JPEG AI: From Paper to Practice – Standardization and Mobile Implementation

João Ascenso, Elena Alshina and Alexander Karabutov

VCIP, Klagenfurt, Austria, December 2025

Speakers: João Ascenso

- Professor at University of Lisbon, Portugal
- Researcher at Instituto de Telecomunicações, Lisbon, Portugal
- Very active in standardization, JPEG AI ad-hoc group chair, CPM sub-group chair
- More than 150 publications in international journals and conferences
- More than 5300 citations with h-index of 34
- Past associate editor of IEEE Transactions on Multimedia, IEEE Transactions on Image Processing, IEEE Signal Processing Letters, etc.
- Technical program chair of well-known international conferences, such as PCS 2022, EUVIP 2022
- Elected member of the IEEE Multimedia Signal Processing Technical Committee
- Best Paper Award at PCS 2015, ICME 2020, MMSP 2024
- Several Excellence Teaching Awards
- His current research interests include visual coding, quality assessment, coding and processing of 3D visual representations, coding for machines, super-resolution, denoising among others.

Speakers: Elena Alshina

- ***Education***:
  - *Master* of Physics Moscow Sate University (1995)
  - *PhD* in Computer Science and Mathematical Modelling (1998)
- ***Carrier***:
  - *Senior Researcher* Russian Academy Of Science (Institute for mathematical Modelling) 1998~2006
  - *Associate Professor* National Research University of Electronic Technology 2000~2006
  - *Principal Engineer* Samsung Electronics 2006 (Moscow), 2007~2018 (Suwon/Seoul)
  - *Chief Video Scientist* Huawei Technologies (Munich) 2018-present
- ***Positions***:
  - *Huawei*: Audiovisual Technology Lab Director; Media Coding Technology Lab Director
  - *JVET*: Neural Network Video Compression AhG co-chair, exploration experiment coordinator
  - *JPEG*: JPEG AI standardization project **chair and editor** (along with Prof. João Ascenso)

- *Mathematical modelling*
- *Video & Image compression & processing*
- *Neural network based algorithms*
- *HEVC/H.265, VVC/H.266, JPEG AI*

Speakers: Alexander Karabutov

- ***Education***:
  - *Master* of Physics Moscow Sate University (2009)
  - *PhD* in Acoustics (2013)
- ***Carrier***:
  - *Researcher* in Russian Academy Of Science (Institute on Laser and Informational Technologies) 2012~2014
  - *Chief Engineer* in the 360 degree camera development company (Panorics, Moscow) 2014~2016
  - *Senior Video Coding Engineer* Huawei Technologies (Moscow), 2016~2022
  - *Principal Engineer* Huawei Technologies (Munich) 2022-present
- ***Positions***:
  - *JVET*: Neural Network Video Compression AhG, a member of SW coordination team
  - *JPEG*: JPEG AI standardization project, SW coordinator

- *Video & Image compression & processing*
- *Neural network based algorithms*
- *MPEG 5, VVC/H.266, JPEG AI*

Outline:

- Context and motivation
- Introduction to the JPEG AI project
  - Scope
  - Use cases
  - Framework
  - Requirements
- Subjective Assessment of Learning-based Coding Solutions
  - Methodology
- Objective Assessment of Learning-based Coding Solutions
  - Objective quality metrics
  - Complexity metrics
  - Training and test sets
- Compressed Domain Processing

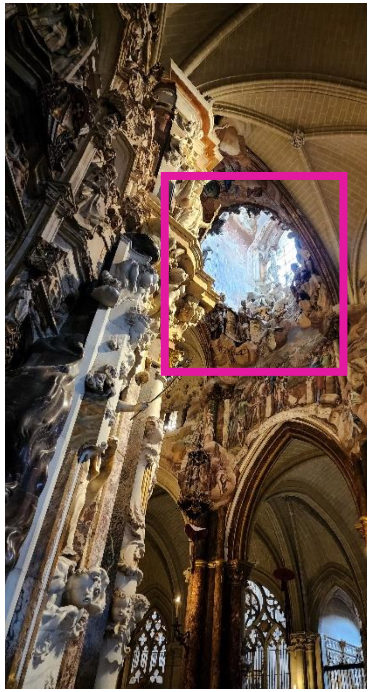
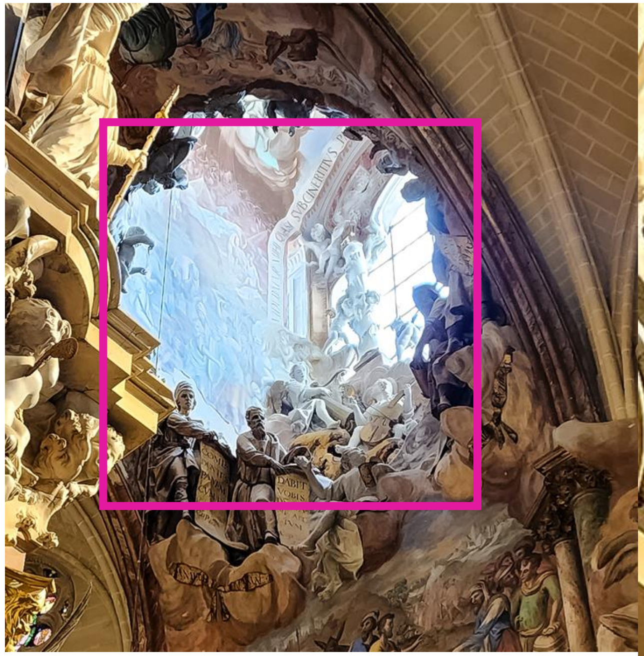
Outline:

- JPEG AI design principles
- JPEG AI encoder and decoder architecture
- JPEG AI basic codec elements:
  - Streams structure and partitioning
  - Multi-thread arithmetic coder: me-tANS
  - Rate control
  - Entropy network with bit-exact behavior
  - Residual coding
  - Latent domain prediction & Context modeling
  - Analysis & synthesis transform
  - Attention modules and transformers
  - Progressive decoding
- JPEG AI conformance
- JPEG AI demo and performance
  - Objective performance results
  - Visual quality analysis
- JPEG AI verification model and reference software
- JPEG AI and intra coding in video

Context and Motivation

Rich Ecosystem for Image Technologies

Image/Video Use Cases are Rather Different !


