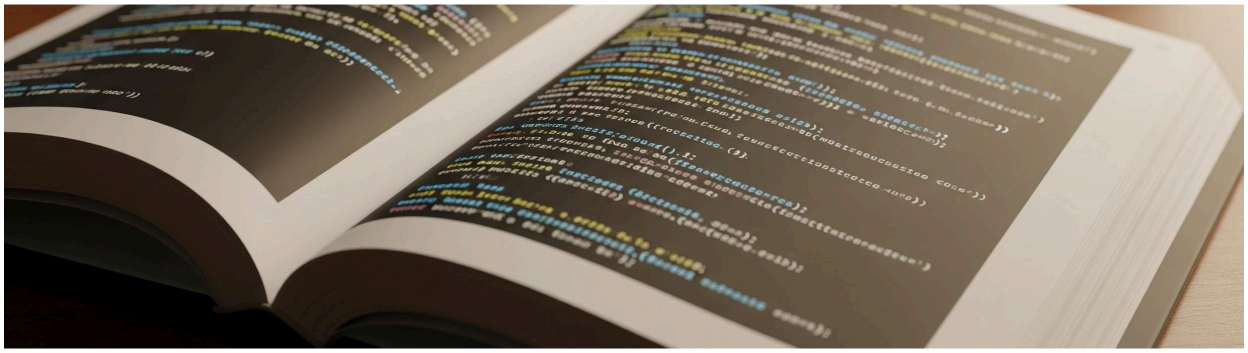


AUTO VMAF ENCODER: The Complete User Manual & Comprehensive Technical Documentation

An intelligent tool for perfectly optimized video encoding.

Version 1.0 Last Updated: August 3, 2025

Chapter 1: Introduction

Welcome to AUTO VMAF ENCODER! Think of this script as a smart robot for your video files. Its main job is to re-encode your videos to a modern format (AV1) while saving as much space as possible, without sacrificing visual quality.

Instead of you guessing the quality settings, AUTO VMAF ENCODER automatically runs tests to find the perfect balance between file size and how good the video looks to the human eye. It's a "fire and forget" tool designed to optimize your media library.

Who Is This For?

This tool is for anyone who wants to save disk space on their video collection, such as:

- Media collectors with large libraries of movies and TV shows.
- Anyone who wants to modernize their video files to a high-efficiency format.

What You'll Need (Prerequisites)

1. Python: The programming language the script is written in.
2. FFmpeg: The essential command-line tool that handles all video processing.
3. A VMAF Model File: A special data file that FFmpeg uses to calculate video quality.
4. (Optional) An NVIDIA GPU: If you want to use the much faster GPU encoding (nvenc). Otherwise, it will use your computer's CPU.
5. (Optional but highly recommended) PySceneDetect: A library for smart video analysis.

Chapter 2: Installation & Setup

Follow these steps carefully to get the script running.

Step 1: Install Python

If you don't have Python, download it from the official website.

1. Go to python.org/downloads
2. During installation, make sure to check the box that says "Add Python to PATH". This is very important.

Step 2: Set Up FFmpeg

FFmpeg is a command-line tool, not a normal program with an icon.

1. Download a **"full build"** of FFmpeg. A good source for Windows is the "Gyan.dev" builds. Found here: https://www.gyan.dev/ffmpeg/builds/
2. Unzip the downloaded file into a simple location, for example: C:\ffmpeg.
3. After unzipping, you should have a bin folder that contains ffmpeg.exe, ffprobe.exe. The path would be C:\ffmpeg\bin.

Step 3: Download VMAF

VMAF, which stands for Video Multimethod Assessment Fusion, is a perceptual video quality metric developed by Netflix. It is designed to mimic how a human would subjectively score the quality of a video by combining multiple quality assessment algorithms using a machine-learning model. This approach aims to provide a more accurate representation of perceived video quality than traditional metrics like PSNR (Peak Signal-to-Noise Ratio) or

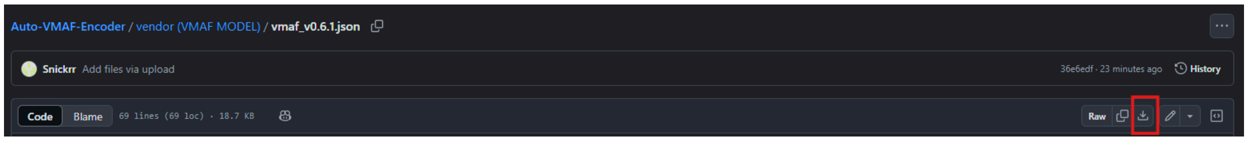
SSIM (Structural Similarity Index). VMAF is an open-source tool that has been widely adopted by the video streaming industry to help optimize video encoding and delivery, ensuring the best possible viewing experience for a given amount of bandwidth.

