that fits in one tile, `9 chunks` once the pixel stage is working through the clip — and fills in as the work lands.



One run, four moments: stage H at step 1 of 3, stage L at step 1 of 5, then the pixel stage at chunk 2 and at chunk 9 of 9. The estimate falls from ~3:44 to ~0:00 and the map switches from tiles to chunks.

Two previews, two jobs. `process_preview` is what you see *during* the run — it can be a fast approximation or a real VAE decode, and a sharper preview costs real time on every update. `result_preview` is the finished picture that stays in the node afterwards, with a frame slider for clips.

4 · Interpolate — the frames in between

Frame interpolation invents the frames that were never shot. Use it to raise a clip's rate, or to smooth motion at the same rate. It is the cheapest way to make a generated clip look less like a slideshow.

Pin	Direction	Carries
frames	in	The source frames.
video	in	A video object instead of loose frames.
frames	out	The interpolated sequence.
frame_count · fps	out	What actually came out — wire <code>fps</code> straight into a save node so the file is written at the rate it was built for.
interp_info	out	A short text record of what was done.

The controls

Control	What it does
ckpt_name	The RIFE checkpoint. The field shows the file's size beside its name, and a missing checkpoint is fetched on first use.
rate_mode	<code>multiplier</code> doubles or quadruples the frame count; the <code>fps</code> mode works out the ratio from <code>source_fps</code> and <code>target_fps</code> instead.
precision	<code>float32</code> is safe everywhere; half precision is faster where the card supports it.
ensemble	Runs the estimate both ways and averages. Steadier, and slower.
fast_mode	The quicker path through the network.
scale_factor	The resolution the motion estimate runs at. Lower it for very large frames or very fast motion.
static_skip	Leaves near-identical frames alone instead of spending work inventing a frame between two that barely differ.
cut_guard	Stops the interpolator from inventing a smooth transition across a hard cut . Above the threshold the pair is treated as a cut and the frame is duplicated rather than blended. This is the artefact that gives machine interpolation away fastest.



Interpolate at 2x, with both `fps` fields filled in so the mode can be switched without re-entering them.

5 · Choosing between them

You want	Reach for	Why
Bigger, same picture	Fast Upscale	Seconds, not minutes, and nothing is invented. The right answer far more often than it gets used.
Bigger <i>and</i> more detail	Power Upscale	It re-samples. Expect real time, and check that the result is still your picture.
Smoother motion	Interpolate	Frame count, not frame size. It does nothing for sharpness.
A mask that survives	Fast Upscale	The only one of the three that scales a mask with the image by the same rule.

An order that works

Refine first, enlarge last, interpolate at the very end. Power Upscale on the frames you actually want to keep; `final_upscale_by` or a Fast Upscale for the last plain enlargement, because no tile pays for it; and Interpolate last, so the invented frames are made from finished ones rather than being refined and enlarged themselves.

Two habits worth having

Watch the first tile, not the clock. In Power Upscale the first `H` tile tells you almost everything: if the tile already looks wrong at step 1, the remaining fifty tiles will not rescue it. Stop, lower `denoise`, start again.

Let the fps pin do the arithmetic. Interpolate's `fps` output is the rate the sequence was actually built at. Wired into the save node it cannot drift; typed in by hand it eventually will.

On the interpolation engine. The network inside Interpolate is not ours. It is the MIT-licensed IFNet from Practical-RIFE, by way of ComfyUI-Frame-Interpolation, carried verbatim and deliberately unmodified — that way a result can be compared against a reference run and the comparison means something. Everything this pack corrects about frame interpolation lives outside the engine. The attribution travels with the code, in `nodes/vfi/NOTICE.md`.