Zooming in, go for details

Image Compression Standards

JPEG

JPEG 2000

JPEG-XS

GIF

PNG

JPEG XL

AV1

HEVC

webp

H.266 VVC

Landscape of Image/Video Codec

Image codec

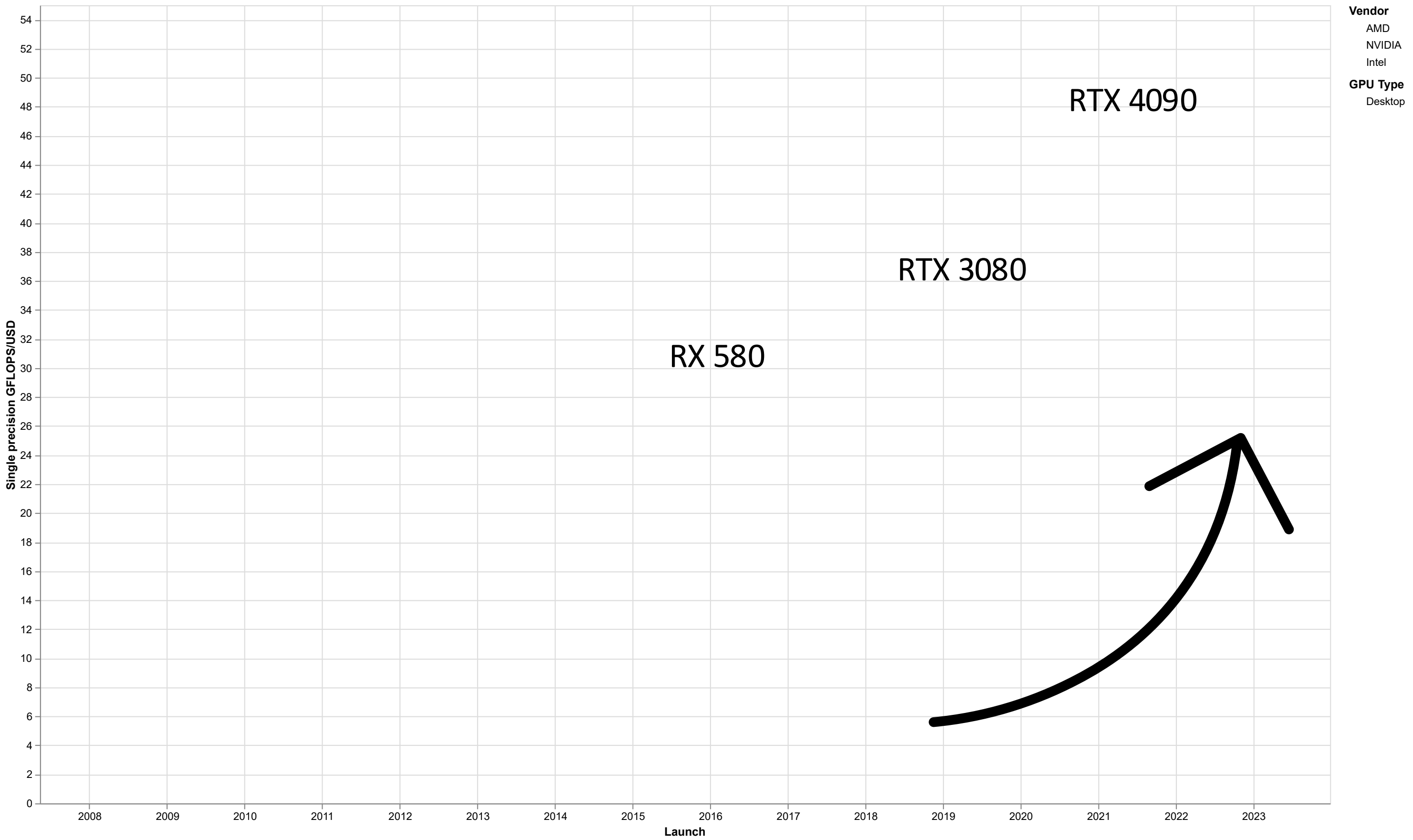
1996	2017	2019	2024
The mostly used Image codec	Default in all Apple devices since 2017	AOM developed	Already here
JPEG	HEIF	AVIF	JPEG AI

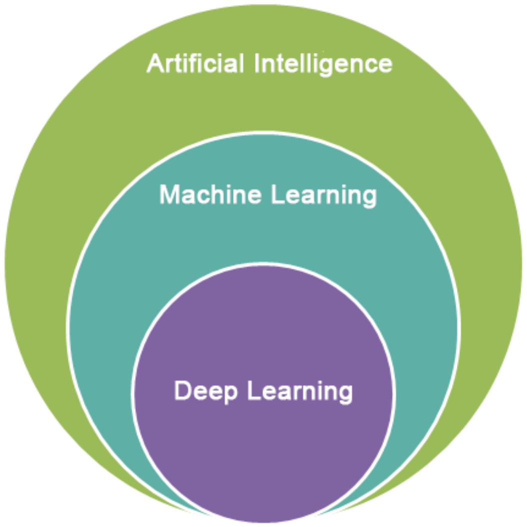
Video codec

2003	2013	2018	2020
H.264 MPEG-4/AVC	H.265 HEVC	AV1	H.266 VCC Versatile Video Coding
Enabled YouTube	Enabled 4K video services		The most recent ISO/ITU standard

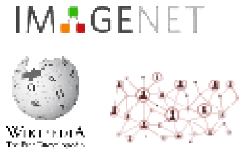
Deep Learning Explosion

Giga FLoating-point Operations Per Second that you can buy with 1 USD

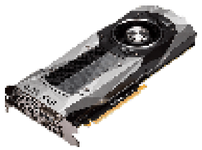




1. Big Data
- Larger Datasets
- Easier Collection & Storage



2. Hardware
- Graphics Processing Units (GPUs)
- Massively Parallelizable



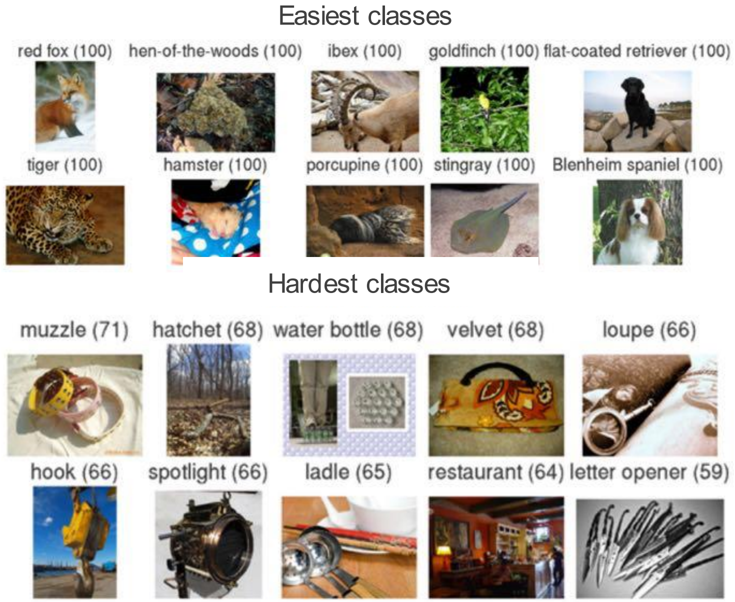
3. Software
- Improved Techniques
- New Models
- Toolboxes

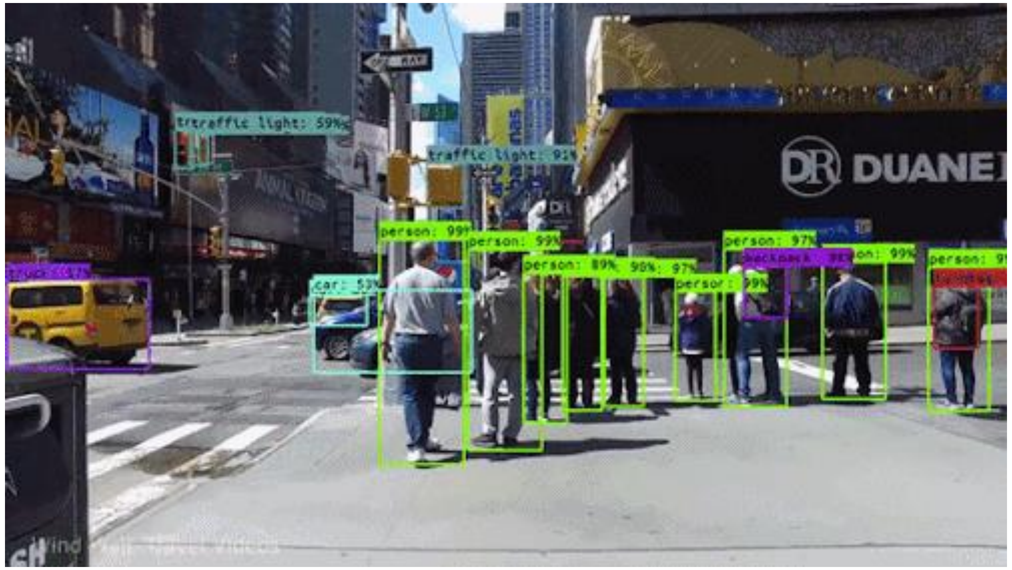
TensorFlow

Deep Learning Achievements: Computer Vision

- Extremely successful in computer vision tasks:
  - Image classification, object detection, semantic segmentation, …
  - Face recognition, image generation, video understanding, …

Image classification





Deep Learning Achievements: Image Processing

- Extremely successful in image processing tasks:
  - Denoising, super-resolution, inpainting, style transfer, segmentation, …
  - Many other image restoration tasks (dehazing, deraining, etc.), …

