uvgVenctester: Open-Source Test Automation Framework for Comprehensive Video Encoder Benchmarking

Joose Sainio, Alexandre Mercat, and Jarno Vanne
Ultra Video Group, Tampere University, Finland
{joose.sainio, alexandre.mercat, jarno.vanne}@tuni.fi

ABSTRACT

The agile and efficient development of modern video encoders calls for automated testing methodologies. This paper presents the first-of-its-kind open-source test automation framework called uvgVenctester (github.com/ultravideo/uvgVenctester) that is designed for comprehensive performance and conformance testing of video encoders with the desired set of test video sequences. Our framework comes with built-in support for the popular AVC, HEVC, VVC, VP9, and AV1 video coding formats and the state-of-the-art HM, Kvazaar, x265, VTM, VVenC, SVT-VP9, and SVT-AV1 video encoders. Furthermore, there are no technical limitations of adopting other formats or encoders. The developers can evaluate the encoder of interest under the three primary usage scenarios: 1) conformance testing of the encoded bitstream; 2) rate-distortion-complexity comparison with the other encoders; and 3) systematic exploration of encoding parameters. The framework provides commonly used analysis tools to quantify encoding quality, speed, and bitrate with versatile set of absolute and comparative results such as Bjøntegaard Delta (BD)-Rate for PSNR, SSIM, and VMAF quality metrics. The supported output formats include CSV, graph, and comparison table. They ensure that the results are available in human and machine-readable formats. To the best of our knowledge, the proposed framework is currently the most comprehensive and modular open-source software toolset for video encoder benchmarking.

CCS CONCEPTS

• Computing methodologies ~ Computer graphics ~ Image compression • Software and its engineering ~ Software creation and management ~ Software verification and validation ~ Empirical software validation

KEYWORDS

Open source, Video encoder, Test automation, Performance comparison, Conformance testing

MMSys 21, September 28-October 1, 2021, Istanbul, Turkey

ACM ISBN 978-1-4503-8434-6/21/09…$15.00
https://doi.org/10.1145/3458305.3478445

ACM Reference format:

Joose Sainio, Alexandre Mercat, and Jarno Vanne. 2021. uvgVenctester: Open-Source Test Automation Framework for Comprehensive Video Encoder Benchmarking. In *Proceedings of ACM Multimedia Systems Conference (MMSys'21). ACM, Istanbul, Turkey, 6 pages.* https://doi.org/10.1145/3458305.3478445

1 Introduction

Cisco has estimated that 82% of all IP traffic will be video by 2022 [1]. To mitigate the pervasive growth of video data, several video coding formats have been released during the past couple of decades. The current landscape of international MPEG/ITU-T video coding standards is dominated by the universal *Advanced Video Coding* (*AVC/H.264*) [2], well-established *High Efficiency Video Coding* (*HEVC/H.265*) [3], and emerging *Versatile Video Coding* (*VVC/H.266*) [4]. In addition, the *Alliance for Open Media* (*AOM*) and its predecessors have published VP9 [5] and *AOM Video 1* (*AV1*) [6] as royalty-free options for the MPEG/ITU-T standards.

The video coding community typically benchmarks video encoders according to the *common test conditions* (*CTC*) specified for different standards, such as those for HEVC [7] and VVC [8]. The main objective of CTC is to harmonize experimental setups for encoder benchmarking by specifying encoding configurations and test video sequences.

Video encoder benchmarking is commonly based on three *key performance indicators* (*KPIs*): encoding quality, bitrate, and speed [9], of which the first KPI can further be split into subjective and objective visual quality assessments. Subjective assessments, such as the *mean opinion score* (*MOS*), entail human evaluation and inherently cannot be automated by machines, so they are omitted from this work. Objective quality metrics quantify the distortion of an encoded video signal by analyzing it mathematically. The most widely-used objective metrics are *Peak Signal to Noise Ratio* (*PSNR*), *Structural Similarity Index* (*SSIM*) [10], and *Video Multimethod Assessment Fusion* (*VMAF*) [11]. *Rate-distortion* (*RD*) performance comparison between two encoders is usually carried out by calculating *Bjøntegaard delta bitrate* (*BD-rate*) [12] that couples the first two KPIs and measures bitrate difference for equal visual quality. The third KPI, encoding speed, depends mainly on the underlying computing platform.

