On VMAF's property in the presence of image enhancement operations

Zhi Li, Netflix
July 13, 2020 (Last updated Aug. 1, 2021)

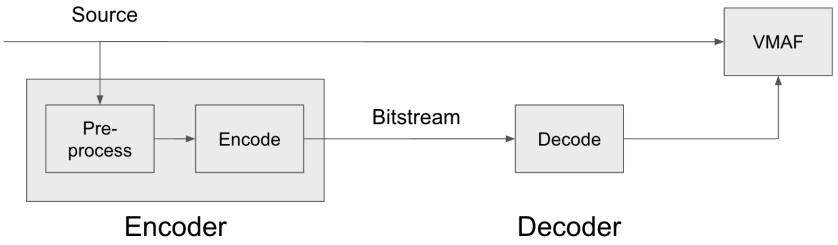
A high-level slide deck summarizing the ideas can be found in this link. Leave questions as comments to this memo, or direct your questions to zli@netflix.com.

TL;DR: ***We discuss VMAF's property in the presence of image enhancement operations, and propose a solution for applications such as codec evaluation where the pre-/post-processing gain needs to be disabled. To disable the enhancement gain, a user can use the `vmaf_v0.6.1neg.json` model ("neg" stands for "no enhancement gain") with the `vmaf` executable from the open source package.***

VMAF, or Video Multi-method Assessment Fusion, is a video quality assessment algorithm developed by Netflix. VMAF was designed with Netflix's streaming use case in mind, in particular, to capture the video quality of professionally generated movies and TV shows, in the presence of compression and scaling artifacts, as perceived by end viewers. At Netflix, VMAF has been applied in codec evaluation, encoding optimization, A/B experimentation, among others.

Since its open-sourcing on Github, a number of new use cases of VMAF have started to emerge, including gaming, VR and user-generated content. In these applications, it is not uncommon to include custom image enhancement operations in the video processing pipeline, in order to enhance the end viewer's subjective experience. As developers started optimizing their video pipelines, it became evident that certain image enhancement techniques, including *sharpening*, *contrasting*, *histogram equalization*, among others, could boost VMAF scores. While such operations could enhance the image quality as perceived by the end viewers if applied properly, it is evident that VMAF tends to overpredict the perceptual quality even when such operations are overused.

More importantly, these operations can be applied within an encoder as its pre-processing step, or, if standardized, as a normative post-processing step within a decoder. Most recently, Google introduced a tune=vmaf mode in libaom as an option to perform quality-optimized AV1 encoding with pre-processing. We highly respect this effort. On the other hand, in *codec evaluation*, it is often desirable to measure the pure gain achieved from compression without also accounting for the gain from image enhancement. As demonstrated by the block diagram below, since it is difficult to strictly separate an encoder from its pre-processing step (especially for proprietary encoders), it may become difficult to use VMAF to assess the pure compression gain.



We consider that there is value for VMAF to disregard the gain introduced by the enhancements that are not part of the codec. In other words, there needs to be knob(s) in VMAF through which we can control the gain measurable from pre-/post-processing enhancement operations.

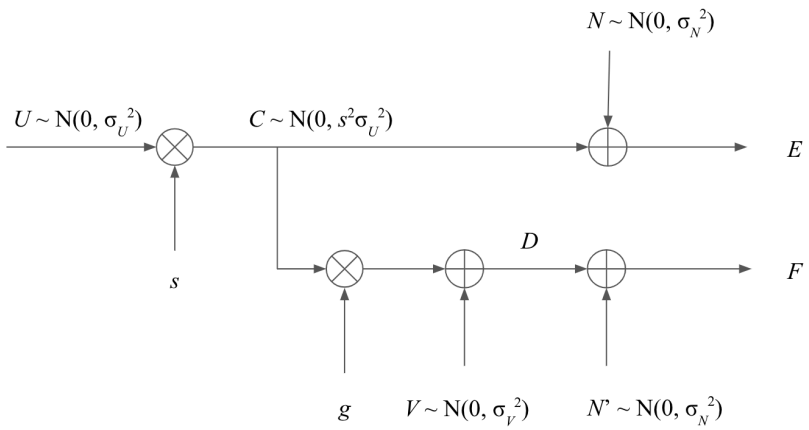
In this memo, we start by discussing the foundations on why VMAF can capture the quality gain from image enhancement operations, which put VMAF in a different class compared to traditional metrics like PSNR or SSIM. Having understood the foundations, we proceed to discuss our proposed modification to the VMAF implementation, where two knobs are introduced to control the measured image enhancement gain. We present experiment results to demonstrate that the modification does not hurt VMAF's correlation with subjective scores in the presence of compression and scaling artifacts. Lastly, we provide details on how a user of the VMAF open-source package could disable or limit the enhancement gain.