This script has been tested with vmaf_v0.6.1 which supports up to 1080p and vmaf_4k_v0.6.1, which supports up to 4K. While there is a 4k version, the difference in 4K HDR reference quality content is close to a margin of error.

While analyzing the Sony Swordsmith HDR UHD 4K Demo, with the same config.ini settings, the difference between VMAF 1080p and VMAF 4k models was of 0.02-0.04. For the sake of simplicity, this script uses the standard vmaf_v0.6 for all content.

1. Download vmaf_v0.6.1.json from my repository.

   Click on Vendor (VMAF MODEL) in my repository and download as "Raw" file.


Auto-VMAF-Encoder / vendor (VMAF MODEL) / vmaf_v0.6.1.json

Snickrr Add files via upload 36e6edf · 23 minutes ago History

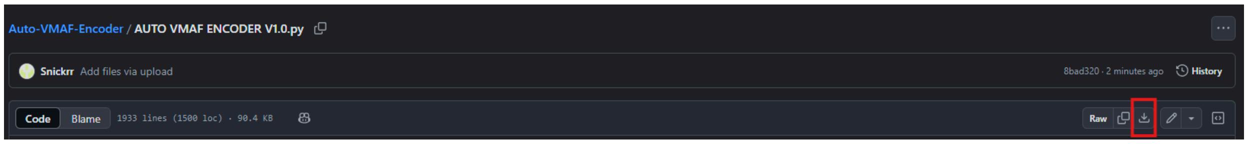
69 lines (69 loc) · 18.7 KB

2. Save downloaded file into a simple location, for example: C:\ffmpeg\bin

Step 4: Get the AUTO VMAF ENCODER Script

Place the two script files, **AUTO VMAF ENCODER. V1.op**y and **config.ini** either in an easily accessible file or in the same file as your videos.

Click on AUTO VMAF ENCODER V1.0.py in my repository and download the "Raw" file as per the image below. Same process for config.ini.


Auto-VMAF-Encoder / AUTO VMAF ENCODER V1.0.py

Snickrr Add files via upload 8bad320 · 2 minutes ago History

1933 lines (1500 loc) · 90.4 KB

Step 5: Install Required Libraries

The script needs some extra Python packages to work.

1. Open a Command Prompt (search for cmd in your Windows Start Menu).

Type the following command and press Enter. This tells Python's package manager (pip) to install the necessary libraries.

```
pip install rich psutil scenedetect[opencv]
```

Step 6: Configure the config.ini File

This is the most important step. Open the config.ini file with a text editor like Notepad. You must correctly set the paths to your tools.

Find the [Paths] section and update the three required paths to match the location where you put FFmpeg.

Always type paths with a "/" instead of the standard Windows "\".

```
ffmpeg_path = C:/ffmpeg/bin/ffmpeg.exe

ffprobe_path = C:/ffmpeg/bin/ffprobe.exe

vmaf_model_path = C:/ffmpeg/bin/vmaf_v0.6.1.json
```

Save the config.ini file. Your setup is now complete!

Step 7: Run the script

Double click on the script 🙂

Chapter 3: Understanding the config.ini - Your Control Panel

The config.ini file is your control panel for the script. It allows you to customize every aspect of the encoding process. This chapter explains what each setting does.

1. Core Settings

This section contains the fundamental settings that define your encoding job's primary goals.

