

VapourWiki — Audion VS Engine preset reference (EN)

Full English reference for all 28 presets across 4 palettes. Starts with a **decision tree** ("which preset for which job"), then per-preset detail: what it does, what material it suits, key parameters, recommended encoder.

Russian counterpart: [VapourWiki_RU.md](#) (this document mirrors it). Per-preset technical docstrings live inside each `.vpy` file in `system_core/presets/<palette>/`.

Decision tree — which preset for which job

Read top-down: the first matching case is your preset.

Material symptom	Preset	Palette
Interlaced legacy (DV, HDV, VHS captures, broadcast TS)	<code>qtgmc_deinterlace</code>	restoration
NTSC 29.97 fps with 3:2 pulldown (telecined film)	<code>tivtc_ivtc</code>	restoration
High-ISO / night noise, must keep detail	<code>mvtools_mcdegrain</code>	restoration
Rainbow / dot crawl on VHS-rip or composite capture	<code>derainbow_decross</code>	restoration
Over-compressed H.264/MPEG (YouTube-rip, WhatsApp, SD broadcast)	<code>deblock_h264_artefacts</code>	restoration
Haze / flat low-contrast — need to "lift" the picture	<code>dehaze_local_contrast</code>	restoration
ML upscale 1080p → ~4K (sharp faces, fabric texture)	<code>vsmlrt_realesrgan_2x</code>	restoration
ML interpolation 24 → 60fps (smoother than Optical Flow)	<code>vsmlrt_rife_60fps</code>	restoration

Material symptom	Preset	Palette
ML denoise of heavy noise / high-ISO without texture loss	<code>dpir_denoise</code>	restoration
Digital noise in shadows only ("night" digital footage)	<code>shadow_denoise_sota</code> ★	precision
Light uniform noise, hardware-agnostic (no CUDA/OpenCL)	<code>mild_denoise</code>	precision
Chroma noise only (dirty blue channel)	<code>chroma_cleanup</code>	precision
Banding in gradients (skies, walls)	<code>deband_safe</code> or <code>deband_fine_grain</code>	precision
One preset for everything filmic	<code>filmic_rebuild</code> ★	precision
Prep for color grading in DaVinci	<code>pregrade_prep</code>	precision
Clean archival master without grain	<code>archive_clean</code>	precision
Subtle filmic look, safe default	<code>cinematic</code>	film_looks
Specific 35mm Kodak (250D / 500T / 50D)	<code>film_35mm</code>	film_looks
16mm organic, lifted shadows	<code>film_16mm</code>	film_looks
Super 8, faded 70s, heavy grain	<code>super8</code>	film_looks
High-contrast desaturated (Se7en, Saving Private Ryan)	<code>bleach_bypass</code>	film_looks
Large-format IMAX feel — minimal grain, gate weave	<code>imax_70mm</code>	film_looks
Hollywood cinemascope (horizontal blue lens flares + teal shadows)	<code>anamorphic_scope</code>	film_looks
VHS / CRT aesthetic	<code>vhs_crt</code>	retro
90s amateur camcorder	<code>camcorder_90s</code>	retro
1970s Polaroid SX-70 (candy-bloom, warm cream)	<code>polaroid</code>	retro

Pipeline scenarios (chain multiple presets via `apply-profile-batch` or manual runs):

Scenario	Steps
Night timelapse → film	<code>shadow_denoise_sota</code> → <code>filmic_rebuild</code> → <code>cinematic</code>
VHS archive → restored master	<code>qtgmc_deinterlace</code> → <code>derainbow_decross</code> → <code>mvtools_mcdegrain</code> → <code>archive_clean</code>
Old YouTube-rip → project-ready	<code>deblock_h264_artefacts</code> → <code>dehaze_local_contrast</code> → <code>pregrade_prep</code>
Telecined NTSC DVD → 24p master	<code>tivtc_ivtc</code> → <code>archive_clean</code>
Modern digital → film look	<code>mild_denoise</code> → <code>film_35mm</code>

PRECISION palette — technical pipeline (8 presets)

Menu order = pipeline logic: Stage 1 (denoise) → Stage 2 (deband) → Stage 3 (compositions).

Stage 1 — denoise

`mild_denoise` — soft universal denoiser

DFTTest spectral denoiser. Pure CPU, hardware-agnostic — same behaviour on any CPU.

Recommended as the **first choice** when nothing specific is known about the material: light noise, gentle clean-up without risk.

- Parameters: `strength` = light (sigma=4.0) / medium (8.0) / strong (14.0)
- Encoder: `h264_crf17` or `h264_crf14` for archival quality
- What's in NLE: only Temporal NR / generic Noise Reduction; coarser results

`shadow_denoise_sota` ★ — flagship shadow denoiser

BM3D with float32 pipeline + smooth luma-mask "shadows only". Noise is killed **only** in zones below `shadow_threshold`; the rest of the picture is untouched. Chroma planes are always denoised (chroma noise is equally ugly everywhere). Optional `grain_back` re-adds micro-grain so cleaned shadows do not look "plastic".