Foundations

In a nutshell, VMAF combines human vision modeling with machine learning. It builds on top of a number of state-of-the-art image quality metrics, including visual information fidelity (VIF) and detail loss metric (DLM) as its building blocks, and rely on support vector regression (SVR) to fuse the predictions from elementary metrics into a final score, to be aligned with the mean opinion scores (MOS) collected from subjective experiments. The key to understanding how VMAF captures the image enhancement gain is to understand the properties of the elementary metrics. To be more specific, *both VIF and DLM have incorporated considerations for image enhancement gains*.

VIF

VIF is based on the premise that quality is complementary to the measure of information loss. Rooted in information theory, VIF models the image quality assessment problem as a communication channel, and the final VIF score can be considered as the ratio between two mutual information measures $I(C;\, F \mid s)$ and $I(C;\, E \mid s)$. This is illustrated in the system diagram below:



Here C represents the original source signal, and E presents the source signal as perceived by the human vision system (HVS). The source follows a model of $C = sU$, where s can be thought as the "context" in the source signal which can be estimated through low-pass filtering, and U is a Gaussian random variable characterizing the local variability. The HVS is modeled as a simple white additive noise channel with noise N. $I(C;\ E \mid s)$ represents the maximum information that can be conveyed through this channel. Similarly, D represents the distorted signal and F represents the distorted signal as perceived by the HVS. The distortion is modeled by two terms: 1) a gain term g, and 2) an additive noise term $V \sim \mathrm{N}(0, \sigma_V^2)$. Given the input C and the output D, the parameters g and σ_V can be solved by maximum likelihood estimation (MLE) with the following optimal solution:

$$g = \sigma_{CD}/\sigma_C^2$$
$$\sigma_V^2 = \sigma_D^2 - g \cdot \sigma_{CD}$$

where σ_{CD} is the covariance between C and D. It is crucial to understand the gain term g. Regular operations such as lossy compression and downsampling act as low-pass filtering, and will result in a gain $g < 1$. However, image enhancement operations, such as sharpening and contrast enhancement, could result in a gain $g > 1$. Take a linear contrast operation for example, where the entire image is multiplied by 1.5, the resulting estimated terms are $g = 1.5$ and $\sigma_V = 0$.

VMAF implements the pixel-domain version of VIF, where the coefficients are calculated in the pixel domain after successive Gaussian filtering. In total, coefficients are calculated in 4 different scales. After simplification, the VIF value at scale λ ($\lambda = 1,\ldots, 4$) can be calculated as:

$$VIF_{\lambda} = \frac{\sum_{i=1}^{N} \log_2\left(1+\frac{g_i^2 s_i^2 \sigma_u^2}{\sigma_{v_i}^2+\sigma_N^2}\right)}{\sum_{i=1}^{N} \log_2\left(1+\frac{s_i^2 \sigma_u^2}{\sigma_N^2}\right)} \qquad (1)$$

In VMAF version 0.6.x, the VIF scores calculated in the 4 different scales are thrown into the SVR as 4 distinct features. It is noted that in a regular use case with compression and scaling, where the gain term $g < 1$ and $\sigma_{V_i}^2 > 0$, the resulting VIF has value ≤ 1. As a special case, when C and D are identical, $g = 1$ and $\sigma_{V_i}^2 = 0$, the VIF has a value of precise 1.

For image enhancement operations, such as the linear contrast operation as mentioned above, if plugging the term $g = 1.5$ and $\sigma_V = 0$, one can immediately see that this will result in a VIF value of > 1. And this will translate into a gain in the final VMAF after the SVR.