Table 1: Main features of the existing and proposed test automation toolsets for video encoder evaluation.

Toolset	Analysis				Automated		Output formats	User interface	License
	Quality	Bitrate	Speed	BD-Rate	encoding	building			
MSU VQMT [13]	×	-	-	-	No	No	CSV, JSON	GUI, CLI	Paid
EPFL VQMT [14]	×	-	-	-	No	No	CSV	CLI	Non-commercial
ITU-T P.1204.3 Ref. Imp. [15]	×	-	-	-	No	No	JSON	CLI	Non-commercial
FFMetrics [16]	×	-	-	-	No	No	Graph	GUI	Not specified
Proposed	×	×	×	×	**Yes**	**Yes**	**CSV, Graph, Table**	**CLI**	**BSD**

In practice, the optimal tradeoff between these three KPIs depends on the media application of interest and many design iterations are typically needed before all requirements are met. In particular, tackling the design complexity of the latest video coding standards requires extensive design space exploration. Therefore, automated testing methodologies are key to the efficient development of modern video encoders and their productization.

Table 1 characterizes the existing toolsets involved in or enabling the automation of video encoder evaluation. *MSU Video Quality Measurement Tool* (*MSU VQMT*) [13] is the most complete toolset among existing approaches. It supports quality measurement with 12 different metrics, *Graphical User Interface* (*GUI*), *Command Line Interface* (*CLI*), as well as CSV and JSON output formats. However, the CLI, commercial use, and input videos larger than 720p require a paid license. *EPFL Video Quality Measurement Tool* (*VQMT*) [14] is built using the OpenCV library and it is able to calculate up to six different quality metrics simultaneously. Its main drawback is the restrictive license. *ITU-T P.1204.3* [15] is a standard for assessing the observed video quality from a standalone video bitstream. It comes with a reference implementation that is released under the same restrictive license as VQMT. Finally, *FFMetrics* [16] is a graphical frontend for the popular multimedia library *FFmpeg* [17] for visualizing PSNR, SSIM, and VMAF at frame level. It has no licensing information.

A common limitation of these existing toolsets is that they all focus on visual quality assessment only. In addition, none of them support automated encoder building or execution. These shortages together with restrictive licenses discourage their wider adoption in different encoder development activities.

This paper presents the *uvgVenctester* (*UVG Video Encoder Tester*) test automation framework for video encoders. The software is being developed by *Ultra Video Group* (*UVG*) at *Tampere University* (*TAU*). The source code of *uvgVenctester* is written in Python and made available on GitHub at

https://github.com/ultravideo/uvgVenctester

under the permissive BSD 2-Clause license. This cross-platform software can be run on *Windows* and *Linux* through CLI.

To the best of our knowledge, *uvgVenctester* is the most comprehensive open-source test automation framework for video encoder evaluation and validation, from automatic encoder building and execution to well-digested absolute and comparative analysis results in human and machine-readable formats. The comprehensive documentation and usage examples support the convenient adoption of the framework and design modularity makes it easily extensible with new video encoders, analysis tools, and output formats. All these features have been designed with one goal in mind: to save video codec developers the trouble of creating dedicated testing environments from scratch for each individual encoder design project.

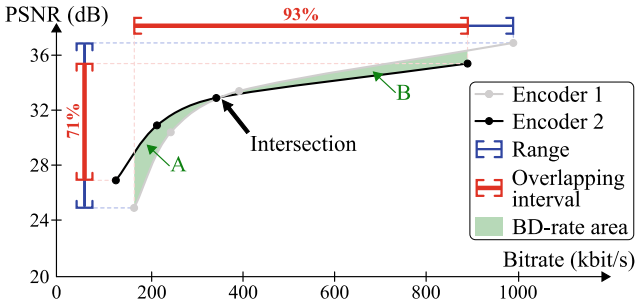


Figure 1: Visualization of RD curves of two encoders.