- Parameters: `sigma=2.5` (BM3D luma), `use_cuda=0/1`, `grain_back=0.6`, `shadow_threshold=0.20`, `transition=0.10`
- On NVIDIA: `--use-cuda 1` gives 25–35% wall-time win on 4K

- Encoder: `h264_crf17` / `prores_lt`
- Not in NLE: zone-targeted denoise with smooth blend, no cheap "threshold mask" hack

`chroma_cleanup` — chroma-only cleanup

DFTTest on planes=[1,2]. Luma untouched. Use when "the blue channel is dirty" (typical on old compact cameras, low-bitrate AVCHD).

- Parameters: `strength` = light / medium / strong
- Encoder: `h264_crf17`
- Not in NLE: clean chroma-only DFTTest without luma side-effects

Stage 2 — deband

`deband_safe` — safe debander

neo_f3kdb with light settings, no grain. Removes banding in gradients (skies, walls, night lighting) without softening edges.

- Parameters: `range=15`, `y=64`, `cb=64`, `cr=64`
- Encoder: `h264_crf17` / `h265_crf21`

`deband_fine_grain` — deband + fine-grain restore

Same neo_f3kdb + AddGrain. Light monochromatic grain after debanding — avoids the "too clean" look characteristic of cheap compression.

- Parameters: `range=15`, `grain_var=1.5`
- Encoder: `prores_lt` (for downstream grading)

Stage 3 — compositions

`filmic_rebuild` ★ — flagship composition

Full chain: BM3D denoise → neo_f3kdb deband → 3-zone luma grain (shadows / midtones / highlights). Zonal grain — heavier in shadows (like real film), lighter in highlights. Recommended as "one preset for everything filmic".

- Parameters: `sigma=2.0`, `use_cuda=0/1`, `deband_range=14`, `grain_shadow=1.5`, `grain_mid=0.9`, `grain_high=0.4`, `shadow_threshold=0.30`, `high_threshold=0.65`
- Encoder: `prores_lt` / `h265_crf21`

archive_clean — neutral archival master

DFTTest + neo_f3kdb. **No grain.** Goal — maximally clean, "flat" picture for archiving. Do not use if downstream grading is planned (grading works better on grainy material).

- Parameters: `denoise=medium`, `range=15`
- Encoder: `prores_422hq` / `prores_422hq_mxf`

pregrade_prep — handoff to DaVinci

Minimum touch: light DFTTest only. No deband, no grain. Goal — give Resolve the cleanest source so the colorist isn't grading on top of compression artefacts.

- Parameters: `strength=light`
- Encoder: `prores_lt_mxf` (round-trip with Adobe / Avid)

FILM_LOOKS palette — film emulations (7 presets)

Each look uses the shared `audion_lib` library (MTF softening, halation bloom, zone grain, gamma curve, black-lift, desaturation).

cinematic — universal subtle filmic

Safe default. Light MTF softening + halation + micro-grain. Not tied to a specific stock. Works on any modern digital material.

- Parameters: `intensity=1.0` (range 0..2)
- Encoder: `prores_lt` / `h264_crf17`

film_35mm — Kodak 35mm with stock variants

Emulates real Kodak stocks: 250D (daylight balanced), 500T (tungsten), 50D (fine grain). Each stock has its own gamma curve, balance, grain density.

- Parameters: `stock=250D|500T|50D`, `intensity=1.0`
- Encoder: `prores_lt` / `prores_lt_mxf`

film_16mm — organic, grainier

Lifted blacks, more pronounced grain, softer than 35mm. Suits "documentary" / arthouse aesthetics.

- Parameters: `intensity=1.0`

- Encoder: `prores_lt`

`super8` — strongest look

Heaviest grain, softest optics, faded 70s. For music videos / stylized inserts.

- Parameters: `intensity=1.0`
- Encoder: `h264_crf17` (grain is already baked in, aggressive compression OK)

`imax_70mm` — large-format IMAX feel ★

Opposite of Super 8: **minimal grain, maximum resolution**, very soft halation on highlights, optional 1px gate weave (deterministic, seed=42 → reproducible). The huge frame of real 70mm IMAX → each silver grain is tiny relative to the picture. Suits shots that should feel "epic", not "filmic".

- Parameters: `intensity=1.0`, `gate_weave=1` (on by default)
- Encoder: `prores_lt` / `h265_crf17`
- Not in NLE: calibrated micro-grain plus gate weave (NLEs only do flat plate grain)

`anamorphic_scope` ★ — Hollywood cinemascope

Anamorphic optics signature: **long horizontal blue lens flares** on highlights only (those "horizontal blue streaks across the sky" in Abrams / Nolan / Villeneuve films, made by Panavision / Hawk / ARRI Master Anamorphic). Plus subtle teal-cool shadows for the classic blockbuster look.

In our implementation: highlight-mask (`flare_thr`) → horizontal low-pass blur of length `flare_len` → blue tint (R 10% / G 55% / B 100%) → additive merge over the original. Then shadow-masked teal shift (R -10% / G +5% / B +12%). All in 16-bit RGB so streaks don't clip.