- **encoder_type:** This is your main choice between speed and potential quality.
    - **svt_av1:** Uses your CPU. It's generally slower but can offer better compression efficiency, resulting in smaller files at a given quality level.
    - **nvenc:** Uses your NVIDIA GPU. It's significantly faster and is the recommended choice for most users.
- **target_vmaf:** This sets your quality goal on a scale of 0 to 100, where 100 is a perfect match to the original. A range of 95-98 is generally recommended as "visually lossless," meaning you are unlikely to notice any difference from the source video. Setting it to 100 is not recommended as it will create unnecessarily large files.
- **vmaf_tolerance:** This sets an acceptable margin of error for the target_vmaf score. For example, if your target is 98.0 and the tolerance is 0.2, any score of 97.8 or higher is considered a success.
- **cq_search_min and cq_search_max:** These settings define the "testing ground" for the script's quality search. The script will only test CQ/CRF values within this range (e.g., between 18 and 35).

---

2. Paths & File Handling

This section tells the script where to find its tools and how to manage your video files.

- **ffmpeg_path, ffprobe_path, vmaf_model_path:** These are required. They must point to the exact location of your FFmpeg tools and the VMAF model file.
- **database_path, vmaf_cache_path, performance_db_path:** These specify the locations where the script will store its log files and cache databases. You can generally leave these as the default.

- **delete_source_file:** Use with caution! If set to true, the original video file will be permanently deleted after a successful encode. It's safest to leave this as false.
- **output_suffix:** This is the text that gets added to the end of the new filename (e.g., MyVideo.mkv becomes MyVideo_av1.mkv).
- **output_directory:** If you want your new files saved in a different folder, specify the path here. Leave it blank to save in the same folder as the original.
- **use_different_input_directory and input_directory:** Set the first to true and provide a path in the second to make the script look for videos in a folder other than its own.
- **min_size_reduction_threshold:** A crucial safety net. The script will only keep the new file if it's at least this much smaller (in percent) than the original. For example, a value of 15.0 means the new file must be at least 15% smaller.
- **Skip_encoding_if_target_not_reached:** If your desired VMAF can't be reached within your CQ/CRF parameters, the video will not be encoded. Its original name will be kept, unless rename_skipped_files is = true.
- **rename_skipped_files:** If true, any file that is processed but ultimately skipped (e.g., for not meeting the size reduction threshold) will be renamed.
- **skipped_file_suffix:** The text to add to a file's name if it is skipped and the above setting is true.

---

3. Detailed Encoder Settings (NVENC)

This section contains settings specifically for the NVENC (NVIDIA GPU) encoder.

- **nvenc_preset:** Controls the speed versus quality balance. The preset values range from p1 (fastest) to p7 (best quality).
- **nvenc_quality_mode:** The quality algorithm used. hq (High Quality) is standard, while uhq (Ultra High Quality) can provide better results on newer GPUs.
- **extra_params:** For advanced users only. This allows you to add any extra command-line parameters directly to the FFmpeg command.

---

4. Detailed Encoder Settings (SVT-AV1)

This section contains settings specifically for the SVT-AV1 (CPU) encoder.

- **svt_av1_preset:** Controls the speed versus quality balance. Presets range from 0 (best quality, slowest) to 13 (fastest). A value of 6 is a great balanced choice.

- **svt_av1_film_grain:** Adds a cinematic film grain effect. 0 means no grain. A value of 8-12 adds a light, pleasant grain.
- **extra_params:** For advanced users only. This allows you to add any extra command-line parameters directly to the FFmpeg command.

---

5. Performance

These settings control how the script uses your computer's resources.

- **max_workers:** The number of video files to process at the same time. Set this according to your CPU cores and system resources to avoid slowdowns.
- **num_parallel_vmaf_runs:** The number of VMAF tests to run in parallel within each worker. This dramatically speeds up the quality search phase. A range of 3-5 is a good starting point.
- **max_iterations:** A safety limit for the quality search. The script will stop searching after this many rounds, even if it hasn't found a perfect match. A value of 7 is usually more than enough.

---

6. Caching

These settings control how the script can speed up future jobs and improve ETA predictions.

- **enable_cache:** Enables the VMAF Cache (The "Answer Key"). This saves the result of every VMAF test. If the script ever needs to run the exact same test again on the same file, it can retrieve the results instantly instead of re-doing the work.
- **enable_performance_cache:** Enables the Performance Cache (The ETA "Memory"). This logs the performance of completed jobs, using the historical data to make ETA predictions smarter and more accurate over time.

---

7. VMAF Sampling & Fine-Tuning

This section controls how the script analyzes videos to determine their quality.