Part VI — Attention & grading

Polyhedron Suite · User Documentation

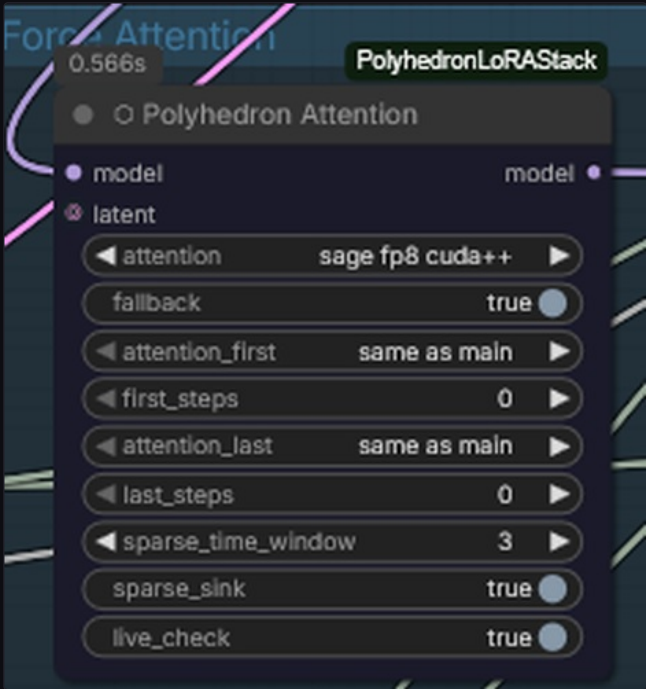
Four nodes that sit **around** the sampler rather than inside it. **Attention** changes the machinery the model computes with — and can change it differently for the opening and closing steps. **NAG** puts guidance back into runs where CFG has been turned down or off. **Filter** grades the picture with a preview that is the grade itself, not an approximation of it. **Audio Stretch** retimes a soundtrack so a slowed clip keeps its sound in step. The first two change how the image is made; the last two change what it looks and sounds like afterwards.

- 1 · Attention — the machinery underneath
- 2 · Attention — choosing a backend
- 3 · NAG — guidance without CFG
- 4 · Filter — grading you can see
- 5 · Audio Stretch — sound that follows the picture

1 · Attention — the machinery underneath

Every diffusion model spends most of its time in attention. Which implementation does that arithmetic is a choice, and it is the single biggest lever on speed that does not touch step count or resolution.

The node takes a model and hands back a patched one. Nothing else in the graph changes — no rewiring, no different sampler. What changes is the kernel the model calls when it computes attention.



The node in a graph. The `latent` input is optional and only the sparse mode needs it.

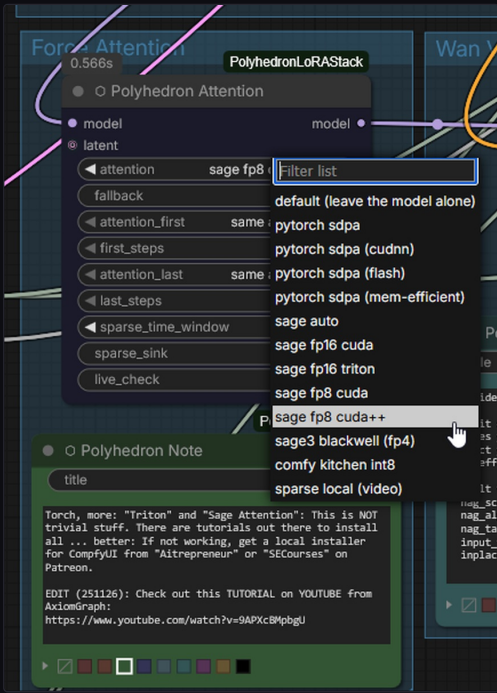
Widget	What it does
<code>attention</code>	The backend for the whole run.
<code>fallback</code>	What to do when a kernel refuses a particular call — a cross-attention shape it does not implement, or a masked call. On, the call goes to the model's own backend and the run continues.
<code>attention_first</code> <code>first_steps</code>	A different backend for the first n steps, where composition is decided. <code>same as main</code> switches the split off.
<code>attention_last</code> <code>last_steps</code>	The same for the closing steps, where fine detail lands.
<code>sparse_time_window</code> <code>sparse_sink</code>	Parameters of the reserved sparse mode (see the note in the next section). They do nothing under any other backend.
<code>live_check</code>	Probe the chosen kernel before the run rather than failing inside it. Costs a fraction of a second and turns a crash 40 minutes in into a message before anything starts.

Why the split exists. The fast quantised kernels are not lossless on every model. Running the exact backend for the first and last few steps and the fast one in between is usually indistinguishable from full precision, and still buys most of the time — because the middle steps are where the bulk of the work sits. If you see artefacts with a quantised backend, this is the dial to reach for before you abandon it.