- Parameters: `intensity=1.0`, `flare_len=96` (48 subtle / **96 default** / 160 dramatic / 256 extreme), `flare_thr=0.78` (0.85 highlights-only / **0.78 default** / 0.70 aggressive / 0.60 heavy), `teal_shadow=0.35` (0.0 off / 0.20 subtle / **0.35 default** / 0.60 heavy)
- Encoder: `prores_lt` (for colorist) or `h265_crf14` (final)
- The preset does **not** crop the frame to 2.39:1 — that's a creative choice; add `ffmpeg -vf crop=W:floor(W/2.39):0:Y` in post if you need true scope.
- Not in NLE: directional anamorphic streak without expensive paid plugins like Optical Flares; smooth threshold for flare cut-in; full 16-bit math.

bleach_bypass — high-contrast desaturated

"Se7en" / "Saving Private Ryan" look. High contrast, silver greys, blown highlights. Hard stylistic choice.

- Parameters: `intensity=1.0`
 - Encoder: `prores_lt`
-

RETRO palette — analog character (3 presets)

vhs_crt — VHS / CRT

Chroma bleed (horizontal color smear), soft optics, analog noise. Emulation of VHS playback through a CRT TV.

- Parameters: `intensity=1.2`
- Encoder: `h264_crf17`

camcorder_90s — amateur camcorder

Softer than VHS, slight overexposure, minimal chroma bleed. 90s home video aesthetic.

- Parameters: `intensity=1.0`
- Encoder: `h264_crf17`

polaroid ★ — 1970s Polaroid SX-70

Strong candy-bloom on highlights, lifted blacks (never goes to true black, creamy fade), warm gamma push, mild desaturation, **radial edge vignette**. Five composited steps — what would take a stack of 5–6 effects to build manually in NLE.

- Parameters: `intensity=1.0`, `vignette=0.55` (range 0..1; 0 = none, 1 = corners to black)
 - Encoder: `h264_crf17` (filmic material tolerates compression)
 - Not in NLE: one-shot Polaroid emulation; radial vignette without geometric distortion or LUT side-effects
-

RESTORATION palette — supercannons (10 presets, Phase 18.B + 18.B-ML)

What's **missing or weak** in Adobe Premiere / DaVinci Resolve.

qtgmc_deinterlace — reference-grade deinterlace

havsfunc.QTGMC (NNEDI3 + MVTools motion estimation). Reconstructs progressive frames from interlaced source via neural-network upscaling of each field + motion-compensated temporal smoothing. **Better than any NLE out-of-the-box.**

- Parameters: `field_order=tff|bff`, `qtgmc-`
`preset=Faster|Fast|Medium|Slow|Slower|Placebo`, `output_fps=single|double`
- When to use: DV, HDV, VHS captures, broadcast TS, S-VHS / U-matic digitization
- Encoder: `prores_lt` (for downstream work) or `h264_crf17` (if final master)

tivtc_ivtc — inverse telecine

TIVTC.TFM (field matching) + TIVTC.TDecimate (drop duplicate). Reference-quality 3:2 pulldown removal: NTSC 29.97i → 23.976p. Recovers the original 24p film master from a telecined source.

- Parameters: `pp=6` (TFM post-processor), `cycle=5`, `rdrop=1` (standard NTSC pattern)
- When to use: NTSC DVDs, broadcast prints, digitized film transfers
- Encoder: `prores_422` / `prores_422hq_mxf`

mvtools_mcdegrain — motion-compensated denoise

MVTools motion estimation → MDeGrain temporal averaging along motion vectors. **Removes noise WITHOUT losing detail** — what Topaz Video Enhance AI and Neat Video do internally. DaVinci Temporal NR is a coarse implementation of the same; here you get reference-grade.

- Parameters: `radius=2` (5-frame window), `thsad=200`, `blksize=16` (HD) / `8` (4K detail)
- When to use: high-ISO night footage, preserving skin / fabric texture
- Encoder: `prores_lt` / `h265_crf17`

derainbow_decross — NTSC composite cleanup

MVTools-based motion-compensated chroma-only smoothing. Removes rainbow / dot crawl / cross-color on composite-capture material (VHS-rip, U-matic, BetaSP, S-Video → SDI). Luma untouched.

- Parameters: `strength=0.6`, `blksize=16`

- When to use: digitized VHS, color "crawls" on thin lines, mosquito noise on B/W edges
- Encoder: `prores_lt` / `h264_crf17`

`deblock_h264_artefacts` — over-compression rescue

`havsfunc.Deblock_QED` — edge-aware deblocker for H.264 / MPEG-2 / MPEG-4. Smooths 8x8 block boundaries, ringing around edges, mosquito noise. Edge-aware — does not smear real detail.

- Parameters: `quant1=24`, `quant2=26`, `aoffset=1`, `boffset=1`
- When to use: old YouTube rips, WhatsApp/Telegram re-encodes, low-bitrate SD broadcast TS
- Encoder: `h264_crf17` (after deblock the material is cleaner; you can re-encode at higher CRF)

`dehaze_local_contrast` — clarity without halos

Luma-only local contrast via high-pass + zone-weighted MaskedMerge. Zone-mask (parabolic, peak at midtones) prevents blown highlights and crushed shadows. Color does not shift (luma-only). 16-bit math.

- Parameters: `strength=1.0`, `radius=8` (clarity) / `12-16` (haze removal)
- When to use: haze, flat low-contrast material, need to "wake the picture up" without destroying highlights
- Not in NLE: Resolve "Dehaze" blows highlights and shifts color; here, no halos
- Encoder: `prores_lt` (handoff to colorist) or `h264_crf17` (final)

Encoders — my patterns

(Audion default workflow per the 2026-04-28 discussion)

Quality ladder (same 14 / 17 / 21 tiers across software, NVENC, QuickSync, and AMF): **14 → 17 → 21**, three distinguishable steps without overlap.