- **sampling_method:** How the script chooses short clips for VMAF testing.

    - tier0 (Scene Detect) is the smartest option but requires the PySceneDetect library.
    - tier1 (Keyframe) and tier2 (Intervals) are reliable fallbacks.
- **sample_segment_duration:** The length, in seconds, of each video clip taken for analysis.
- **num_samples:** How many of these clips are taken from the video to be combined into one master sample for testing.
- **master_sample_encoder:** The encoder used to create the temporary sample file. raw is the recommended option.
- **min_scene_score:** (For tier0 only) The sensitivity for detecting scene changes. A lower number means it will detect more scenes.
- **min_scene_changes_required and min_keyframes_required:** The minimum number of scenes or keyframes that must be found for tier0 or tier1 to be used. If the video doesn't meet this threshold, the script will fall back to a simpler sampling method.
- **skip_start_seconds and skip_end_seconds:** (For tier1 & tier2) Tells the script to ignore the first and last few seconds of a video (like intros or credits) when taking samples.

---

8. File Filtering

These settings are rules to automatically skip files that don't meet certain criteria.

- **min_duration_seconds:** Skip files shorter than this many seconds.
- **min_filesize_mb:** Skip files smaller than this many megabytes.
- **min_bitrate_4k_kbps, min_bitrate_1080p_kbps, min_bitrate_720p_kbps:** Skip files already below a certain bitrate (in kbps). A value of 0 for any of these means the filter is not applied.

---

9. Output Color Depth

- **output_bit_depth:** Determines the color depth of the final video.
    - 10bit is highly recommended for AV1 as it improves quality and compression, even if the source is 8-bit.
    - source will keep the original bit depth.
    - 8bit will force the output to be 8-bit.

Chapter 4: Running the Script

Step 1: Place Your Video Files

By default, the script will look for video files in the same folder where it is located. You can either place your videos there or configure the input_directory in the config.ini.

Step 2: Double click the script

Step 3: Understanding the Output

The script will start and display a live dashboard.

- Configuration Summary: At the top, it will show you all the settings it loaded from your config.ini. Always check this to make sure it's what you intended.
- Encoding Summary: This is the main overview, showing total progress, time elapsed, and estimated time remaining (ETA).
- Worker Panels: Each file being processed will get its own "Worker" panel, showing a real-time log of what the script is doing (e.g., "Searching for VMAF," "Encoding...").

Chapter 5: Interpreting the Results

Output Files

Your new, smaller video files will be in the location you specified (output_directory or the source folder). They will have the output_suffix added to their name (e.g., MyVideo_av1.mkv).

The Log File (encoding_log.txt)

This file is a simple text log that contains a detailed summary of every file that was processed, whether it succeeded or was skipped. It's useful for reviewing a large batch job.

The Database Files (.db)

You will see vmaf_cache.db and performance.db files created. These are the script's "memory." They help it run faster on subsequent runs and make its ETA more accurate over time. You can safely leave these files alone. If you want to reset the script's memory, you can delete them, and the script will create new ones.

Chapter 6: Troubleshooting & FAQ

- **"Error: ffmpeg not found!"**
  - This means the paths in your config.ini are incorrect. Double-check that ffmpeg_path, ffprobe_path, and vmaf_model_path point to the exact location of those files. The easiest way to fix this is to put the script files in the same bin folder as ffmpeg.exe.
- **"The script crashes right away."**
  - The script runs a quick test on startup. If it fails, it means there's a problem with your FFmpeg installation or the VMAF model file. Ensure you downloaded a "full build" of FFmpeg and that the vmaf_model_path is correct.
- **"The script is running very slowly.**"
  - Check your encoder_type. If you have an NVIDIA graphics card, using nvenc will be dramatically faster than svt_av1 (CPU). If you are using the CPU, a lower svt_av1_preset (like 4 or 5) is very slow but gives higher quality. A preset of 6 or 7 is a good balance.
- **"My computer becomes unusable while the script is running."**
  - Your max_workers setting is likely too high. This number determines how many videos are processed at the same time. A good rule of thumb is to set it to half the number of your CPU cores. For example, if you have an 8-core CPU, set max_workers = 4.
- **"Why did the audio or subtitles not copy correctly?"**
  - The script uses a simple copy command (-c:a copy -c:s copy) that works for most files. However, videos with multiple audio or subtitle tracks can sometimes be complex. The script tries to copy the first of each type it finds. If you have a file with specific track needs, you may need to process it manually with FFmpeg.
- **"The colors in my new video look washed out."**
  - This can happen with HDR (High Dynamic Range) videos. Ensure your output_bit_depth in the [Output] section is set to 10bit, as this is required to preserve HDR color.
- **"My ETA is wrong at the beginning."**