Mordvinsev et al, 2015
Gatys et al, 2016

And Many More…

A white teddy bear sitting in the grass

A man in a baseball uniform throwing a ball

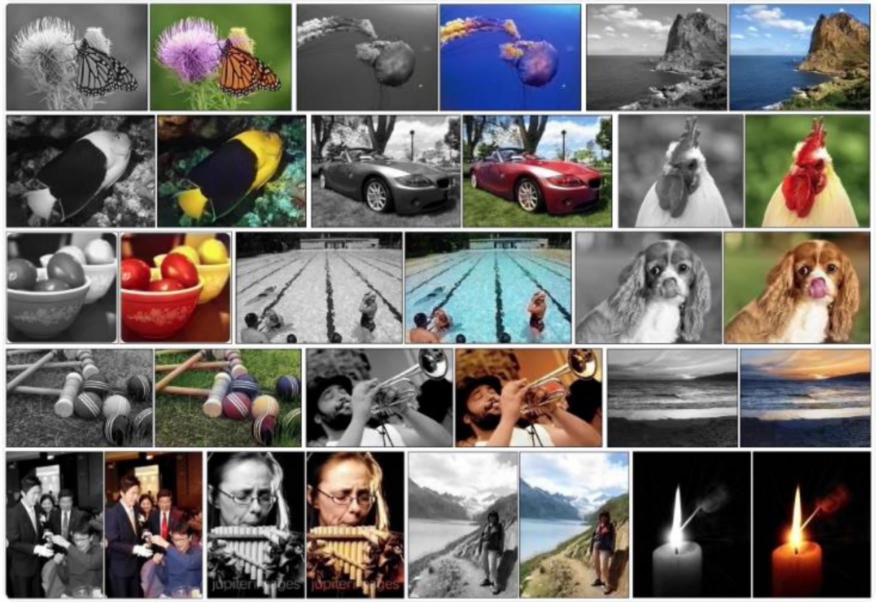
A woman is holding a cat in her hand

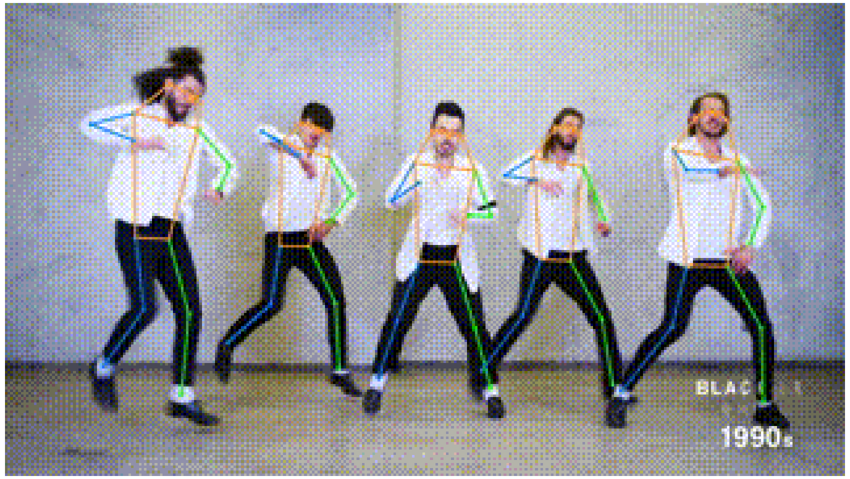
5

A man riding a wave on top of a surfboard

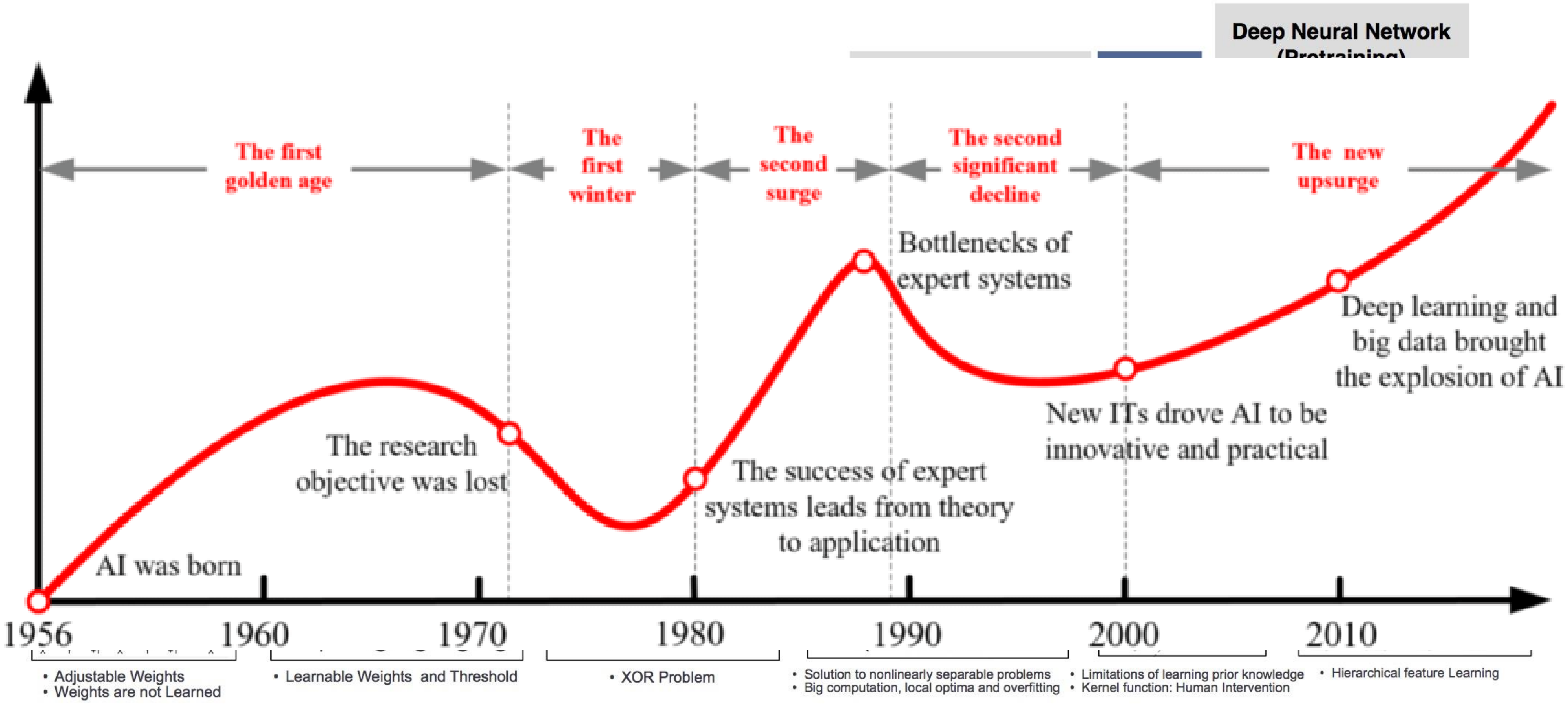
A cat sitting on a suitcase on the floor

A woman standing on a beach holding a surfboard





History of Neural Network Hype



Neural Networks MATCH Compression

- **Learning-based image compression**
  - **Non-linear transformations, entropy coding models, etc.**
- Learning-based video compression
  - Optical flow, motion compensation, multi-frame fusion, etc.
- Models for typical image/video compression modules
  - Intra-prediction, in/out loop-filtering, entire encoder, etc.
- Learning-based point cloud compression
  - Geometry and attribute compression methods, etc.
- Learning-based light-field compression
  - Stereoscopic and multi-view representations, NeRF, etc.
- Neural networks models and feature compression
  - Enabling the efficient transmission of large models (or feature)

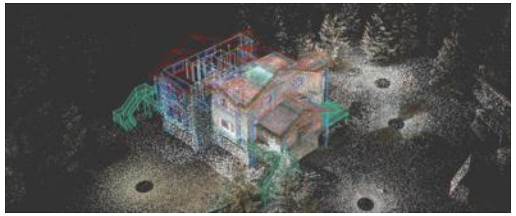
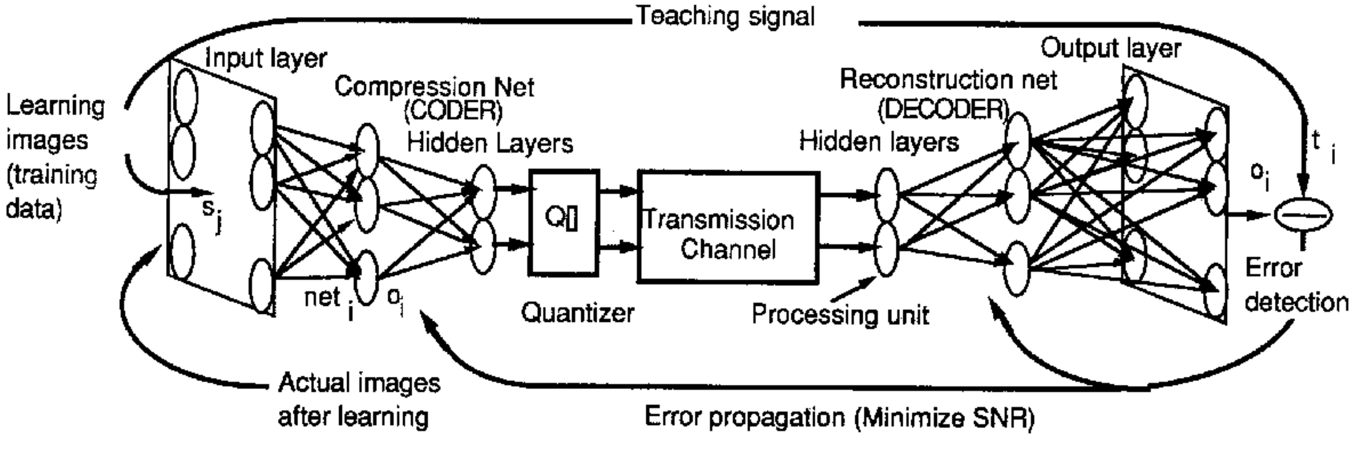