A call the chosen kernel cannot serve — cross-attention, a masked call, grouped heads, an unfamiliar sequence length — is handed straight back to the model's own implementation. That is not an error and it is not warned about; it is simply not this node's call to make.

2 · Attention — choosing a backend

The list shows what is installed **and loadable on this machine**. A mode whose wheel imports but cannot actually run at scale is not offered, because an option you can pick and then watch fail is worse than one that was never there.



The list on a machine with the SageAttention family installed. On a machine without them, those rows are simply absent.

Mode	What it is
default	Leave the model alone. The honest baseline — use it for any A/B comparison.
pytorch sdpa (+ cudnn / flash / mem-efficient)	Torch's own scaled dot-product attention, either letting torch pick or pinning one implementation. Always available, always exact.
sage auto	SageAttention picks its own kernel for the hardware.
sage fp16 triton sage fp16 cuda	Half precision, two routes to it.
sage fp8 cuda sage fp8 cuda++	Eight-bit. Fastest of the widely available options; the ++ variant refines the accumulation.
sage3 blackwell (fp4)	Four-bit, Blackwell only, and only with the matching wheel.
comfy kitchen int8	ComfyUI's own int8 kernel where the build ships it.
sparse local (video)	Reserved. Offered only when the machinery behind it can be verified; see the note.

On the sparse mode. Cutting attention's temporal reach is the published route to real speed-ups on video. The implementations worth having are also intricate, and a subtly wrong mask does not crash — it quietly degrades the video, which is the worst way for anything to fail. The mode therefore stays parked unless it can be proven correct on the machine it will run on. If you pick it where it is unavailable, the node says so and stops rather than guessing.

3 · NAG — guidance without CFG

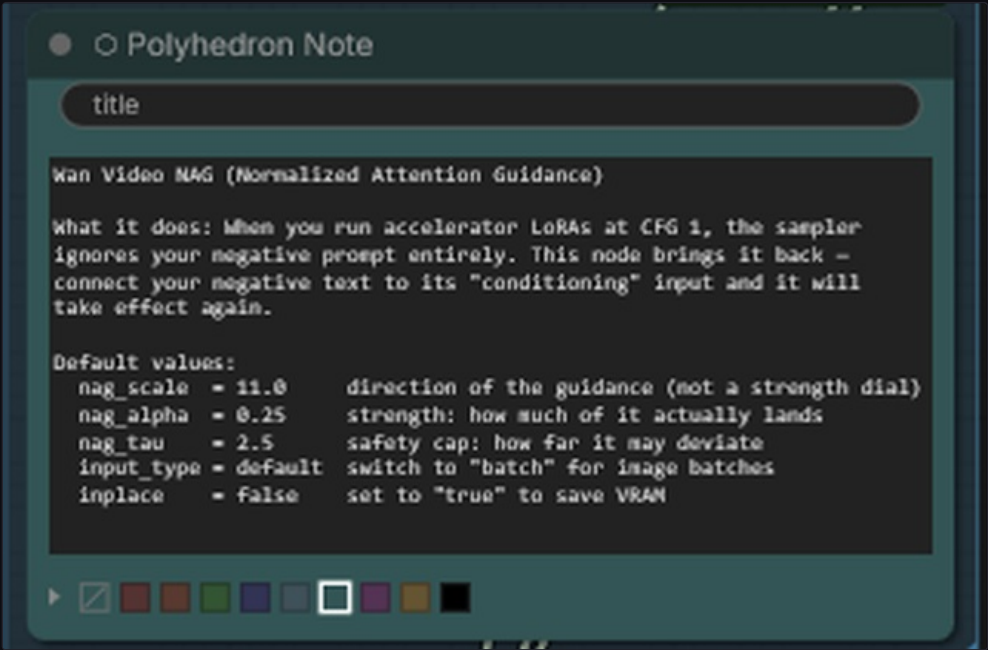
Classifier-free guidance needs two predictions per step, so accelerated and distilled models turn it down or off — and with it goes the negative prompt. Normalized Attention Guidance puts the negative back where CFG cannot: inside attention.



Model in, model out; the negative text goes to `conditioning`.