The rest of this paper is structured as follows. Section 2 introduces the basics of video encoder evaluation. An overview of the proposed uvgVenctester framework is given in Section 3 followed by an example usage scenario in Section 4. Section 5 addresses the extensibility aspects of our framework. Finally, Section 6 concludes the paper.

2 Video Coding Benchmarking

In video encoder evaluation, PSNR is the most popular objective visual quality metric. Even though it does not reflect the quality perceived by the *human visual system* (*HVS*), it is still widely adopted in the video coding research and development due to its relatively low computational complexity. SSIM metric [10] was originally developed as a replacement for PSNR in image quality analysis but it is also increasingly used in video quality assessment. It measures the structural similarity between images in order to pay special attention to objects in the visual scene and thereby correspond to subjective quality better. Unlike PSNR and SSIM, VMAF metric [11] is particularly designed for video quality analysis. It is based on machine learning techniques and addresses both the spatial and temporal domains of video for more in-depth correlations with HVS characteristics.

The state-of-the-art video encoders offer several mechanisms for setting the *RD target*. Adjusting the *quantization*

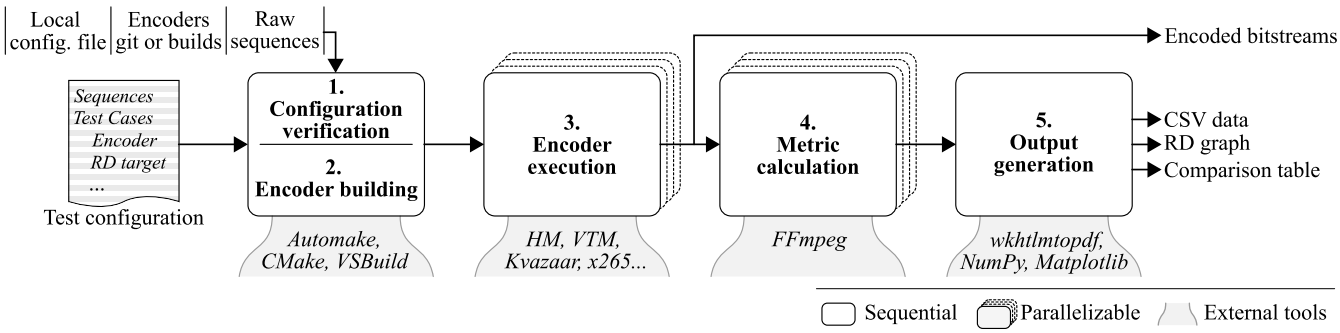


Figure 2: uvgVenctester workflow and integration with external media processing tools.

parameter (*QP*) is academically the most common approach [7], [8] and it usually has the largest effect on the video coding quality and bitrate. However, setting a specific QP value does not result in a specific quality or bitrate but the output bitrate varies strongly as a function of video content. In real-life scenarios, such as live streaming or broadcasting, unpredictable bitrate is detrimental and therefore not recommended. To overcome this, *rate control* (*RC*) algorithms are needed to meet the bitrate constraints. Additionally, *constant rate factor* (*CRF*) can be used for more constant quality in scenarios that do not have bitrate constraints.

Bitrate and quality of encoded video are closely interdependent in that quality tends to improve as a function of bitrate. Therefore, they are commonly combined into a single metric, either BD-rate [12] or BD-distortion. BD-rate reports the bitrate difference of two encoders setups for the same quality, whereas BD-distortion reports the quality difference for the same bitrate. In both cases, the quality can be measured with PSNR, SSIM, or VMAF metric. Both BD-rate and BD-distortion are computed as the integral of the difference between the overlapping parts of the RD curves, which are shaped using cubic fitting.