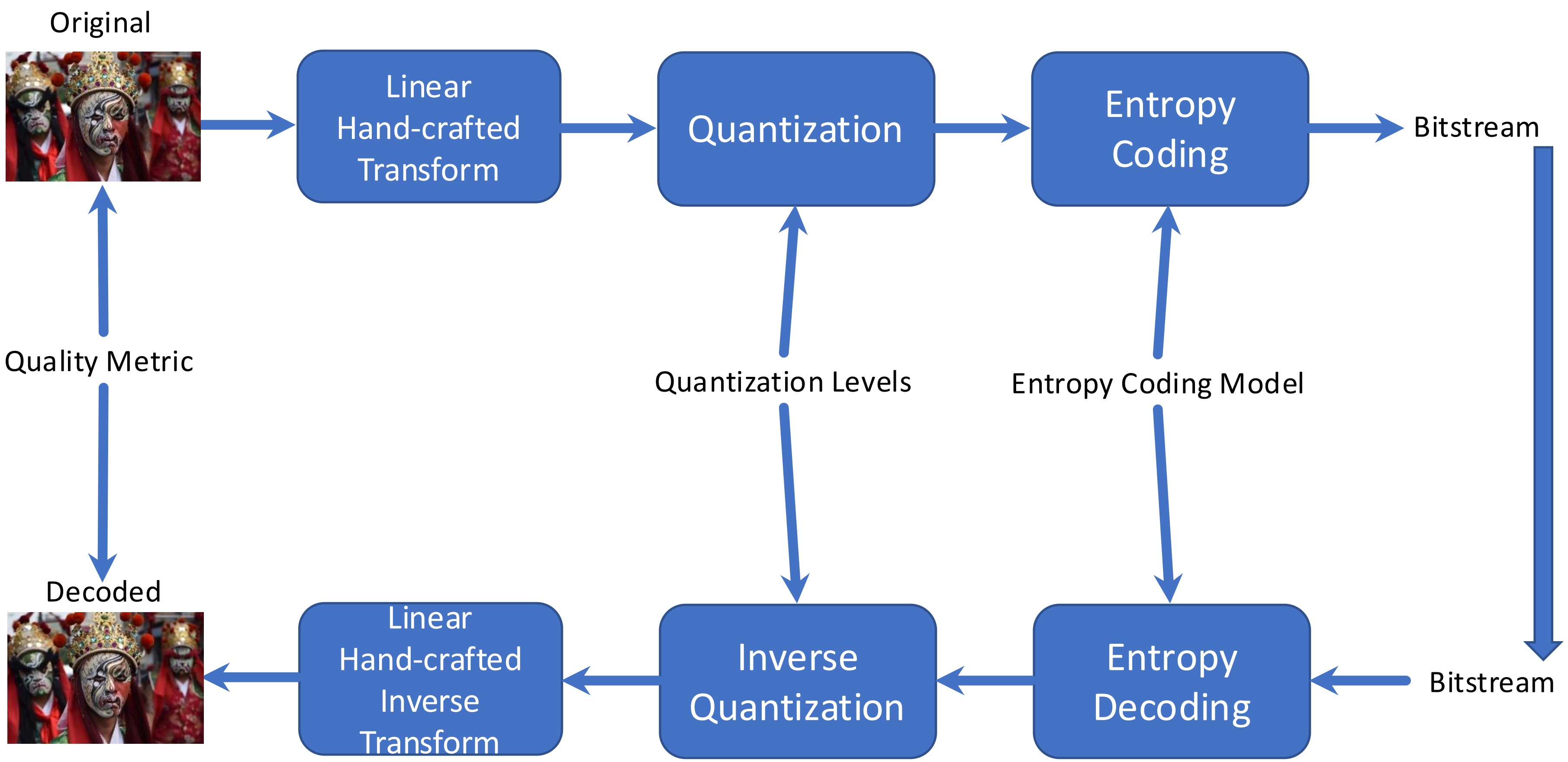
Image Compression with Neural Networks

- Very ~~recent~~ and promising field
  - N. Sonehara, M. Kawato, S. Miyake, K. Nakane, Image data compression using neural network model, Proceedings of the International Joint Conference On Neural Networks, Washington DC, **1989**, pp. 35–41.
  - G.L. Sicurana, G. Ramponi, Artificial neural network for image compression, Electron. Lett. 26, (7) (**1990**) 477–479.

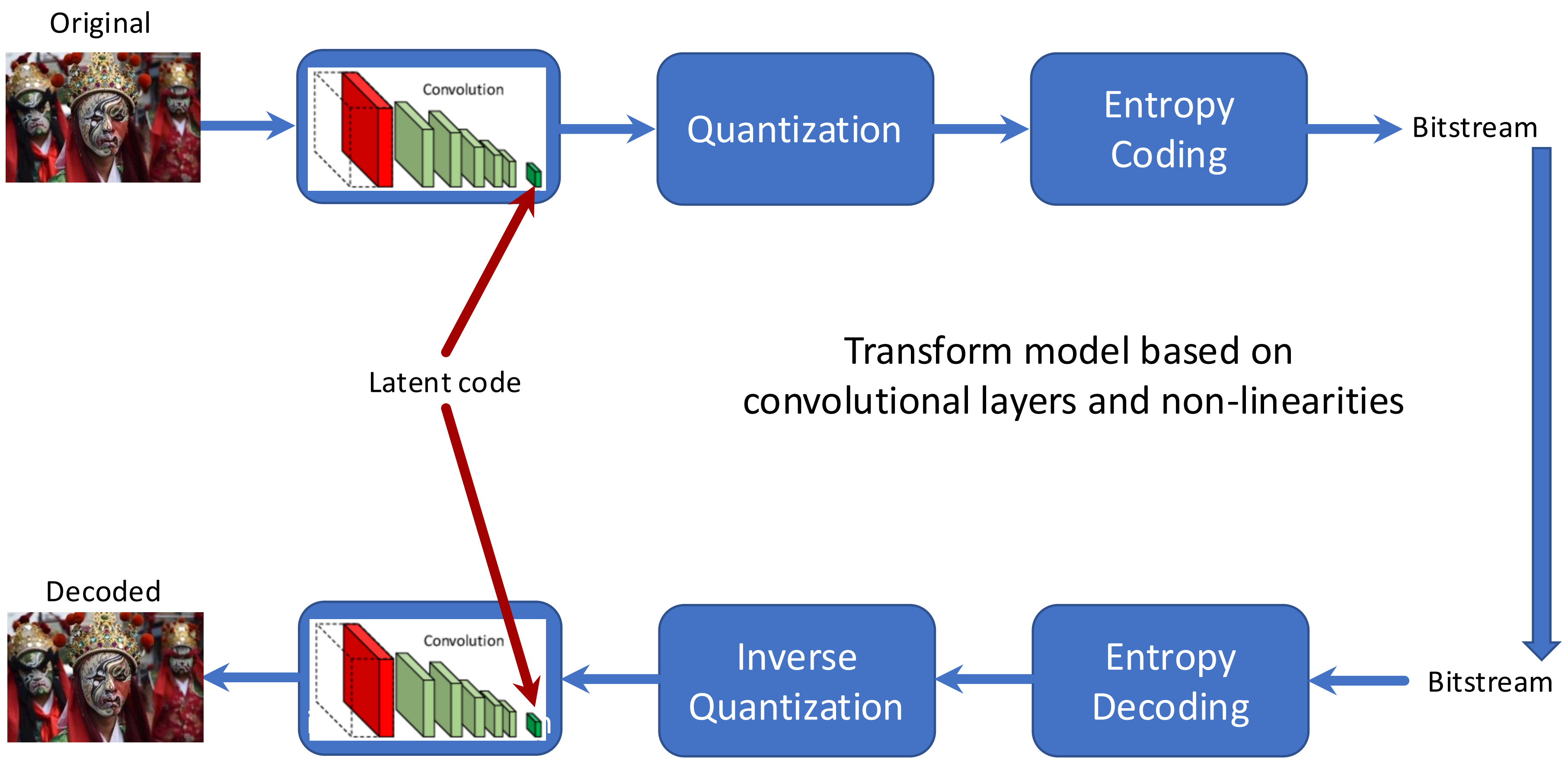


As old as JPEG !!!