DLM

DLM is part of a perceptual image quality metric that separately evaluates detail losses and additive impairments. The premise is that humans respond to the losses of details and the addition of impairments differently, and it is worth treating them separately before combining them. Due to historical reasons, the term ADM is used in the VMAF source code as the combination of the terms DLM (detail loss metric) and AIM (additive impairment metric), but it is in fact only the DLM part of the metric that is used in VMAF.

DLM operates in the wavelet domain. Specifically, it uses 4-scale Daubechies 2 (db2) wavelets. After the wavelets decomposition, a key step is to decompose the target image T into a restored image R and an additive impairment image A, guided by the original image O. Specifically: $T = R + A$, where the restored image can be calculated as:

$$R = clip_{[0,1]}\left(\frac{T}{O}\right) \cdot O \qquad (2)$$

Here T, R, A and O each represents a coefficient of a location (x, y) in a subband θ of scale λ.

A special case is made to handle contrast enhancement. The authors propose to detect contrast enhancement operation using the rule $|\psi_O - \psi_T| < 1^{\circ}$, where ψ is the angular representation (i.e. arctan) of the ratio between two coefficients co-located in the vertical subband ($\theta = 2$) and the horizontal subband ($\theta = 4$) of the same scale λ. This intuitively

makes sense. Consider the simple example of linear contrast where the original image O is multiplied by 1.5 to produce the target image T. Since the magnitude of the coefficients gets proportionally scaled by 1.5, the ratio between the vertical and the horizontal subbands remain unchanged, and thus $\psi_O = \psi_T$. The more complicated image enhancement operations such as sharpening and histogram equalization can be considered as localized linear contrasting operations. The coefficient in the restored image is updated to:

$$R = T, \text{ if } |\psi_O - \psi_T| < 1^\circ \qquad (3)$$

One can easily see that without triggering the contrast enhancement rule, R is bounded to $|R| \leq |O|$. On the other hand, with the contrast enhancement rule triggered, it is possible to have $|R| = |T| > |O|$. This is crucial to understand the image enhancement gain in DLM.

After decoupling, the restored signal R and the original signal O go through a contrast sensitivity function (CSF) and a contrast masking (CM) function. The final formula of DLM can be summarized as:

$$DLM = \frac{\sum_{\lambda=1}^{4}\sum_{\theta=2}^{4}\left(\sum_{i,j\in center} CM(CSF(R(\lambda,\theta,i,j)))^3\right)^{\frac{1}{3}}}{\sum_{\lambda=1}^{4}\sum_{\theta=2}^{4}\left(\sum_{i,j\in center} CSF(O(\lambda,\theta,i,j))^3\right)^{\frac{1}{3}}} \qquad (4)$$

The coefficients in R after CSF and CM (and O after CSF) are then Minkowski-pooled with power 3, and summed within the center region of each subband with a border factor 0.1. The results are then summed over the vertical ($\theta = 2$), horizontal ($\theta = 4$) and diagonal ($\theta = 3$) subbands, and then over the 4 scales ($\lambda = 1,\ldots, 4$). For the formula, one can easily see that with the contrast enhancement rule triggered, $|R| > |O|$ will result in a DLM value of > 1. And this will translate into a gain in the final VMAF after the SVR.

Modifications

Having understood the foundations on why VMAF's elementary metrics VIF and DLM incorporate considerations for image enhancement gains, let us look at possible modifications to the two metrics such that the enhancement gains can be fully or partially disabled.

VIF

From Equation (1), it is straightforward to see that to boost the VIF values, the "positive" influence from g_i needs to outweigh the "negative" influence from $\sigma_{V_i}^2$. We empirically find

that to limit each local gain term g_i to below 1.0 is a good strategy to limit the image enhancement gain while balancing the impact on VMAF's correlation with subjective scores. Intuitively speaking, a upper limit of g_i at 1.0 eliminates the possibility of having VIF's value to go above 1.0. To control the enhancement gain measurable by VIF, we introduce a parameter called *VIF enhancement gain limit* EGL_{VIF}, where $EGL_{VIF} \geq 1.0$. In the VMAF source code, this parameter is named `vif_enhn_gain_limit`. The proposed modification is

$$g_i = \min(g_i, EGL_{VIF})$$

To fully disable the enhancement gain, we simply set $EGL_{VIF} = 1.0$.