Figure 1 depicts an example of two cubically fitted RD curves used to compute BD-rate. The horizontal and vertical red bars represent the overlapping intervals of the RD curves in terms of bitrate and quality, respectively. Since the example RD curves intersect, the BD-rate is calculated as the difference between the green overlapping areas (*A* and *B*). However, summing up negative and positive values complicates the interpretation of BD-rate. For example, the BD-rate computation gives 12% better BD-rate for Encoder 1 in Figure 1, but it is unable to express that Encoder 2 has better quality with high bitrates.

Let the encoding speed be derived from a wall-clock execution time of an encoding process. In practice, the encoding speed measurement is not straightforward to arrange because it depends on the experimental setup and conditions. For example, modern operating systems perform many tasks in the background and the CPUs dynamically switch between power states based on the load and thermal constraints. These random factors introduce measurement noise, which can be mitigated by executing the tests multiple times and averaging the obtained results. Additionally, operating systems cache the encoder binary and input video sequence, so it is better to discard the first sample or otherwise cache the encoder and sequence in advance.

3 uvgVenctester Test Automation Framework

Figure 2 depicts the workflow of the proposed test automation framework with external tool dependencies. A single test is defined by a test configuration file that specifies the applied test video sequences and test cases. Furthermore, each test case is composed of an encoder configuration and RD targets.

The framework also requires a local configuration file, encoder binaries or git repositories, and raw test video sequences in YUV format. The local configuration file contains all variables that remain static between multiple test configurations, e.g., Visual Studio version and git repository URLs. Currently, the framework supports *HEVC Test Model* (*HM*) [18], Kvazaar [19], x265 [20], *VVC Test Model* (*VTM*) [21], both versions of *Fraunhofer Versatile Video Encoder* (*VVenC*) [22], *Scalable Video Technology* (*SVT*)-VP9 [23], and SVT-AV1 [24] encoders. Raw test video sequences are available in open datasets like [25]–[27].

The framework generates outputs in three formats: CSV data, RD graph, and comparison table. Additionally, encoded bitstreams are provided as an intermediate result. The proposed framework is written in Python, taking advantage of its verbosity and large standard and auxiliary libraries, e.g., NumPy [28] and Matplotlib [29].

The workflow of the proposed framework is split into five steps: 1) configuration verification; 2) encoder building; 3) encoder execution; 4) metric calculation; and 5) output generation. In Figure 2, the first two steps are combined as they are interleaved.

3.1 Configuration Verification

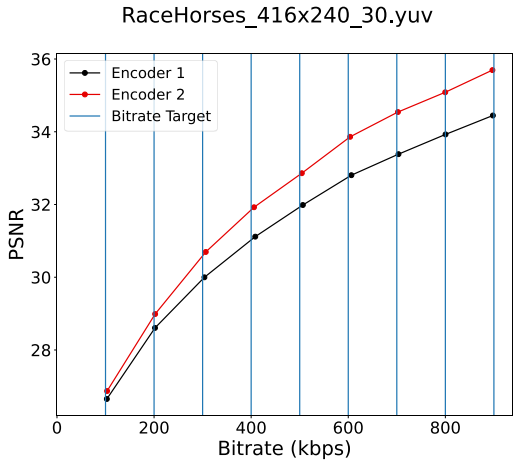
In the beginning, the test setup is verified to minimize errors that may occur during testing; some test runs can take several days or weeks. The verification is split into two phases. The first phase ensures that all necessary tools, e.g., *git*, *CMake* [30], and *FFmpeg* [17], as well as the needed test video sequences are available.