Profile	When to use
<code>h264_crf14</code> / <code>h265_crf14</code> ★ default	Semi-lossless archive, master for downstream work. Audion default since 2026-04-28.
<code>h264_crf17</code> / <code>h265_crf17</code>	"Almost invisible" lossy, final master
<code>h264_crf21</code> / <code>h265_crf21</code>	Web preview / proxy / preview

Profile	When to use
<code>prores_lt</code> ★	Audion default ProRes — no keying, made for grading and round-trip
<code>prores_lt_mxf</code> ★	ProRes LT in MXF wrapper — for Adobe Premiere / Avid round-trip
<code>prores_422</code>	If you need slightly more bitrate than LT
<code>prores_422_mxf</code>	ProRes 422 in MXF wrapper for Adobe / Avid handoff
<code>prores_422hq</code>	Only if keying is planned
<code>prores_422hq_mxf</code>	HQ + MXF for broadcast / Avid finishing
<code>dnxhr_lb</code> / <code>_sq</code> / <code>_hq</code> / <code>_hqx</code>	Avid-style DNxHR (LB low / SQ standard / HQ high / HQX 10-bit)
<code>h264_nvenc_q14</code> / <code>_q17</code> / <code>_q21</code>	NVENC hardware H.264 on NVIDIA. Removes the libx264 CPU bottleneck — VS filters don't compete with encode. CQ ≈ CRF . 8-bit yuv420p.
<code>h265_nvenc_q14</code> / <code>_q17</code> / <code>_q21</code>	NVENC hardware HEVC, 10-bit p010le. Ideal for full pipeline on NVIDIA: VS filtering on CPU/CUDA + encode on NVENC = no CPU bottleneck.
<code>h264_qsv_q14</code> / <code>_q17</code> / <code>_q21</code>	Intel QuickSync H.264. Good for Intel iGPU batch/proxy work when CPU should stay free for VapourSynth.
<code>h265_qsv_q14</code> / <code>_q17</code> / <code>_q21</code>	Intel QuickSync HEVC, p010le for 10-bit output where supported.
<code>h264_amf_q14</code> / <code>_q17</code> / <code>_q21</code>	AMD AMF H.264 via CQP, matching the same visual ladder.
<code>h265_amf_q14</code> / <code>_q17</code> / <code>_q21</code>	AMD AMF HEVC, p010le output; useful on Radeon hosts.

Rule: for any non-final pass → `prores_lt` / `prores_lt_mxf` or DNxHR if the receiving app prefers it. For final delivery → `h264_crf14` / `h264_crf17` (software, best density per bit) OR the matching hardware ladder for the host (`nvenc`, `qsv`, `amf`) when wall-time matters.

When to use hardware encode vs software

Scenario	Encoder
Heavy VS pipeline (filmic_rebuild, mvtools_mcdegrain) on NVIDIA	NVENC — CPU is freed for the VS filter, total wall-time drops 30-50%
Final delivery where maximum bit density matters	libx264/libx265 CRF — software encoder is still slightly more efficient at low CRF
Intel iGPU host / laptop batch	QuickSync — good throughput with low CPU pressure
AMD/Radeon host	AMF — same 14/17/21 CQP ladder, avoids CPU encode bottlenecks
Big batch over hundreds of files	NVENC / QSV / AMF — time savings compound
No working hardware encoder	Software only (<code>h264_crf*</code> / <code>h265_crf*</code>)

NVENC requires: NVIDIA driver R525+. Quality: NVENC on Ada/Blackwell (RTX 40/50) is **comparable** to libx264 medium at the same CQ; on older generations (Pascal/Turing) software CRF wins slightly on bitrate efficiency for the same quality, but NVENC is still many times faster.

Install & service scripts — when to run what

There are many scripts in `install/`, and not all are obvious. This section is the "when do you call which" map with concrete arguments and examples. Every script comes as a `.cmd` (thin wrapper, resolves portable PowerShell) + `.ps1` (full logic) pair. Usually you launch them via `builder_main.cmd` (FZF menu), but any can be run directly via double-click or CLI.

Entry point: `builder_main.cmd`

Main menu of service operations. FZF navigation (if `system_core\fzf.exe` exists), otherwise CMD fallback with letter hotkeys. Menu map:

#	Item	What it does	Underlying script
<code>[01]</code>	Build portable env CMD builder	initial build of <code>runtime/</code>	<code>Build_Portable_Env_Build.cmd</code>