DLM

To control the enhancement gain achievable by DLM, we modify Equation (3) such that increment in R can be constrained. Similar to VIF, we introduce a parameter called *DLM enhancement gain limit* EGL_{DLM}, where $EGL_{DLM} \geq 1.0$. In the VMAF source code, this parameter is named `adm_enhn_gain_limit` instead. The proposed modification to (3) is:

- $R = \min\left(R \cdot EGL_{DLM}, T\right)$, if $|\psi_O - \psi_T| < 1^\circ$ and $R > 0$;
- $R = \max\left(R \cdot EGL_{DLM}, T\right)$, if $|\psi_O - \psi_T| < 1^\circ$ and $R < 0$.

One can make the following observations:

- With EGL_{DLM} set to 1.0, the modification restrains R's value to $|R| \leq |O|$, thus effectively limiting the DLM value to below 1.0.
- With EGL_{DLM} set to a large value, say 100.0, the new formulation is almost identical to the original Equation (3), with one exception: when T and O are of different signs, or $T \cdot O \leq 0$. We have empirically verified that the occurrence of this exception is rare, and the numerical change to the DLM and the final VMAF value is infinitesimally small.

Results

We first demonstrate the modifications' impact on the VMAF prediction in the presence of pure image enhancement operations. For simplicity, we only illustrate the case where the enhancement gain is fully disabled. That is, we set $EGL_{VIF} = 1.0$ and $EGL_{DLM} = 1.0$.

The images below illustrate pure image enhancement operations on the first frame of Akiyo (352x288), without any compression. The left is the original image; the center is the image after applying unsharp filtering with radius 1.0 and amount 1.0; the right is the image after applying histogram equalization with clip limit 0.005.

Original

Sharpening

Histogram Equalization

The table below summarizes the elementary feature scores and the final VMAF prediction, with the enhancement gain enabled/disabled. For each column, the VMAF calculation is against the original image (for example, the first column is original vs. original; the second column is sharpening vs. original, and so on).

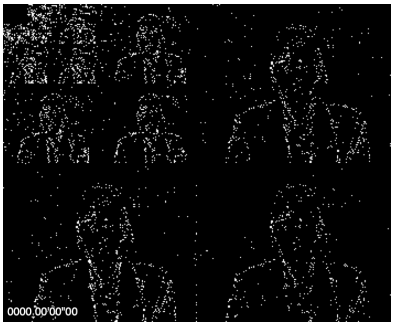
		Original	Sharpening	Histogram Equalization
Enhn. Gain Enabled	VIF Scale 0	1.0	0.7616	1.0537
	VIF Scale 1	1.0	0.9639	1.0791
	VIF Scale 2	1.0	0.9924	1.0773
	VIF Scale 3	1.0	0.9993	1.0689
	DLM	1.0	1.0692	1.1682
	VMAF	97.4277	111.9868*	144.0195*
Enhn. Gain Disabled	VIF Scale 0	1.0	0.6772	0.9455
	VIF Scale 1	1.0	0.9396	0.9643
	VIF Scale 2	1.0	0.9766	0.9617
	VIF Scale 3	1.0	0.9900	0.9560
	DLM	1.0	0.9531	0.9380
	VMAF	97.4280	85.3330	78.7122

*Note: by default, the VMAF score is clipped between $[0, 100]$ in the last step of the calculation. To better illustrate the numerical change, we disable the clipping (by the command line option `disable_clip`).

Next, we show how VMAF behaves with the enhancement gain disabled in libaom's tune=vmaf mode. The image on the left below is the first frame of Akiyo (352x288) compressed with libaom tune=vmaf mode and CQ 43. The center is the visualization of the enhancement gain achievable by VIF, where the gray level represents the strength of the gain, in 4 scales concatenated horizontally. The right is the visualization of the enhancement gain achievable by DLM represented in a wavelet fashion, where the white represents the loci where the gain is positive.