The second phase is performed after the encoders are built. In this phase, the framework parses the test cases and confirms their compliance by encoding a single all-black frame with minimum

(a)

```
Encoder;Sequence;QP;PSNR;PSNR BD-rate;FPS;Speedup;
Encoder 1;PeopleOnStreet;22;43.23;0.0;0.75;1.0;
Encoder 1;PeopleOnStreet;27;42.52;0.0;0.87;1.0;
Encoder 1;PeopleOnStreet;32;40.32;0.0;0.95;1.0;
Encoder 1;PeopleOnStreet;37;38.87;0.0;1.06;1.0;
Encoder 2;PeopleOnStreet;22;43.07;0.104;0.57;0.75;
Encoder 2;PeopleOnStreet;27;42.27;0.104;0.65;0.76;
Encoder 2;PeopleOnStreet;32;40.07;0.104;0.71;0.77;
Encoder 2;PeopleOnStreet;37;38.53;0.104;0.80;0.74;
Encoder 1;Traffic;22;41.85;0.0;0.86;1.0;
Encoder 1;Traffic;27;40.13;0.0;0.93;1.0;
Encoder 1;Traffic;32;38.35;0.0;1.04;1.0;
Encoder 1;Traffic;37;36.21;0.0;1.18;1.0;
Encoder 2;Traffic;22;41.35;0.132;0.65;0.74;
Encoder 2;Traffic;27;39.87;0.132;0.70;0.73;
Encoder 2;Traffic;32;37.99;0.132;0.78;0.73;
Encoder 2;Traffic;37;34.83;0.132;0.89;0.72;
```

(b)



(c)

Video	BD-BR (PSNR)	BD-BR (SSIM)	BD-BR (VMAF)	Speedup
hevc-A	**17.0%**	**43.1%**	**-10.8%**	**1.17×**
PeopleOnStreet_2560x1600_30	13.9%	40.8%	-6.9%	1.13×
Traffic_2560x1600_30	20.1%	45.4%	-14.6%	1.20×
hevc-B	**20.3%**	**47.1%**	**-12.4%**	**1.26×**
BasketballDrive_1920x1080_50_500	5.4%	29.5%	-16.5%	1.37×
BQTerrace_1920x1080_60_600	36.4%	86.3%	-15.3%	1.27×
Cactus_1920x1080_50	25.2%	49.3%	-3.7%	1.24×
Kimono1_1920x1080_24	10.3%	25.5%	-14.6%	1.24×
ParkScene_1920x1080_24	23.9%	44.7%	-11.8%	1.17×
Averages	**19.3%**	**45.9%**	**-11.9%**	**1.23×**

Anchor: Test name: Encoder 1
encoder: Kvazaar
version: origin/master
Using QP: [22, 27, 32, 37]
- --preset ultrafast

Test name: Encoder 2
encoder: x265
version: origin/master
Using QP: [22, 27, 32, 37]
- --preset ultrafast

Figure 3: Example of a generated output. (a) CSV data. (b) RD graph. (c) Comparison table.

supported resolution. The verification is performed implicitly, and manual intervention is only required if any incompatibilities are detected.

3.2 Encoder Building

The framework can either use prebuilt encoders or build them from source. For each encoder, the framework checks whether the corresponding encoder version is already available. Otherwise, it fetches the correct version of the source code using *git* and builds the encoder with the platform-dependent build system. Currently, the framework is compatible with *Visual Studio* and *CMake* [30] via *Visual Studio* on *Windows* as well as *Automake* [31] and *CMake* on *Linux*. Additionally, the framework supports compiler definitions. Providing complete build support is out of scope of the proposed framework, but it can cover most usual use cases.

3.3 Encoder Execution

Encoder execution is usually the longest step of testing. This step loops through all sequences, test cases, and RD targets specified in the test configuration. The encoding tasks are either executed fully sequentially or distributed between multiple parallel instances. The task distribution is included because the currently supported reference encoders have no built-in concurrency.

Currently, the framework accepts the following RD targets: 1) QP; 2) CRF; and 3) four bitrate scaling methods including a) static, i.e., the same bitrate for all sequences; b) linear scaling with the number of pixels regardless of frame rate; c) scaling with the square root of the number of pixels; and d) linear scaling with the number of pixels and frame rate.

The framework also implements two techniques to improve the measurement accuracy: the warmup phase and encoding reiteration. The warmup phase ensures that the encoder binaries and sequences are cached by the operating system. This is necessary, e.g., with hard disks as their read speeds have a noticeable effect on the encoding speed, particularly with faster encoders. The number of encoding reiterations can be specified for all test cases, which are then run accordingly, and the results are averaged.