    - This is normal for the very first run! The ETA system needs to process a few files to learn how fast your computer is. It will become much more accurate over time as it builds up its performance.db file.
- **"It skipped my file!"**
    - Check the filtering rules in your config.ini. The script will skip files that are too short, too small, or have too low a bitrate. It will also skip any file that already has the output_suffix (e.g., _av1) or skipped_file_suffix (e.g., _notencoded) in its name.
- **"How do I safely stop the script?"**
    - Pressing Enter in the console window will tell the script to stop queuing new files. It will wait for the currently running jobs to finish safely. To force-quit immediately, press Ctrl+C, but be aware this may leave temporary files behind.

AUTO VMAF ENCODER: Comprehensive Technical Documentation

Executive Summary

The AUTO VMAF ENCODER is a sophisticated Python script designed for automated video transcoding with perceptual quality optimization. It leverages the VMAF (Video Multimethod Assessment Fusion) metric to intelligently determine optimal encoding parameters, ensuring output videos maintain a target perceptual quality while maximizing compression efficiency. The system supports both NVIDIA hardware acceleration (NVENC) and software encoding (SVT-AV1), featuring advanced capabilities such as complexity-aware processing, intelligent sampling strategies, and comprehensive performance optimization.

Core Architecture and Design Philosophy

The script embodies a data-driven approach to video encoding, moving beyond traditional fixed-bitrate or constant-quality methods. Instead of relying on predetermined encoding parameters, it dynamically analyzes each video's characteristics and iteratively searches for the optimal quality setting that achieves a specific VMAF score. This approach ensures consistent perceptual quality across diverse content types while maximizing compression efficiency.

The architecture follows a modular design pattern with clear separation of concerns. Configuration management, media analysis, encoding operations, and performance tracking operate as distinct subsystems that interact through well-defined interfaces. This modularity enhances maintainability and allows for easy extension of functionality.

Major Functional Components

1. Configuration Management System

The configuration system serves as the central nervous system of the application, managing all operational parameters through an INI file interface. The `EncodingSettings` dataclass encapsulates these settings, providing type safety and clear documentation of available options.

The configuration covers several critical domains:

Path Management: The system requires paths to FFmpeg, FFprobe, and VMAF model files. It also manages database paths for caching and performance tracking, ensuring all external dependencies are properly configured.

Performance Tuning: Parameters control parallel processing capabilities, including the number of concurrent file encodings and simultaneous VMAF calculations. Memory limits prevent system overload, while iteration counts balance search accuracy against processing time.

Encoder Configuration: The system supports multiple encoding backends with extensive customization options. For NVENC, users can specify presets, quality modes, and advanced parameters. SVT-AV1 configurations include preset levels, film grain synthesis, and custom encoding parameters.

Quality Targeting: The heart of the system's intelligence lies in its VMAF targeting parameters. Users specify a target VMAF score with acceptable tolerance, along with search ranges and clamping values to guide the optimization process.

2. Media Analysis and Filtering System

Before processing begins, the system performs comprehensive media analysis to determine video characteristics and assess encoding viability. This analysis serves multiple purposes:

Technical Validation: The system extracts detailed metadata including duration, bitrate, resolution, pixel format, and color space information. This data informs encoding decisions and ensures compatibility with selected encoders.

Content Filtering: Sophisticated filtering rules prevent processing of files unlikely to benefit from re-encoding. Files below minimum duration, size, or bitrate thresholds are automatically skipped. Resolution-specific bitrate requirements ensure already-compressed content isn't further degraded.

Complexity Assessment: The system analyzes content complexity through scene detection, keyframe distribution, or bitrate analysis. This complexity data influences sampling strategies and performance predictions, leading to more accurate processing time estimates.