libaom tune=vmaf

VIF enhancement gain



DLM enhancement gain

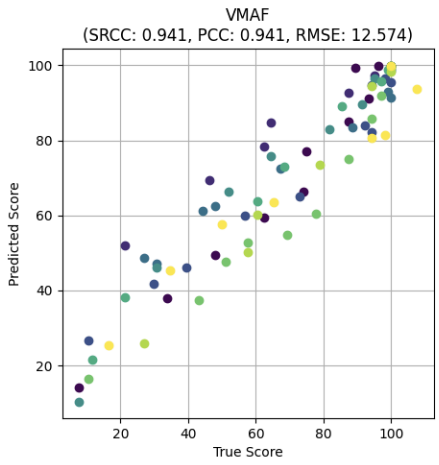
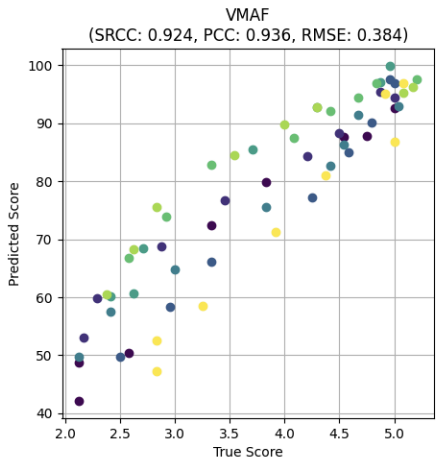
The table below summarizes the elementary feature scores and the final VMAF prediction, with the enhancement gain enabled/disabled. As comparison, we also show the result on the first frame of a libaom encode with CQ 43 but no tune=vmaf, with enhancement gain enabled/disabled.

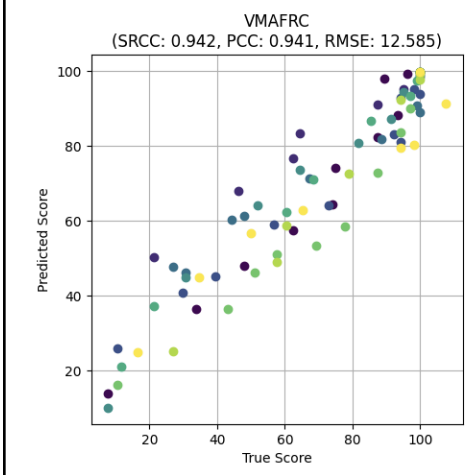
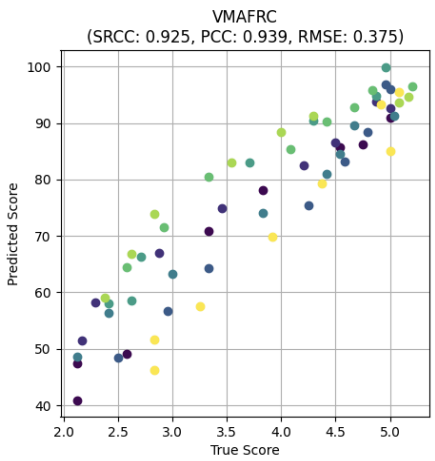
		libaom tune=vmaf	libaom
Enhn. Gain Enabled	VIF Scale 0	0.7053	0.7979
	VIF Scale 1	0.9653	0.9843
	VIF Scale 2	0.9906	0.9931
	VIF Scale 3	0.9967	0.9958
	DLM	1.0375	0.9928
	VMAF	104.8277*	95.1425
Enhn. Gain Disabled	VIF Scale 0	0.6403	0.7686
	VIF Scale 1	0.9436	0.9769
	VIF Scale 2	0.9763	0.9879
	VIF Scale 3	0.9879	0.9921
	DLM	0.9645	0.9873

	VMAF	87.6951	93.4151

*Note: by default, the VMAF score is clipped between $[0, 100]$ in the last step of the calculation. To better illustrate the numerical change, we disable the clipping (by the command line option `disable_clip`).