#	Item	What it does	Underlying script
[02]	Build portable env PS	alt PowerShell-based builder	<code>Build_Portable_Env.ps1</code>
[03]	Install portable offline	offline install from a pre-downloaded wheelhouse	<code>install_portable_offline.cmd</code>
[04]	Verify portable env	orchestrator-Python sanity	<code>verify_portable_env.cmd</code>
[05]	Update FZF	bump <code>fzf.exe</code> to latest	<code>launcher-tools-update_fzf.cmd</code>
[06..08]	Licenses	collect / prune / dedupe license files	<code>system_core\license\Run-*.cmd</code>
[09]	Make release archive	release zip excluding <code>output/</code> , <code>logs/</code> , <code>._runtime/</code> , <code>install/download/</code> , private <code>MEMORY.md</code>	<code>make_release_archive.cmd</code>
[04]	PowerShell	portable pwsh 7 → <code>system_core\powershell\</code>	<code>Install-Portable-PowerShell.cmd</code>
[10]	VapourSynth	latest VS stable + latest Python 3.12.x embed → <code>system_core\vapoursynth\</code>	<code>Install-Portable-VapourSynth.cmd</code>
[11]	VS plugins	vsrepo plugin set (incl. <code>bm3dcuda</code> and <code>znedi3</code> by default)	<code>Install-VS-Plugins.cmd</code>
[12]	FFmpeg	Exact-stable Gyan by driver policy; BtbN rolling opt-in → <code>Tools\ffmpeg\bin\</code>	<code>Install-Portable-FFmpeg.cmd</code>
[13] ★	VS-MLRT LEAN	fresh-download vs-mlrt TensorRT bundle, clean <code>plugins\vsmlrt</code> , lean-trim to current SM	<code>Install-VS-mlrt.cmd /LEAN</code>
[14]	VS-MLRT FULL	fresh-download same install, no trim — for USB distribution	<code>Install-VS-mlrt.cmd /FULL</code>

#	Item	What it does	Underlying script
[70]	Clean install cache	remove transient install downloads, staging dirs, and bytecode caches while preserving portable payloads	Clean-Install-Cache.cmd
[90..99]	Open / Project	explorer for subfolders / jump to project launcher	—
[00]	Exit	—	—

From-scratch order: [01] -> [03] -> [04] -> [10] -> [11] -> [12] -> (optional) [13] -> [70]. After all install steps, run `runtime\python.exe system_core\doctor.py` once — it should be all green.

`install\Install-Portable-VapourSynth.{cmd,ps1}` — latest VS stable + own Python 3.12.x

Resolves the latest stable VapourSynth release and latest Python 3.12.x embed, cleans `system_core\vapoursynth\`, installs the matching VapourSynth wheel, and drops the `portable.vs` marker so vsrepo enters portable mode. After the wheel install it explicitly creates and prints the active plugin dir from `vapoursynth.get_plugin_dir()`; VS R74+ no longer treats legacy `vs-plugins\` as the real autoload target. Uses `Expand-7zArchive` (via `Ensure-7zip.ps1`) — ~3× faster than `Expand-Archive` on large archives.

At the end calls `Repair-PipShims.cmd` — fixes shebangs in `Scripts\*.exe` immediately after wheel install (a fresh install needs no manual fix).

Flags: `/R <rev>` (e.g. `/R R76` to pin version). `/F` is accepted for compatibility; current installer already refreshes by default.

`install\Install-VS-Plugins.{cmd,ps1}` — VapourSynth plugins via vsrepo

Installs:

- v1.0 set: `lsmas`, `ffms2`, `fmtconv`, `neo_f3kdb`, `addgrain`, `knlmeanscl`, `bm3dcpu`, `dfttest`
- Restoration set (Phase 18.B): `havsfunc`, `mvsfunc`, `mvtools`, `tivtc`
- CUDA: `bm3dcuda` **by default** (Phase 18.A0). Opt-out via `/NO-CUDA` (on a clean non-NVIDIA host, to skip a plugin that would fail at load anyway).

Also installs `vsutil` via pip (imported at the top level of `havsfunc.py` — without it, Restoration presets crash with `ModuleNotFoundError`).

Update-safe: `vsrepo update` refreshes the package database, the installer cleans the active `vapoursynth.get_plugin_dir()` path while preserving `plugins\vsmlrt` if present, then `vsrepo install` installs the requested list again. Safe to re-run after updating the plugin list.

`install\Install-Portable-FFmpeg.{cmd,ps1}` — BtbN GPL latest, Gyan.dev fallback

Installs a driver-compatible exact-stable Gyan build by default. A BtbN rolling release-branch build remains available only by explicit opt-in. Drops `ffmpeg.exe` / `ffprobe.exe` / `ffplay.exe` into `Tools\ffmpeg\bin\`. Fast extraction via `Expand-7zArchive`.

Flags: `/V <variant>` (`gpl` default / `lgpl` / `gpl-shared`), `/F` (force).

`install\Install-VS-mlrt.{cmd,ps1}` — ML stack (Phase 18.B-ML)

Only for those who want the ML presets (`vsmlrt_realesrgan_2x`, `vsmlrt_rife_60fps`, `dpir_denoise`). Installs the full TensorRT bundle: VSTRT + VSORT + VSOV + VSNCNN + ONNX runtime + TensorRT runtime DLLs + the complete ONNX model collection. After extraction, Audion moves OpenVINO/vsov out of the active autoload folder to `system_core\vapoursynth\disabled_plugins\vsmlrt-openvino`: TensorRT / TensorRT-RTX / ONNX Runtime remain active, while Windows `Bad Image` `0xc0e90002` from OpenVINO DLLs cannot break startup.