3. Intelligent Sampling System

The sampling system represents one of the most innovative aspects of the encoder. Rather than analyzing entire videos (computationally expensive) or using fixed sample points (potentially unrepresentative), it employs a tiered approach to identify optimal sample segments:

Tier 0 - Scene Detection: When PySceneDetect is available, the system performs sophisticated scene change analysis. It identifies transition points between distinct visual segments, using these as natural sampling boundaries. The system also calculates complexity metrics based on scene duration and cut frequency, providing valuable data for performance prediction.

Tier 1 - Keyframe Analysis: As a fallback or alternative, the system analyzes keyframe distribution within the video. Keyframes, which represent complete image frames rather than differences, often correlate with scene changes or significant visual transitions. The distribution and frequency of keyframes provide both sampling points and complexity indicators.

Tier 2 - Temporal Sampling: When scene-based methods aren't applicable, the system falls back to evenly distributed temporal samples. Even this basic approach includes intelligence, such as avoiding intro/outro sections that may not represent the main content.

The sampling system's multi-tiered approach ensures robust operation across diverse content types while maximizing the representativeness of selected samples.

4. VMAF-Based Quality Optimization Engine

The quality optimization engine represents the core innovation of the system. It implements an intelligent binary search algorithm to find the optimal CQ (Constant Quality) or CRF (Constant Rate Factor) value that achieves the target VMAF score:

Master Sample Creation: The system creates a concatenated sample containing all selected segments, encoded with lossless or near-lossless settings. This master sample serves as the reference for all VMAF comparisons, ensuring consistency across quality tests.

Parallel Search Strategy: Rather than testing quality values sequentially, the system tests multiple CQ/CRF values simultaneously within each iteration. This parallel approach dramatically reduces search time while maintaining accuracy.

Intelligent Caching: VMAF calculations are computationally expensive, so the system implements a sophisticated caching mechanism. Cache keys incorporate file hashes, sample timestamps, and complexity scores, ensuring cache validity while maximizing reuse opportunities.

Adaptive Search Bounds: The search algorithm dynamically adjusts its bounds based on results, quickly narrowing the search space while ensuring the optimal value isn't missed. Clamping parameters prevent the selection of extreme values that might produce undesirable results.

5. Production Encoding System

Once optimal parameters are determined, the production encoding system handles the full video transcoding:

Color Space Preservation: The system carefully preserves color space, primaries, and transfer characteristics from the source video. This attention to detail ensures encoded videos maintain proper color reproduction across different playback environments.

Pixel Format Intelligence: Rather than using fixed pixel formats, the system intelligently selects appropriate formats based on source characteristics, encoder capabilities, and user preferences. It handles 8-bit to 10-bit conversions, chroma subsampling variations, and encoder-specific requirements.

Stream Mapping: The encoder preserves all video, audio, and subtitle streams from the source file. Audio and subtitles are copied without re-encoding, maintaining quality while minimizing processing time.

Atomic File Operations: To prevent data loss, the system encodes to temporary files and only replaces originals after successful completion and size validation. This approach ensures system crashes or interruptions don't result in corrupted or lost files.

6. Performance Monitoring and Prediction System

The performance monitoring system goes beyond simple progress tracking, implementing sophisticated prediction algorithms:

Component-Based Timing: The system tracks performance of individual operations (sample creation, VMAF calculation, encoding) separately. This granular data enables accurate predictions for future files with similar characteristics.

Complexity-Aware Predictions: Performance predictions incorporate content complexity scores, recognizing that action scenes or rapidly changing content require more processing time than static footage.

Adaptive Learning: As the system processes files, it continuously refines its performance model. Early predictions improve in accuracy as more data becomes available, with exponential smoothing preventing outliers from skewing estimates.

Real-Time ETA Updates: The system provides countdown-style ETA calculations that update in real-time between file completions, giving users accurate progress information throughout the batch processing.

7. Memory Management System

Process Tree Monitoring: Rather than monitoring only the main Python process, the system tracks memory usage across all spawned FFmpeg processes.

8. User Interface and Progress Visualization

The rich console interface provides comprehensive real-time information about encoding progress:

Multi-Level Progress Display: The interface simultaneously shows overall batch progress, individual file progress, and sub-task progress (such as VMAF iterations). This hierarchical display keeps users informed at all granularity levels.