Lastly, we demonstrate VMAF's prediction accuracy compared to the subjective scores, on two datasets: 1) NFLX Public dataset (compression and scaling distortions) and 2) VQEG HD3 dataset (compression distortion only). The plots below show the scatter plot of the predicted scores (y-axis) versus the MOS scores (x-axis). The numbers are reported in SRCC (Spearman rank-order correlation coefficient), PCC (Pearson linear correlation coefficient, after sigmoid fitting) and RMSE (root mean-squared error, after sigmoid fitting). By showing that the correlation number does not move before and after muting the enhancement gain, *we virtually demonstrate that the modification only affects the corner cases of enhancement gains*, thus does not impact the accuracy of VMAF in the traditional use case. One thing to note is that the absolute score of VMAF does drop slightly, typically by 1~3.

	NFLX Public	VQEG HD3
Enhn. Gain Enabled		

Enhn. Gain Disabled		

Usages

The option to disable the enhancement gain is implemented in the VMAF open-source package. Note that there is an old executable `vmafossexec` and a new executable `vmaf`. The option to disable the enhancement gain is only implemented in the new executable `vmaf`. We are in the process of making the new `libvmaf` API compatible with `FFmepg`.

We offer two options to disable the enhancement gain:

- a) by taking in a new VMAF model file `vmaf_v0.6.1neg.json` ("neg" stands for "no enhancement gain") instead of using the default model file `vmaf_v0.6.1.json`.
- b) by directly controlling the enhancement gain knobs through command-line options. This option allows disabling enhancement gain to work with any models (e.g. `vmaf_b_v0.6.3.json`).

1) Get the latest VMAF master from:
https://github.com/Netflix/vmaf

2) Build by (VMAF build depends on python3 meson and ninja. Check here for instructions):
`make`

3) Download test videos from:
https://github.com/Netflix/vmaf_resource/tree/master/python/test/resource/yuv
The following two video sequences will be needed:
`refp_vmaf_hacking_investigation_0_0_akiyo_cif_notyuv_0to0_identity_vs_akiyo_cif_notyuv_0to0_multiply_q_352x288`
`disp_vmaf_hacking_investigation_0_0_akiyo_cif_notyuv_0to0_identity_vs_akiyo_cif_notyuv_0to0_multiply_q_352x288`

4) Run VMAF on a linearly contrasted akiyo frame:

```
./libvmaf/build/tools/vmaf --reference
./python/test/resource/yuv/refp_vmaf_hacking_investigation_0_0_akiyo_
cif_notyuv_0to0_identity_vs_akiyo_cif_notyuv_0to0_multiply_q_352x288
--distorted
./python/test/resource/yuv/disp_vmaf_hacking_investigation_0_0_akiyo_
cif_notyuv_0to0_identity_vs_akiyo_cif_notyuv_0to0_multiply_q_352x288
--width 352 --height 288 --pixel_format 420 --bitdepth 8 --output
/dev/stdout --model path=./model/vmaf_v0.6.1.json:name=vmaf
```

This will generate output:

```
<VMAF version="2.1.1">
  <params qualityWidth="352" qualityHeight="288" />
  <fyi fps="160.93" />
  <frames>
    <frame frameNum="0" integer_adm2="1.116700"
integer_adm_scale0="1.044966" integer_adm_scale1="1.104599"
integer_adm_scale2="1.127935" integer_adm_scale3="1.137287"
integer_motion2="0.000000" integer_motion="0.000000"
integer_vif_scale0="1.052402" integer_vif_scale1="1.070149"
integer_vif_scale2="1.072518" integer_vif_scale3="1.072513"
vmaf="100.000000" />
  </frames>
  ...
</VMAF>
```

5a) Run VMAF on a linearly contrasted akiyo frame with the “no enhancement gain” model `vmaf_v0.6.1neg.json`:

```
./libvmaf/build/tools/vmaf --reference
./python/test/resource/yuv/refp_vmaf_hacking_investigation_0_0_akiyo_
cif_notyuv_0to0_identity_vs_akiyo_cif_notyuv_0to0_multiply_q_352x288
--distorted
./python/test/resource/yuv/disp_vmaf_hacking_investigation_0_0_akiyo_
cif_notyuv_0to0_identity_vs_akiyo_cif_notyuv_0to0_multiply_q_352x288
--width 352 --height 288 --pixel_format 420 --bitdepth 8 --output
/dev/stdout --model name=vmaf:path=./model/vmaf_v0.6.1neg.json
```