Working set after install: ~3.5 GB at `/LEAN` (default), ~10 GB at `/FULL`. Every run fresh-downloads the selected vs-mlrt assets and cleans only the `plugins\vsmlrt` subtree before copying runtime DLLs, scripts and models. Extraction via `7zr.exe` (needed for BCJ2-filtered streams in TRT runtime DLLs — `py7zr` cannot decode them).

Full update order: if you run `[10] VAPOURSYNTH` or `[11] VS PLUGINS`, run `[13] VS-MLRT LEAN` again before any Full all-presets smoke. ML presets should never be validated against an older MLRT layer after the base VS runtime/plugins were refreshed.

Flags:

- `/LEAN` (default via `[13]`) — trims TensorRT builder resources to the current SM (detected via `nvidia-smi --query-gpu=compute_cap`), removes unused models (cugan, waifu2x). On non-NVIDIA, falls back to FULL behaviour.
- `/FULL` (via `[14]`) — full bundle for USB distribution to another machine.

- `/DROP-CACHE` — after success, calls the central `Clean-Install-Cache` policy for transient install downloads, staging dirs, and bytecode caches.
- Default from builder = `/LEAN` and keeps install cache. Use `[70] Clean install cache` or `/DROP-CACHE` explicitly to remove cached downloads.

After install, the backend in ML presets is selected **automatically**: TRT (if NVIDIA + matching TRT runtime) → ORT_DML (DirectML, any DX12 GPU including Intel Xe / AMF) → ORT_CPU. Cross-vendor stable (via `audion_lib.vsmmlrt_backend_chain()`) — on non-NVIDIA we don't even attempt to compile a TRT engine, saving 30–120 s).

Annoyance: Windows Defender blocks the unsigned `openvino_intel_npu_plugin.dll`. The script removes it automatically after extract (not needed on NVIDIA / DirectML setups).

`install\Clean-Install-Cache.{cmd,ps1}` — disk reclaim

Removes transient install downloads (`.7z`, `.zip`, `.tar.*`, `.msi`, `.exe`), exact installer staging dirs, and Python bytecode caches outside payload/user-data zones. Preserve list inside

`install\download\` (always kept):

- `.gitkeep`
- `get-pip.py` (re-used on every fresh build)
- `7z*-extra.7z` (Ensure-7zip bootstrap helper, ~2 MB)

When you need it: after a successful `[13] VS-MLRT LEAN` or `[14] VS-MLRT FULL` (~3.5 GB compressed archives), periodically after `Install-Portable-*` (fresh FFmpeg build / new VS R75+).

Philosophy: **storage > bandwidth**. Re-download = ~1.5 minutes on gigabit; 3.5 GB on disk sit there forever. Working set after lean+cleanup: ~3.5 GB instead of 10.

`install\Repair-PipShims.{cmd,ps1}` — fix `Scripts\*.exe` after a move

Symptom: after moving the project to a different drive letter / path / machine, `vspipe.exe` and other `Scripts\*.exe` silently exit 1 with no output. `doctor.py` shows `[FAIL] vspipe runs` while `python.exe -c "import vapoursynth"` works fine.

Cause: pip embeds an absolute shebang (`#!"<full python.exe path>"`) inside each `Scripts\*.exe`. After the move, that path no longer exists.

What to do: run `install\Repair-PipShims.cmd`. It rewrites the shebang in every `Scripts\*.exe` to the current `system_core\vapoursynth\python.exe`. The launcher stub and trailing zip are left

intact.

Flags:

- `/WHATIF` or `/N` — dry-run preview without writing.

In normal operation **not needed**: `system_core\engine\selfheal.py` is hooked into the start of `main.py` and `doctor.py` — it reads the first 16 KB of `vspipe.exe` on startup, parses the embedded shebang, and on mismatch silently calls `Repair-PipShims.cmd`. Idempotent via `AUDION_SELFHEAL_DONE=1`. Manual repair is needed only if you hit vspipe outside `main.py` / `doctor.py` (e.g. a direct CLI bench script that doesn't go through self-heal).

`install\Bench-CUDA.{cmd,ps1}` — pure-pipeline CPU vs CUDA on BM3D

What it benches: two BM3D-heavy presets (`shadow_denoise_sota` and `filmic_rebuild`), each run twice — on CPU and on CUDA. Pipeline `vspipe → ffmpeg -f null` (no encoding). No-encode is needed so libx264 doesn't eat all wall-time on a fast multi-core CPU and hide the CUDA win.

How to use:

```
:: Drag-n-drop a file onto Bench-CUDA.cmd, or:
install\Bench-CUDA.cmd "C:\clips\test.mov"
install\Bench-CUDA.cmd "C:\clips\test.mov" 2.5    :: second arg = sigma (default 2.5)
```

Output: 4 timing lines + a summary table CPU/CUDA + Δ %. Nothing is written to disk.

How to read it:

- On the reference DCI 4K ProRes 25-second clip (RTX 5070 / Ryzen 9 5900X):
`shadow_denoise_sota` = -35%, `filmic_rebuild` = -24%.
- On clips shorter than ~5 s the CUDA setup cost is not amortized → result can be noise or even slightly slower than CPU.
- **Task Manager lies**: even on a working CUDA path it often shows 5–10% GPU. BM3D is bandwidth-bound and runs in bursts; surrounding `fmtc` bit-depth conversions and frame-prop tags happen on the CPU. The authoritative "CUDA is alive" signal is the wall-time delta and `[OK] bm3dcuda runtime - live BM3D CUDA invocation succeeded` in `doctor.py`.