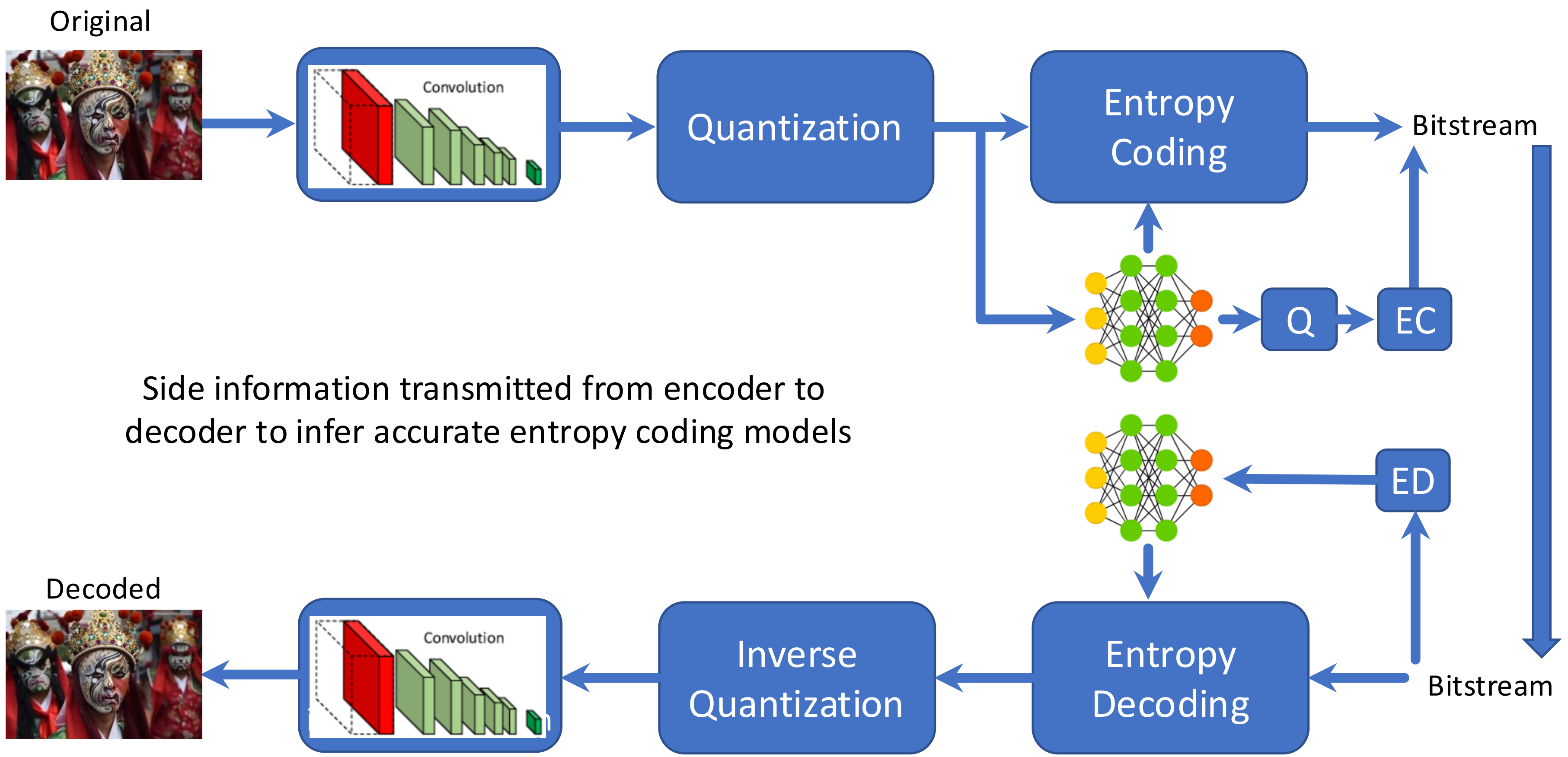
Classical Image Compression Pipeline



Learning-based Image Compression Pipeline



Learning-based Image Compression Pipeline



Introduction to the JPEG AI Standard

IEEE ICIP, Abu Dhabi, October 2024

JPEG Family of Standards

JPEG RF

JPEG AI

Visual Coding

JPEG JPEG | JBIG JBIG2 | JPEG LS | JPEG 2000 | JPEG Pleno

JPEG XR | JPEG XT | JPEG XS | JPEG XL | JPEG AI

Systems

JPEG Trust | JPEG Systems | JPEG JPSearch

Others

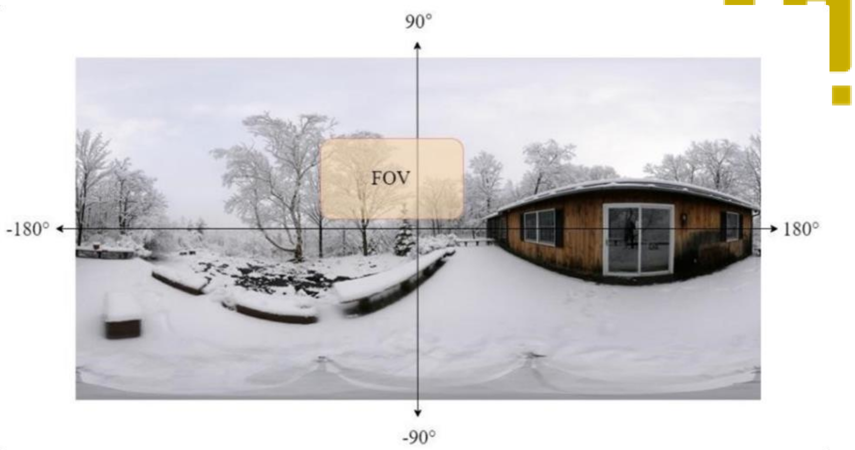
JPEG DNA | JPEG AIC | JPEG XE

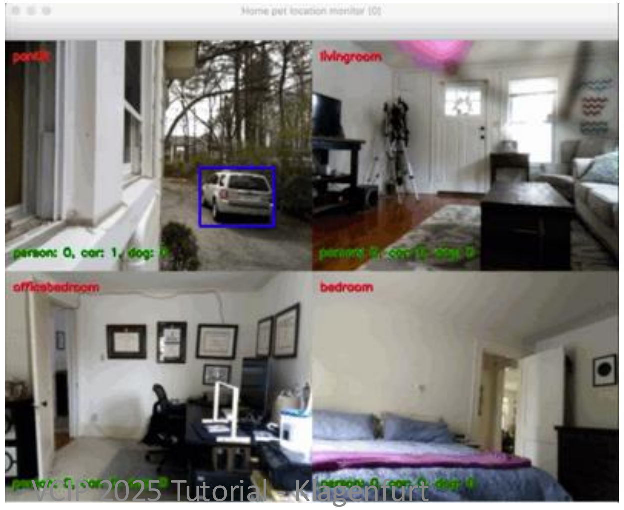
JPEG AI Achievements

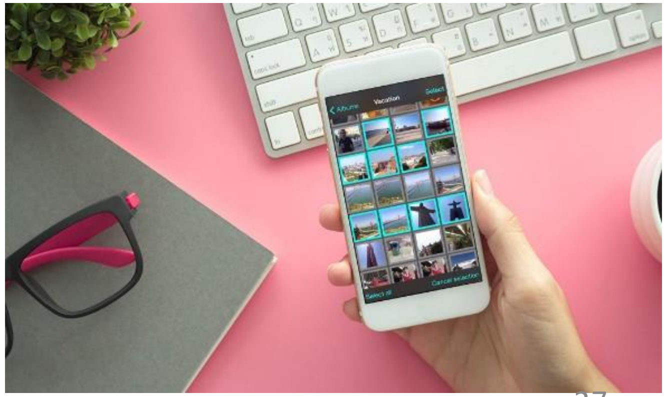
- JPEG AI Project (ISO/IEC 6048) within the JPEG standardization group develops and standardize learning-based image compression
  - Joint standardization effort between SC29/WG1 and ITU-T SG16
  - Active since 2019, International Standard expected in January 2025
- Call for Evidence combined with MMSP Workshop Grand Challenge
  - 6 codecs submitted (out of 8 registered)
- Some relevant public documents:
  - White Paper on JPEG AI Scope and Framework
  - JPEG AI Uses Cases and Requirements
  - JPEG AI Training and Test Conditions
  - JPEG AI Call for Proposals
  - And many more …
- Check for more information: https://jpeg.org/jpegai/