This will generate output:

```
<VMAF version="2.1.1">
  <params qualityWidth="352" qualityHeight="288" />
  <fyi fps="141.56" />
  <frames>
    <frame frameNum="0" integer_adm2_egl_1="0.957433"
integer_adm_scale0_egl_1="0.979451"
integer_adm_scale1_egl_1="0.964563"
integer_adm_scale2_egl_1="0.956307"
integer_adm_scale3_egl_1="0.947698" integer_motion2="0.000000"
integer_motion="0.000000" integer_vif_scale0_egl_1="0.983708"
integer_vif_scale1_egl_1="0.997443"
integer_vif_scale2_egl_1="0.998483"
integer_vif_scale3_egl_1="0.999151" vmaf="88.030464" />
  </frames>
  ...
</VMAF>
```

5b) Run VMAF on a linearly contrasted akiyo frame, with enhancement gain disabled through command-line options (`vif_enhn_gain_limit=1.0` and `adm_enhn_gain_limit=1.0`):

```
./libvmaf/build/tools/vmaf --reference
./python/test/resource/yuv/refp_vmaf_hacking_investigation_0_0_akiyo_
cif_notyuv_0to0_identity_vs_akiyo_cif_notyuv_0to0_multiply_q_352x288
--distorted
./python/test/resource/yuv/disp_vmaf_hacking_investigation_0_0_akiyo_
cif_notyuv_0to0_identity_vs_akiyo_cif_notyuv_0to0_multiply_q_352x288
--width 352 --height 288 --pixel_format 420 --bitdepth 8 --output
/dev/stdout --model
version=vmaf_v0.6.1:vif.vif_enhn_gain_limit=1.0:adm.adm_enhn_gain_lim
it=1.0
```

This will generate output:

```
<VMAF version="2.1.1">
  <params qualityWidth="352" qualityHeight="288" />
  <fyi fps="167.08" />
  <frames>
    <frame frameNum="0" integer_adm2_egl_1="0.957433"
integer_adm_scale0_egl_1="0.979451"
integer_adm_scale1_egl_1="0.964563"
integer_adm_scale2_egl_1="0.956307"
integer_adm_scale3_egl_1="0.947698" integer_motion2="0.000000"
```

```
integer_motion="0.000000" integer_vif_scale0_egl_1="0.983708"
integer_vif_scale1_egl_1="0.997443"
integer_vif_scale2_egl_1="0.998483"
integer_vif_scale3_egl_1="0.999151" vmaf="88.030464" />
  </frames>
  ...
</VMAF>
```

6) Run VMAF with additional metrics, such as PSNR, SSIM, MS-SSIM:

```
./libvmaf/build/tools/vmaf --reference
./python/test/resource/yuv/refp_vmaf_hacking_investigation_0_0_akiyo_
cif_notyuv_0to0_identity_vs_akiyo_cif_notyuv_0to0_multiply_q_352x288
--distorted
./python/test/resource/yuv/disp_vmaf_hacking_investigation_0_0_akiyo_
cif_notyuv_0to0_identity_vs_akiyo_cif_notyuv_0to0_multiply_q_352x288
--width 352 --height 288 --pixel_format 420 --bitdepth 8 --output
/dev/stdout --model name=vmaf:path=./model/vmaf_v0.6.1neg.json
--feature float_psnr --feature float_ssim --feature float_ms_ssim
```

This will generate output:

```
<VMAF version="2.1.1">
  <params qualityWidth="352" qualityHeight="288" />
  <fyi fps="23.96" />
  <frames>
    <frame frameNum="0" integer_adm2_egl_1="0.957433"
integer_adm_scale0_egl_1="0.979451"
integer_adm_scale1_egl_1="0.964563"
integer_adm_scale2_egl_1="0.956307"
integer_adm_scale3_egl_1="0.947698" integer_motion2="0.000000"
integer_motion="0.000000" integer_vif_scale0_egl_1="0.983708"
integer_vif_scale1_egl_1="0.997443"
integer_vif_scale2_egl_1="0.998483"
integer_vif_scale3_egl_1="0.999151" float_psnr="24.383040"
float_ssim="0.986562" float_ms_ssim="0.991559" vmaf="88.030464" />
  </frames>
  ...
</VMAF>
```