`install\Bench-AllPresets.{cmd,ps1}` ★ — smoke across all 28 presets

Walker over `system_core\presets\<palette>\*.vpy`. Every preset is run through `vspipe -c y4m --end <Frames-1> | ffmpeg -f null` — pure pipeline, no disk writes. Smoke goal: confirm that (a) the preset parses, (b) every plugin namespace it needs resolves, (c) frames make it to ffmpeg without exceptions.

How to use:

```
:: Drag-n-drop, or:
install\Bench-AllPresets.cmd "C:\clips\test.mp4"
install\Bench-AllPresets.cmd "C:\clips\test.mp4" 30 sweep auto
```

Current local references:

- Portable CPU/ORT fallback: `28/28 PASS` with `Frames=1`, `Cuda=off`, `MLBackend=ort_cpu` (2026-05-16).
- RTX 5070 validation: `36/36 PASS` with `Frames=1`, `Cuda=sweep`, `MLBackend=auto` (2026-05-27). The 36 rows are 28 presets plus the extra CPU/CUDA Precision sweep.

Arguments:

- Video path (required)
- Frames per preset (default 30)
- Cuda mode: `off` / `on` / `sweep` (default off)
 - `off` — every preset on CPU
 - `on` — every Precision preset with `AUDION_VS_USE_CUDA=1`
 - `sweep` — Precision runs twice (CPU and CUDA), other palettes once
- ML backend: `auto` / `trt` / `ort_dml` / `ort_cpu` / `sweep` (default auto)
 - `auto` — vs-mlrt picks TRT → ORT_DML → ORT_CPU itself
 - `sweep` — ML presets run twice (auto and `ort_cpu`) for GPU vs CPU baseline comparison

How many rows you get:

- `off auto` — one row per preset (28 total)
- `sweep auto` — Precision×2 + others×1 = 8×2 + 7 + 3 + 10 = 36 rows
- `sweep sweep` — Precision×2 + ML restoration×2 + others×1 = 16 + 7 + 3 + 6 + 4×2 = 40 rows

At the end — Summary `Total / PASS / FAIL`; for `-Cuda sweep` a `CPU vs CUDA Δ %` table over Precision presets, plus a JSON report at `logs\bench_all_presets_<TS>.json`.

When to run it:

- After `Install-VS-Plugins` or `Install-VS-mlrt` — confirm every preset is alive (`PASS == Total`).
- After moving the project (post-`Repair-PipShims`).
- After an NVIDIA driver upgrade — to confirm TRT engines rebuild without errors.
- When adding a new preset — make sure it didn't break neighbour imports.

Hidden-but-important details:

- Per-preset env overrides (`$presetOverrides` table in `Bench-AllPresets.ps1`) — without them `AUDION_VS_STRENGTH=medium` (a Precision tag) breaks Restoration presets (`dehaze`, `derainbow`) that parse STRENGTH as float. And `AUDION_VS_MODEL` differs across ESRGAN / RIFE / DPIR.
- Separate stderr temp files for vspipe and ffmpeg (a single cmd.exe pipe cannot `2>file.log` from both processes — `ERROR_SHARING_VIOLATION`).
- Local row vars are **not** named `$cuda` — that would clobber the script param `$Cuda` (PowerShell variables are case-insensitive → `ValidateSet` breaks on assignment).

`install\Ensure-7zip.ps1` — dot-source helper for 7-Zip

Not run directly — other `*.ps1` files dot-source it:

```
. "$PSScriptRoot\Ensure-7zip.ps1"
$exe = Ensure-7zr -ProjectRoot $ProjectRoot
Expand-7zArchive -Archive $zip -Destination $dst -ProjectRoot $ProjectRoot
```

Build env steps `[01]/[02]` explicitly install `7zr.exe` (~1.5 MB, pure 7z extractor with BCJ2 support) and `7za.exe` (~1.7 MB, universal — zip / 7z / tar / gz / bz2 / xz) into `system_core\7zip\` before Python setup. Fetched once, then travels with the project.

Why we need it:

- `Expand-Archive` (.NET ZipArchive) chokes on ZIPs > 2 GB (BtbN FFmpeg, VS portable) and is memory-hungry.
- `py7zr` cannot decode BCJ2-filtered streams (vs-mlrt TensorRT runtime DLLs).
- `7za` is format-stable and portable.