JPEG AI Use Cases

- Cloud storage
- Visual surveillance
- Autonomous vehicles and devices
- Image collection storage and management
- Live monitoring of visual data
- Media distribution
- 360º photo sharing







Application-driven Requirements

- High coding efficiency is important for many applications such as cloud storage or media distribution

- Content understanding is vital for many applications such as visual surveillance, autonomous vehicles, image collection management, etc
  - Objects may need to be recognized
  - Images may need to be classified for organization purposes
  - Actions or events may need to be recognized

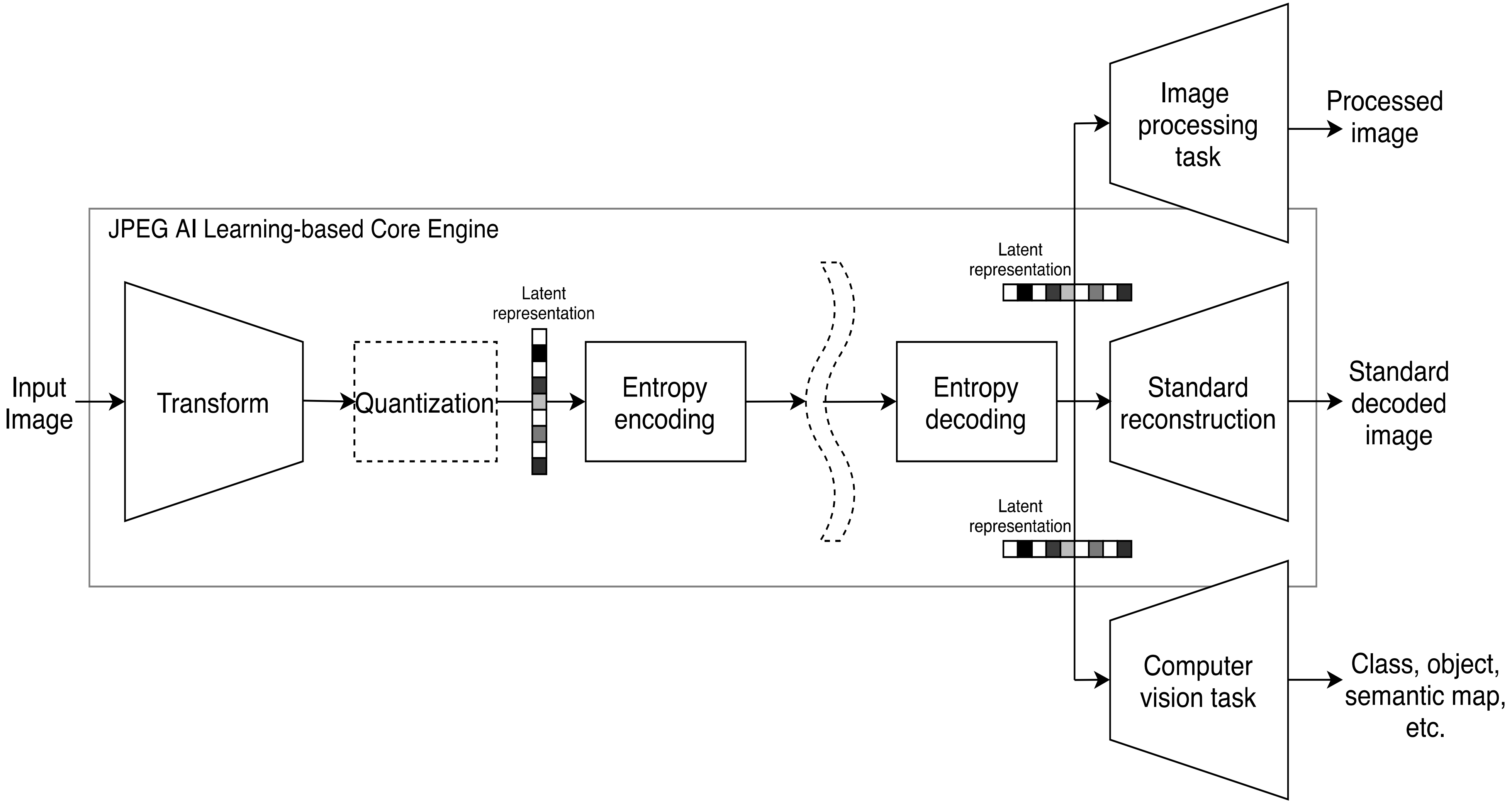
- Content is not consumed by humans in the same way as the original reference in many applications such as in media distribution
  - Noise can be reduced
  - Resolution can be enhanced
  - Colors can be corrected

JPEG AI Scope

The JPEG AI scope is the creation of a learning-based image coding standard offering a single-stream, compact, compressed domain representation, targeting both human visualization, with significant compression efficiency improvement over image coding standards in common use at equivalent subjective quality, as well as effective performance for image processing and computer vision tasks, with the goal of supporting a royalty-free baseline

- Advantages:
  - Same compressed stream is useful for decoding as well as image processing and computer vision tasks
  - Reduces the resources needed to perform image processing and computer vision tasks
  - Allows performing processing and computer vision tasks using features extracted from the original instead of the lossy decoded images

JPEG AI Framework



JPEG AI: Three Pipelines

- Standard JPEG AI decoding
- Image processing tasks:
  - Super-resolution
  - Denoising
  - Low-light enhancement
  - Color correction
  - Exposure compensation
  - Inpainting
- Computer vision tasks:
  - Image retrieval and classification
  - Object detection, recognition and identification
  - Semantic segmentation
  - Event detection and action recognition
  - Face detection and recognition

JPEG AI Core Requirements

- Effective compressed domain image processing and computer vision tasks
- Significant compression efficiency improvement over coding standards in common use at equivalent subjective quality
- Reconstructed images with both high subjective quality and high fidelity as measured by full reference objective quality metrics and double stimulus subjective assessment protocols
- Reconstruction reproducibility, from the same bitstream, if decoders in different platforms (CPU and GPU) provide different decoded images, it should not be greater than around 0.5% of BD-rate
- Hardware platform agnostic, encoder and decoder should be implementable in a wide range of hardware platforms.
- Hardware/software implementation-friendly encoding and decoding (in terms of parallelization, memory, complexity, and power consumption)
- Support for 8- and 10-bit depth
- Support for efficient coding of images with text and graphics
- Support for progressive decoding

JPEG AI Desirable Requirements

- Support for higher bit depth (e.g., 12 to 16-bit integer and floating-point HDR) images
- Support for region of interest-based coding
- Support for progressive decoding up to lossless
- Support for lossless alpha channel/transparency coding
- Support for animated image sequences
- Support for wide color gamut coding
- Support for different color representations
- Support for very low file size image coding (e.g. 64×64 pixel images)
- Support for a low-complexity profile - low encode/decode time even on resource-constrained hardware (e.g., mobile devices)
- Minimal generation loss when lossy compression is applied multiple times

JPEG AI Achievements and Organization

- JPEG AI already addressed all ‘core’ and ‘desirable’ requirements with emphasis on:
  - Compression efficiency for standard reconstruction
  - High fidelity, enabling visually lossless
  - Reproducibility of the reconstructed image in multiple devices
  - Practical implementations across a wide range of devices and platforms on the market
- JPEG AI Version 1 specification includes:
  - Part 1 specifies the normative algorithms required for parsing the codestream, reconstructing the decoded image and the NN weights and parameters
  - Part 2 defines profiles and levels
  - Part 3 specifies the reference software
  - Part 4 specifies requirements for conformance
  - Part 5 specifies the file format
- JPEG AI Version 2 will address/include:
  - JPEG AI requirements not yet addressed in version 1, e.g. related to processing and computer vision tasks
  - Significantly improved solutions for JPEG AI requirements already addressed in Version 1, e.g. compression efficiency

Subjective Assessment of Learning-based Coding Solutions

VCIP, Klagenfurt, Austria, December 2025

Quality of Experience

Quality of Experience (QoE) “is the degree of delight or annoyance of the user of an application or service. It results from the fulfillment of his or her expectations with respect to the utility and/or enjoyment of the application or service in the light of the user’s personality and current state”

User Quality: Mostly Signal Fidelity

- Subjective Evaluation

- Objective Evaluation

$$\mathrm{PSNR} = 10 \times \lg\left(\frac{255^2}{\mathrm{MSE}}\right)$$

$$\mathrm{MSE} = \frac{1}{M \times N}\sum_{i=1}^{N}\sum_{j=1}^{M}\left[I(i,j) - I'(i,j)\right]^2$$

Subjective Quality Assessment

- Subjective quality tests are psychophysical experiments in which a number of viewers rate a given set of stimuli
  - How quality is perceived by humans !?