Worker-Specific Panels: Each parallel encoding task receives its own panel showing recent log messages and current operation progress. This design allows users to monitor multiple simultaneous operations without confusion.

Intelligent Layout Management: The interface dynamically adjusts to show active workers while maintaining a clean, organized display. Completed or idle worker slots are hidden to maximize screen real estate.

Color-Coded Information: The system uses consistent color coding to convey information types: cyan for informational messages, yellow for warnings, red for errors, and green for successful completions.

Advanced Features and Optimizations

In-Memory Processing Pipeline

The system implements sophisticated in-memory processing for VMAF calculations, minimizing disk I/O and improving performance. Encoded samples are passed between processes using pipes and temporary memory-backed files, reducing latency and wear on storage devices.

Intelligent Cache Key Generation

The caching system uses multi-factor cache keys incorporating file content hashes, sample timestamps, and complexity scores. This approach ensures cache validity while enabling reuse across similar operations, dramatically reducing redundant calculations.

Encoder-Specific Optimizations

The system includes deep knowledge of encoder characteristics, applying appropriate settings for each backend. For NVENC, it leverages hardware capabilities while working around limitations like restricted pixel format support. For SVT-AV1, it configures advanced features like film grain synthesis and scene detection parameters.

Failure Recovery and Logging

Comprehensive error handling ensures graceful degradation when issues occur. Failed files are logged with specific failure reasons, temporary files are cleaned up, and batch processing continues despite individual file failures. Detailed logs capture all operations for post-processing analysis.

Dynamic Resource Allocation and Load Balancing

One of the most elegant aspects of this system is how it manages computational resources dynamically. The script doesn't simply spawn a fixed number of workers and hope for the best. Instead, it implements a sophisticated work-stealing pattern where new tasks are only initiated when resources become available.

Consider the interplay between the max_workers setting and the num_parallel_vmaf_runs parameter. These aren't independent values but rather form a resource matrix. When a worker begins processing a file, it may spawn multiple VMAF calculations simultaneously during the quality search phase. The system must balance these competing demands to prevent resource exhaustion. For instance, with 4 workers and 3 parallel VMAF runs, the system could theoretically have 12 FFmpeg processes running simultaneously during peak load. The memory manager actively monitors this situation, tracking not just the Python process but the entire process tree.

The Complexity Score: A Hidden Intelligence Layer

The complexity scoring system deserves special attention as it represents a form of content-aware intelligence that permeates the entire encoding pipeline. When the system analyzes a video, it doesn't just count scenes or keyframes - it builds a mathematical model of the content's visual complexity.

For scene-based analysis, the system examines the distribution of scene durations. A video with many quick cuts (scenes under 2 seconds) receives a higher complexity score than one with long, stable shots. This isn't arbitrary - rapid scene changes typically indicate action sequences, music videos, or other content that's harder to compress efficiently. The formula complexity_score = min(1.0, (scenes_per_minute / 30.0) * 0.5 + quick_cut_ratio * 0.5) elegantly combines scene frequency with cut speed to produce a normalized complexity indicator.

This complexity score then influences multiple downstream decisions. The performance prediction system uses it to adjust time estimates, recognizing that complex content takes longer to encode. The caching system incorporates complexity into cache keys, ensuring that VMAF results from a simple video aren't incorrectly applied to complex content from the same source.

The Art of VMAF Calculation Optimization

The VMAF calculation pipeline showcases several clever optimizations that might not be immediately apparent. First, the system creates a "master sample" encoded with near-lossless settings. This serves as the reference for all quality comparisons, but the encoding choice here is crucial. The system supports three options: hardware-accelerated H.264 (via NVENC), software H.264 (via x264), or raw video.

Why offer these choices? Each has trade-offs. Hardware encoding is fast but may introduce subtle artifacts even at maximum quality. Software encoding is slower but more predictable. Raw video eliminates compression artifacts entirely but requires enormous memory bandwidth. The choice depends on available hardware and quality requirements.

The in-memory processing pipeline for VMAF calculations is particularly sophisticated. Rather than writing encoded samples to disk and reading them back, the system uses pipes and memory-mapped temporary files. This approach reduces I/O latency and SSD wear while enabling parallel processing. The careful management of temporary files with immediate cleanup prevents disk space exhaustion during large batch operations.