3.4 Metric Calculation

The calculated metrics can be divided into two categories: absolute and comparative metrics. The metric calculation can be sped up by using multiple parallel instances. If the encoder reiteration is enabled, variances are calculated over all iterations. Calculating variances facilitates efforts to accurately sample the encoding time, evaluate non-deterministic algorithms, or identify non-deterministic behaviour with algorithms that should be deterministic.

The absolute metrics include encoding time, frame rate, bitrate, PSNR, SSIM, VMAF, bitrate error, and bitstream conformance. PSNR, SSIM, and VMAF are measured using the respective *FFmpeg* filters [17]. Since the calculation of VMAF is computationally expensive, the filter graph is dynamically built

based on the actually required metrics. Bitrate error measures the difference between the target and actual bitrate of the encoder. It can only be measured when one of the bitrate scaling methods is selected as the RD target.

The comparative metrics include speedup, BD-rate, BD-distortion, proportional RD curve overlapping, and the number of RD curve intersections. Speedup is reported as the relative difference in encoding speed between the compared encoders. Both BD-rate and BD-distortion can be calculated with PSNR, SSIM, and VMAF quality metrics. As illustrated with red bars in Figure 1, the proportional RD curve overlapping indicates the relative bitrate and quality intervals, where both RD curves overlap. The number of RD curve intersections equal the number of times the RD curves intersect in the RD space (only once in Figure 1). The BD-rate calculation is not relevant unless the RD curves overlap, and these two RD curve metrics save the trouble of having to manually check the validity of the RD curves for each comparison.

3.5 Output Generation

Currently, the framework supports three different output formats: CSV data, RD graph, and comparison table, which are depicted in Figures 3(a)–(c), respectively. The fully customizable fields of the CSV data are composed of metadata and analysis results. The RD graphs are generated using *matplotlib* [32]. The comparison table can be created in both HTML and PDF output formats. The PDF table is generated from the HTML version using *wkhtmltopdf* [33].

4 Example Usage Scenario

The encoder under test can be evaluated with the following three primary usage scenarios:

1) Conformance testing of the encoded bitstream;
2) Rate-distortion-complexity comparison with other encoders; and
3) Systematic exploration of encoding parameters.

Listing 1 gives an example of the usage scenario 2 and more advanced ones are available on the *uvgVenctester* repository.

This example Python code is used to benchmark Kvazaar [19] and x265 [20] *ultrafast* presets. Lines 1–3 import the local configuration file, the necessary data structures, and the benchmarked encoder structures, respectively. Lines 6–8 define a list of the used test video sequences, which can be declared by regular expressions.

Line 10 initiates a test case for Kvazaar and lines 11–14 specify the following mandatory parameters for it:

- name: a unique name for the test case;
- encoder_type: the encoder under test;
- encoder_revision: the encoder version (git branch, commit, tag, or named prebuilt version);
- cl_args: the command-line arguments for testing.

Lines 15–23 detail the optional parameters and their default values (except for anchor_names):

Listing 1: Example of a test configuration.

```
1  import my_cfg
2  from tester import Tester, Test, QualityParam
3  from tester.encoders import Kvazaar, X265
4
5  def main():
6      sequences = [
7          "hevc-[A-F]/*.yuv",
8      ]
9
10     kvz = Test(                      #test case 1
11         name="Kvazaar",
12         encoder_type=Kvazaar,
13         encoder_revision="master",
14         cl_args="--preset ultrafast",
15         anchor_names=["Kvazaar"],
16         quality_param_type=QualityParam.QP,
17         quality_param_list=[22, 27, 32, 37],
18         seek=0,
19         frames=None,
20         rounds=1,
21         encoder_defines=[],
22         use_prebuilt=False,
23         env={},
24     )
25
26     x265 = kvz.clone(                #test case 2
27         name="x265",
28         encoder_type=X265,
29     )
30
31     context = Tester.create_context(
32         [kvz, x265],
33         sequences
34     )
35
36     Tester.run_tests(
37         context,
38         parallel_runs=1
39     )
40
41     Tester.generate_csv(
42         context,
43         "example.csv",
44         parallel_calculations=4
45     )
46
47 if __name__ == '__main__':
48     main()
```