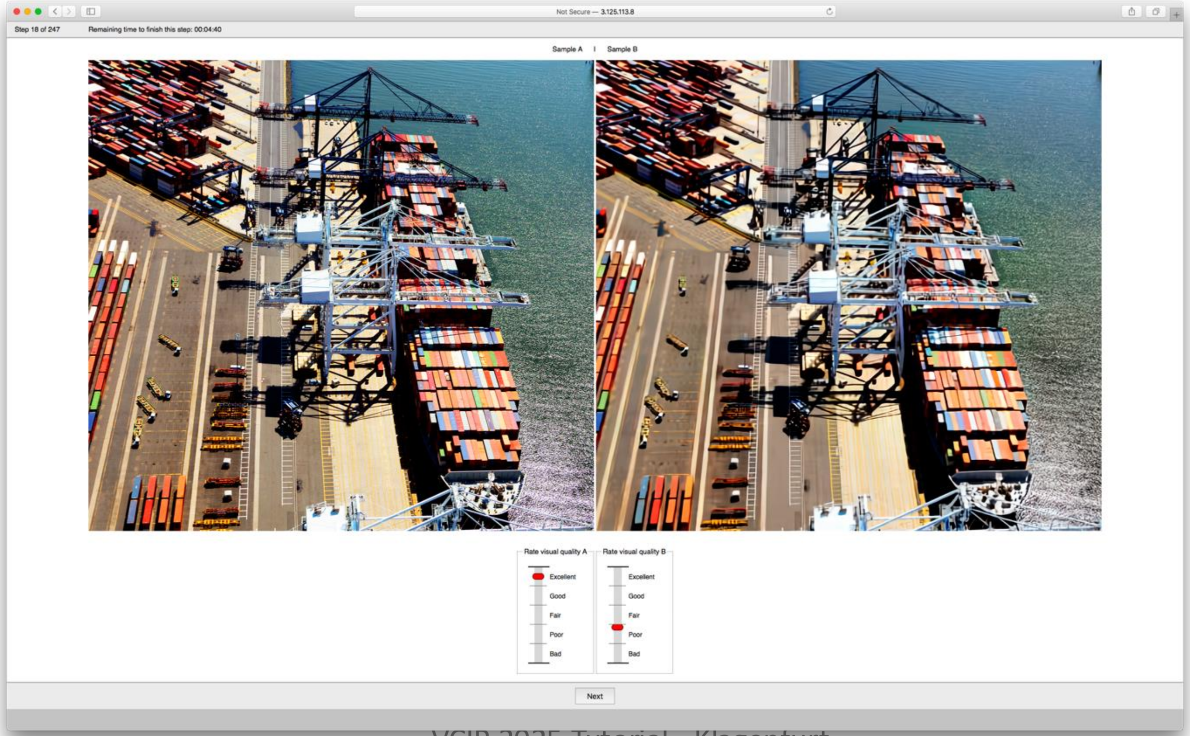
- Subjective tests produce **User Opinion Scores** as a delicate mixture of ingredients and choices:
  - Test material
  - Test environment
  - Test subjects
  - Test methodology
  - Test conditions
  - Data analysis

Subjective Assessment Study

- Subjective evaluation was performed following a crowdsourcing approach
  - Platform: Amazon Mechanical Turk and QualityCrowd2.1 software
  - Requirement: monitor with 1920x1080px resolution or higher, HiDPI/Retina mode disabled.
- DSCQS (ITU Rec. BT-500) methodology:
  - Reference and the impaired stimuli are shown side by side in randomized order
  - Both reference and impaired stimuli quality are assessed by subjects
  - Difference between these two scores is then used to quantify changes in quality
- Total number of the images to assess in the experiment: 416
  - Three sessions with 142, 142 and 141 images
- AWS Mechanical Turk service was used to get subjects
  - 2 anchors: HEVC Intra, VVC Intra JPEG 2000 visually optimized
  - 8 test images: fixed set of crops from the original and decoded images
  - 4 bitrate points covering a wide range of qualities, selected by expert viewing
  - Over 100 naïve subjects

DSCQS Grading Scale

- Vertical scales provide a continuous rating system that avoids quantizing errors
- Scales are divided into five equal lengths which correspond to the normal ITU-R five-point quality scale