Intelligent Progress Tracking and User Feedback

The progress tracking system goes far beyond simple percentage displays. It implements a multi-layered approach to user feedback that provides both immediate responsiveness and accurate long-term predictions.

At the micro level, the system parses FFmpeg's stderr output in real-time, extracting time codes and frame numbers to provide smooth progress updates during encoding operations. The regex patterns used here are carefully crafted to handle various FFmpeg output formats while maintaining parsing efficiency.

At the macro level, the system maintains a sophisticated state machine tracking not just completed files but predicted remaining time with continuous refinement. The performance ratio calculation (old_ratio * 0.7) + (ratio * 0.3) implements exponential smoothing, preventing sudden ETA jumps while allowing the system to adapt to changing conditions.

The visual presentation uses the Rich library's capabilities to create an information-dense yet readable display. Color coding isn't just aesthetic - it creates a visual language where users can assess system state at a glance. The dynamic panel system that shows only active workers prevents information overload while maintaining visibility into parallel operations.

Error Recovery and Graceful Degradation

The system's approach to error handling demonstrates defensive programming at its finest. Rather than allowing failures to cascade, each component implements local error recovery with graceful degradation.

Consider the sampling tier system. If PySceneDetect isn't available, the system doesn't fail - it seamlessly falls back to keyframe detection. If that fails (perhaps due to a corrupted file), it falls back again to temporal sampling. Each tier provides progressively less sophisticated analysis, but the system continues functioning.

The encoding pipeline similarly handles partial failures intelligently. If a file can't be encoded, it's logged with the specific failure reason, optionally renamed to indicate its status, and processing continues with the next file. Temporary files are cleaned up even in error cases, preventing disk space leaks.

The Hidden Performance Database

The performance database system acts as the script's long-term memory, learning from every encoding operation to improve future predictions. This isn't just simple averaging - it's a sophisticated statistical model that considers multiple factors.

The database schema includes not just timing data but contextual information: resolution, encoder settings, complexity scores, and sampling methods. When predicting performance for a new file, the system queries for similar files, using resolution and complexity as primary matching criteria. If insufficient data exists for a specific complexity level, the system gracefully degrades to broader matches.

The component-based timing breakdown is particularly clever. By tracking sample creation, VMAF search, and final encoding separately, the system can predict performance even when settings change. For example, if a user increases the number of VMAF samples, the system can adjust only the sampling component of its prediction while keeping encoding estimates unchanged.

Color Science and Format Preservation

The system's handling of color spaces and pixel formats demonstrates deep understanding of video encoding challenges. Modern video content includes various color spaces (Rec. 709, Rec. 2020, DCI-P3), transfer functions (SDR, HDR10, HLG), and pixel formats (4:2:0, 4:2:2, 4:4:4, different bit depths).

The script doesn't simply preserve these attributes - it intelligently maps between source and encoder capabilities. For instance, NVENC requires the P010LE pixel format for 10-bit encoding, while SVT-AV1 uses standard planar formats. The system handles these conversions transparently while preserving color accuracy.

The build_color_args function extracts color metadata and ensures it's properly propagated to the output file. This attention to detail prevents color shifts that could occur if metadata were lost during encoding.

Configuration Philosophy and User Experience

The configuration system embodies a philosophy of progressive disclosure. Basic users can modify simple parameters like target VMAF scores and worker counts, while advanced users can access deep encoder customization through the advanced parameter strings.

The configuration validation includes intelligent defaults and safety checks. CPU core detection ensures the system doesn't oversubscribe resources. Bitrate thresholds prevent re-encoding of already heavily compressed content. The minimum size reduction threshold ensures processing time isn't wasted on marginal improvements.

Conclusion: A System Greater Than Its Parts

What makes this script exceptional isn't any single feature but how all components work together to create an intelligent, adaptive system. The complexity scoring informs performance prediction. The caching system accelerates iterative searches. The memory manager enables aggressive parallelization while preventing system instability. Each component enhances the others, creating emergent behaviors that go beyond simple automation to approach true intelligence in video encoding decisions.

This holistic design, combined with careful attention to error handling, user experience, and extensibility, creates a system that's both powerful for experts and accessible for general users. It represents not just a tool but a framework for thinking about quality-driven video encoding in an automated world.