- anchor_names: list of test case name(s) with which the current test case is compared (empty by default);
- quality_param_type: RD targets, as specified in Section 3.3;
- quality_param_list: list of the values for the selected RD target;
- seek: start frame for encoding;
- frames: the maximum number of encoded frames (None stands for all frames);
- rounds: number of encoding iterations;
- encoder_defines: list of #define directives used in the building step;
- env: dictionary of environmental values passed to the encoder instance.

Lines 26–29 create a copy of the Kvazaar test case with a new name and encoder_type set to x265. Lines 31–34 create a context for the test, gathering all test cases and video sequences. This step

also performs the verification and builds the encoders. Lines 36–39 run the tests with a single instance (parallel_runs set to 1), i.e., sequentially.

Lines 41–45 generate the csv output file *example.csv*. This time, four parallel instances are allocated for metric calculation (parallel_calculations set to 4). Finally, lines 47–48 include the main guard that is necessary when parallel instances are used.

Executing the example presented in Listing 1 means in practice that the framework performs a total of 176 encoding runs using two different encoders, 22 sequences, and up to four parallel instances. The whole usage scenario can be written in less than 50 lines of Python code.

5 Extensibility

The framework is designed for easy extensibility. The full documentation for adding new encoders, build systems, and using analysis tools is included in the repository. Adding new encoders requires defining the encoder, including command-line argument splitting, encoder building, and handling the encoding command. Implementing all these steps takes less than 150 lines of Python code. The inclusion of build systems requires passing the defines and calling the build command. The framework supports analysis tools that calculate metrics by using either native Python code or an external tool.

6 Conclusion

This paper presented the *uvgVenctester* test automation framework for video encoder benchmarking. Unlike the other existing approaches, our solution incorporates all primary testing stages into a single framework, from automated encoder building and execution to reporting calculated rate-distortion-complexity results in human and machine-readable formats. Currently, our framework supports seven state-of-the-art encoders (HM, Kvazaar, x265, VTM, VVenC, SVT-VP9, and SVT-AV1), BD-rate analysis with three quality metrics (PSNR, SSIM, and VMAF), encoding speed measurement, and three distinct output formats (CSV, RD graph, and comparison table). The ultimate goal of this proposal is to motivate video encoder developers to replace their own custom-designed testing environments with this multipurpose testing framework and thereby save development time.

In the future, we plan to 1) extend the set of supported video encoders, quality metrics, and output formats; 2) streamline the procedure of adding new analysis tools to the framework; and 3) speed up testing by making it possible to distribute encoder execution across multiple machines.

ACKNOWLEDGMENTS

This work was supported in part by the AI for situational Awareness (AISA) project led by Nokia and funded by Business Finland. The authors would also like to thank Anton Ihonen for his contribution to the implementation of the framework.

REFERENCES

[1] Cisco Systems. (Dec. 2018). *Cisco Visual Networking Index: Forecast and Trends 2017-2022*. [Online]. Available: http://web.archive.org/web/20181213105003/https:/www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.pdf

[2] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H. 264/AVC video coding standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 560–576, Jul. 2003.

[3] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[4] International Telecommunication Union (ITU). *New Versatile Video Coding Standard to Enable Next-Generation Video Compression*. [Online]. Available: https://www.itu.int/en/mediacentre/Pages/pr13-2020-New-Versatile-Video-coding-standard-video-compression.aspx

[5] A. Grange, P. De Rivaz, and J. Hunt. *VP9 Bitstream & Decoding Process Specification*. [Online]. Available: http://www.webmproject.org/vp9/

[6] *AV1*. [Online]. Available: https://aomedia.googlesource.com/aom

[7] F. Bossen, *Common HM Test Conditions and Software Reference Configurations*, document JCTVC-L1100, Geneva, Switzerland, Jan. 2013.