Quick "when do I call what" map

Scenario	Scripts in order
Fresh install from scratch	builder_main → [01] → [10] → [11] → [12] → [13] → opt. [14] → [16] → doctor.py
Moved the project to another drive / machine	doctor.py (selfheal calls Repair-PipShims itself) — or manually <code>install\Repair-PipShims.cmd</code>
Need/update the ML stack	builder_main → [13] VS-MLRT LEAN after [10] / [11] / [12], then Bench-AllPresets with <code>-MLBackend sweep</code>
Want to confirm CUDA is actually alive	<code>install\Bench-CUDA.cmd <video></code> — measures real wall-time delta
Want to confirm all 28 presets work locally without NVIDIA	<code>system_core\powershell\pwsh.exe -NoLogo -NoProfile -ExecutionPolicy Bypass -File install\Bench-AllPresets.ps1 -ProjectRoot . -InputFile <video> -Frames 1 -Cuda off -MLBackend ort_cpu</code>
RTX / TensorRT validation	<code>system_core\powershell\pwsh.exe -NoLogo -NoProfile -ExecutionPolicy Bypass -File install\Bench-AllPresets.ps1 -ProjectRoot . -InputFile <video> -Frames 1 -Cuda sweep -MLBackend auto</code>
Disk is tight	builder_main → [16] CLEAN INSTALL CACHE — frees ~3.5 GB compressed archives
Updated NVIDIA driver / swapped GPU	doctor.py (live <code>bm3dcuda</code> smoke) → Bench-CUDA → Bench-AllPresets (TRT engines rebuild on first ML preset)
Added a new preset	Bench-AllPresets for smoke + register in <code>engine/presets.py</code> + add a launcher entry + update this wiki
Preparing release archive	builder_main → [09] Make release archive — output excludes <code>output/</code> , <code>logs/</code> , <code>._runtime/</code> , <code>install/download/</code> , <code>release/</code> , API keys, and private <code>MEMORY.md</code> ; <code>MEMORY.example.md</code> stays public

Additional — full reference

- English docstring per preset: at the top of each `system_core/presets/<palette>/<preset>.vpy`
- CLI flags: `runtime\python.exe system_core\main.py run --help`

- Profiles (cross-palette combinations): `runtime\python.exe system_core\main.py list-profiles`
- Recursive batch with mirror folder structure: `apply-profile-batch --recursive`
- Stack health: `runtime\python.exe system_core\doctor.py`
- AI / ML stack install and scenarios: see *Install & service scripts* section above → `Install-VS-mlrt.cmd`

AI / ML (Phase 18.B-ML) — vs-mlrt stack

These presets use **vs-mlrt** (ONNX inference inside VapourSynth). Backend is auto-selected: `trt` (TensorRT, NVIDIA, fastest) → `ort_dml` (DirectML, any DX12 GPU including Intel Xe / AMF) → `ort_cpu` (CPU fallback). Install via `builder_main.cmd` → `[13] VS-MLRT LEAN` (~3.5 GB download, ~6 GB extracted).

`vsmlrt_realesrgan_2x` ★ — ML 2x upscale

Real-ESRGAN — the de-facto reference for ML video super-resolution. On 1080p input it produces ~3840×2160 with face/fabric sharpness that's **unreachable** via Lanczos / Spline36 / Resolve SuperScale.

- Parameters: `model` (general-x4v3 default / animevideov3 / general-wdn-x4v3 / animejanaiV2-L1/L2/L3), `tile=384`, `tile_pad=16`, `backend=auto`
- Encoder: `prores_lt` (for downstream work) or `h265_crxf14` (final)
- Not in NLE: real ML upscale (NLE plugins are usually paid add-ons)
- First TensorRT pass compiles an engine in 30-120s (cached after that)

`soft_hd_rebuild_2x` — cleanup + ML reconstruction for soft HD

For formally-HD footage whose real detail is closer to 480-720p because of soft optics, binning, aggressive OLPF, old codecs, or archive-generation loss. It cleans the frame first, runs Real-ESRGAN 2x, then applies a conservative luma detail pass.

- Parameters: `rebuild=conservative|balanced|aggressive`, `cleanup=light|medium|strong`, plus Real-ESRGAN `model`, `tile`, `tile_pad`, `backend`
- Encoder: `prores_lt` for grading/handoff or `h265_crxf14` for final delivery
- When to use: soft 1080p, archive HD with low real frequency content, under-sampled consumer footage

vsm1rt_rife_60fps ★ — ML frame interpolation

RIFE 4.x — modern ML model for frame interpolation. Understands content (not just pixel motion) → handles occlusion, transparent objects, fades correctly. Visibly smoother than Resolve Optical Flow on complex motion.

- Parameters: **fps_mul=2.5** (24→60 default; alternatives 2/3/4), **model=rife_v4.6** (default; v4.4 / v4.9), **backend=auto**
- Encoder: **h264_crf17** or **prores_lt**
- When to use: 24fps film material → 60Hz target, or slow-mo from regular 24/30/60fps footage

dpir_denoise — ML denoise for heavy noise

DPIR — a deep network for denoise that **preserves** skin and fabric texture where BM3D / DFTTest start to smear. Use it when BM3D "kills" detail on high-ISO material.

- Parameters: **strength=10** (sigma 1-50; 5 light / 10 medium / 15 heavy / 25 very heavy / 50 extreme), **model=drunet_color** (default; gray for B/W; deblocking_color for JPEG/MPEG block artefacts), **tile=384**, **backend=auto**
- Encoder: **prores_lt** (handoff to colorist) or **h264_crf14** (final)
- When to use: ISO 6400+, low-light phone, broken-sensor archive

Last updated: 2026-05-27 — 28 presets across 4 palettes, Soft HD Rebuild 2X in Restoration; local CPU/ORT smoke **28/28 PASS** (**Frames=1**, **Cuda=off**, **MLBackend=ort_cpu**); RTX 5070 CUDA/auto smoke **36/36 PASS** (**Frames=1**, **Cuda=sweep**, **MLBackend=auto**).