Widget	What it does
<code>nag_scale</code>	The direction of the guidance rather than a strength dial. Useful values start well above 1.
<code>nag_alpha</code>	How much of the guided result is blended back in — this is the strength.
<code>nag_tau</code>	The safety cap: how far any single token may deviate. It keeps one runaway token from dragging the frame with it.
<code>start_percent</code> <code>end_percent</code>	The slice of the run it applies to.

It sits where any model patch sits: after the LoRAs, before the sampler. Set `nag_scale` to 0 and it does nothing, which makes an A/B on the same seed a one-widget change.



A Polyhedron Note carrying the settings next to the node it documents — notes belong to the graph and cost nothing at run time.

4 · Filter — grading you can see

The preview is not an approximation of the grade. The browser applies the **same** `.cube` file the backend will, read through the pack's own route, so what you set is what you render.



Exposure through sharpening, a LUT with its own strength, and the preset row underneath. The A/B button swaps to the ungraded original.

Group	Controls
Light	exposure , contrast , gamma , highlights , shadows
Colour	temperature , tint , saturation , vibrance , hue_shift
Detail	sharpen_amount , sharpen_radius
LUT	lut_name from the pack's luts/ folder, with lut_strength to blend it in
Presets	Save the current look, load it back, delete it. Files live in the pack's presets/ folder.

The one node here that opens server routes. Three of them, in their own module, because the preview genuinely needs to read the LUT file. Both folders are fixed and derived from the pack's own location, and the caller supplies nothing but a bare filename — nothing outside the pack is reachable. Presets pass through one whitelist in both directions, so a hand-edited or downloaded file cannot push unknown keys into the node.

5 · Audio Stretch — sound that follows the picture

Interpolate a clip to twice the frame rate and the pictures last the same time; slow it down and they do not. The soundtrack has to be told either way.

The node retimes audio to a target it measures rather than one you assert. It shares its machinery with the Interpolate node's `audio_mode` — one implementation reached two ways, so the free-standing node and the built-in path can never drift apart.

Reach it via	When
Interpolate → <code>audio_mode</code>	The ordinary case: the clip is being retimed and its sound should follow. <code>keep</code> passes the original through, <code>stretch</code> retimes it.
Audio Stretch, free-standing	When the audio comes from somewhere else, or when the retime happens outside the interpolation chain.

Why not simply resample. Playing a recording slower lowers its pitch, which is fine for an engine note and ruinous for a voice. The stretch changes the tempo without moving the pitch, so speech survives it.

The number that actually matters is `source_fps` on the Interpolate node. A soundtrack that drifts is almost always a wrong rate there rather than a fault in the retime: interpolation places source frame k at output index $k \times \text{multiplier}$, so at the matching output rate every original second stays where it was. What is left is a tail a fraction of a frame long — milliseconds. Anyone seeing more than that should check the frame rate before anything else.

Appendix · Running on a network

Everything in this manual assumes the normal case: ComfyUI running on your own machine, reachable at `localhost`. If you ever start it with `--listen` so other machines can reach it, one detail of the Media Loader deserves a word.

The loader browses **your file system**. That is the whole point of it: you point it at any folder and it shows you what is inside. The routes behind that browsing take the folder from the client — from the node in your browser — and read it on the machine ComfyUI runs on. On a single machine, the client and that machine are the same thing, so this is simply the feature working.

Bound to a network, they are not the same thing any more. Every client that can reach the server could then ask it to list, read, or upload into arbitrary directories on the host — and could pop a native file dialog on it.

What the pack does about it

When ComfyUI is bound beyond `localhost`, the eleven routes that take a client-supplied path refuse with **403**. That covers browsing folders, listing and reading files, thumbnails and dimensions, uploads, the native file dialogs and the “open in explorer” action. The three routes that work strictly inside the loader’s own managed sequence folder keep working, since they never take a path from outside.

If that is not what you want — a trusted studio LAN, or a machine reachable only through a tunnel you control — set the environment variable `ULS_ALLOW_LAN_BROWSE=1` before starting ComfyUI, and the routes open up again. The refusal message names the variable, so you do not have to remember it.

Nothing changes on a normal setup. On the default localhost bind the loader behaves exactly as described in Part I — the lock is not active, and there is nothing to configure. It is also deliberately fail-open: if the pack cannot determine how the server is bound, it treats the case as local rather than locking a working setup out of its own files.