[8] F. Bossen, J. Boyce, K. Suehring, X. Li, and V. Seregin, *VTM Common Test Conditions and Software Reference Configurations for SDR Video*, document JVET-T2010, Teleconference, Oct. 2020.

[9] Multicoreware Inc. *How to Compare Video Encoders*. [Online]. Available: http://x265.org/compare-video-encoders/

[10] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, Apr. 2004.

[11] Z. Li, A. Aaron, I. Katsavounidis, A. Moorthy, and M. Manohara. *Toward a Practical Perceptual Video Quality Metric*. [Online]. Available: https://netflixtechblog.com/toward-a-practical-perceptual-video-quality-metric-653f208b9652

[12] G. Bjøntegaard, *Calculation of Average PSNR Differences Between RD-Curves*, document VCEG-M33, Austin, Texas, USA, Apr. 2001.

[13] *MSU Video Quality Measurement Tool*. [Online]. Available: https://www.compression.ru/video/quality_measure/video_measurement_tool.html

[14] *VQMT: Video Quality Measurement Tool*. [Online]. Available: https://www.epfl.ch/labs/mmspg/downloads/vqmt/

[15] R. R. R. Rao, S. Goring, P. List, W. Robitza, B. Feiten, U. Wustenhagen, and A. Raake, "Bitstream-based model standard for 4K/UHD: ITU-T P.1204.3—Model details evaluation analysis and open source implementation," in *Proc. Int. Conf. Qual. Multimedia Exper.*, May 2020, pp. 1–6.

[16] *FFMetrics*. [Online]. Available: https://github.com/fifonik/FFMetrics

[17] *FFmpeg*. [Online]. Available: https://ffmpeg.org

[18] *HEVC Reference Software Version 16.20*. [Online]. Available: https://vcgit.hhi.fraunhofer.de/jct-vc/HM/-/tags/HM-16.20

[19] A. Lemmetti, M. Viitanen, A. Mercat, and J. Vanne, "Kvazaar 2.0: fast and efficient open-source HEVC inter encoder," in *Proc. ACM Multimedia Syst. Conf.*, Istanbul, Turkey, June 2020, pp. 237–242.

[20] MulticoreWare. *X265 HEVC Encoder / H.265 Video Codec*. [Online]. Available: https://bitbucket.org/multicoreware/x265/downloads

[21] *VVC Reference Software Version 10.0*. [Online]. Available: https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM

[22] *Fraunhofer Versatile Video Encoder (VVenC)*. [Online]. Available: https://github.com/fraunhoferhhi/vvenc.

[23] *Scalable Video Technology for VP9 Encoder (SVT-VP9 Encoder)*. [Online]. Available: https://github.com/OpenVisualCloud/SVT-VP9

[24] *Scalable Video Technology for AV1 (SVT-AV1 Encoder and Decoder)*. [Online]. Available: https://github.com/AOMediaCodec/SVT-AV1

[25] A. Mercat, M. Viitanen, and J. Vanne, "UVG dataset: 50/120fps 4K sequences for video codec analysis and development," in *Proc. ACM Multimedia Syst. Conf.*, Istanbul, Turkey, June 2020, pp. 297–302.

[26] Xiph.org. *Video Test Media*. [Online]. Available: https://media.xiph.org/video/derf/

[27] SJTU Media Lab. *SJTU 4K Video Sequences*. [Online]. Available: http://medialab.sjtu.edu.cn/web4k/index.html

[28] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, and R. Kern, "Array programming with numpy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sept. 2020.

[29] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Comput. Sci. Eng.*, vol. 9, no. 3, pp. 90–95, May 2007.

[30] *CMake*. [Online]. Available: https://cmake.org/

[31] *Automake*. [Online]. Available: https://www.gnu.org/software/automake/

[32] *Matplotlib*. [Online]. Available: https://matplotlib.org/

[33] *Wkhtmltopdf*. [Online]. Available: https://wkhtmltopdf.org/