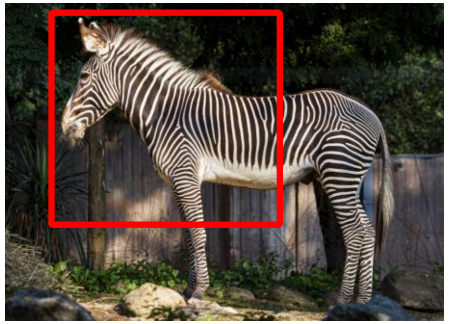
Selected JPEG AI Test Images

00001

00007

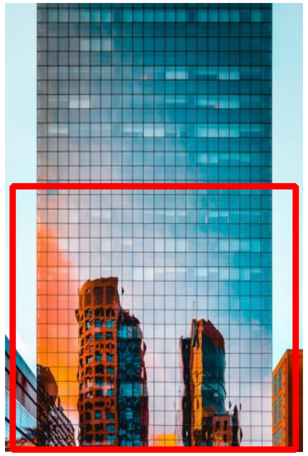
00009

00011



00012

00015

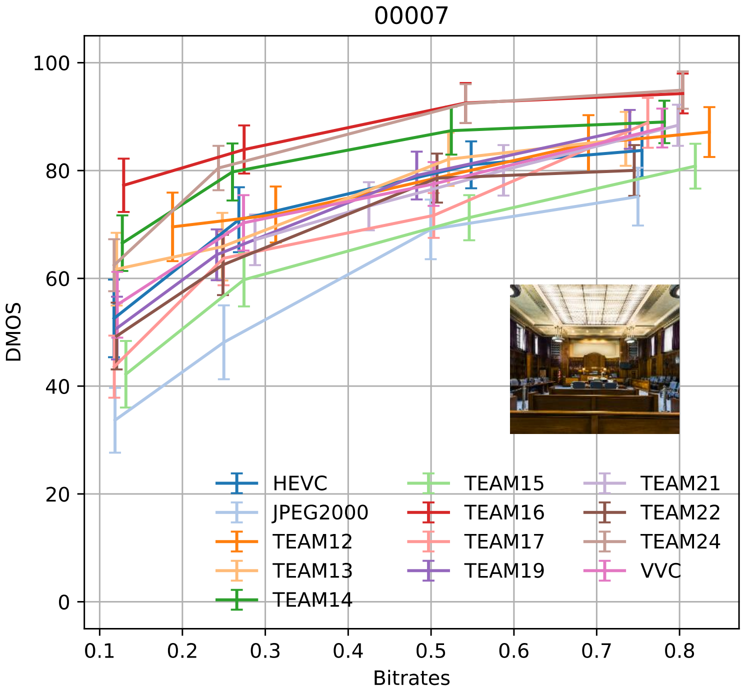


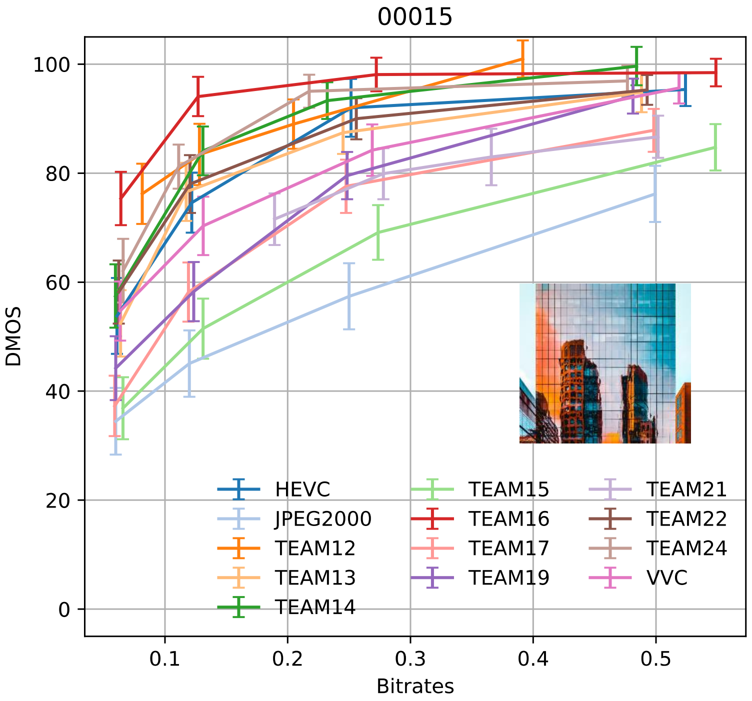
00019

00020

Call for Proposals Subjective Assessment Results

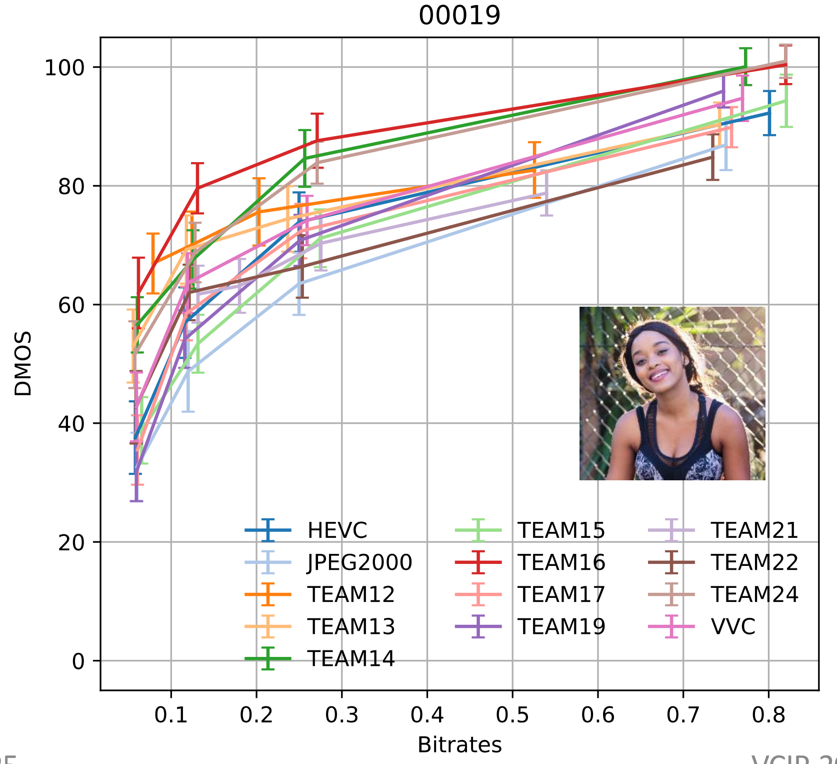
- For DMOS=80, ~ 60% of rate reduction can be observed for the best team in comparison to VVC Intra
- Very high qualities can be reached
- 4 teams have clearly better performance than VVC Intra

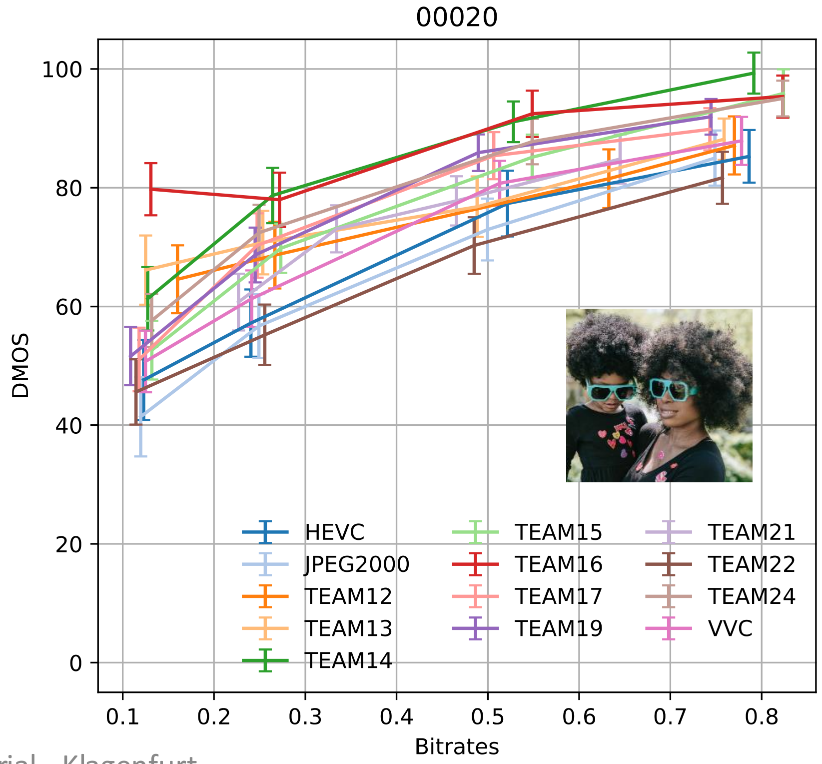




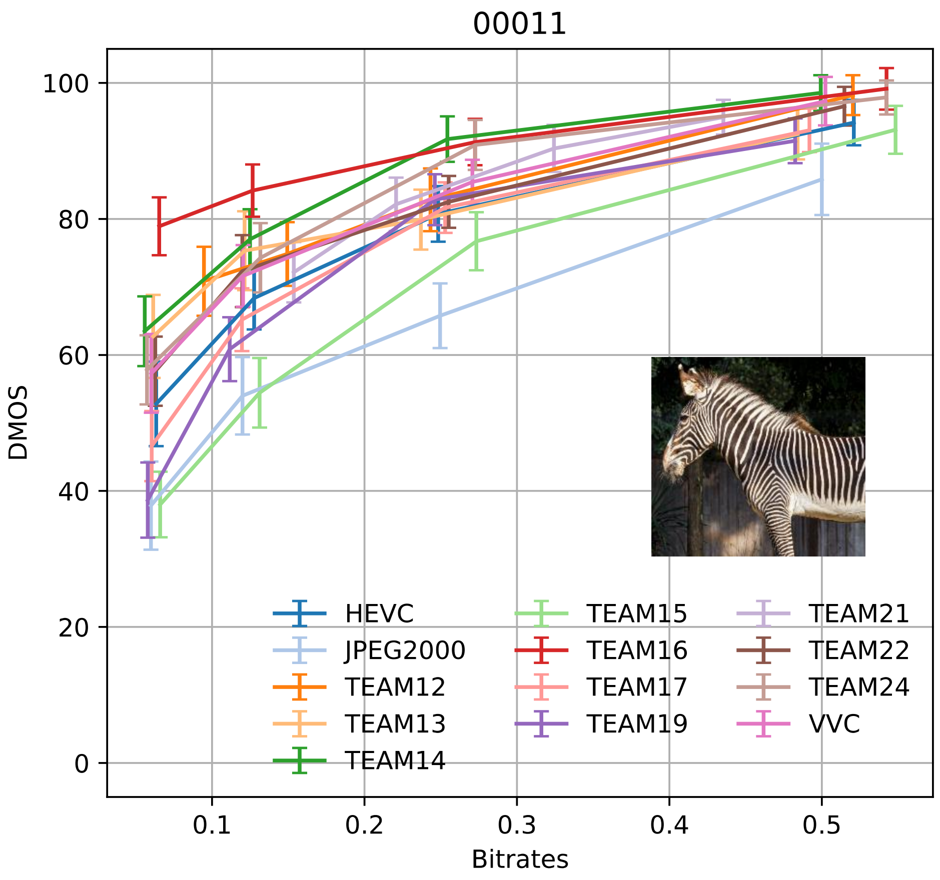
Call for Proposals Subjective Assessment Results

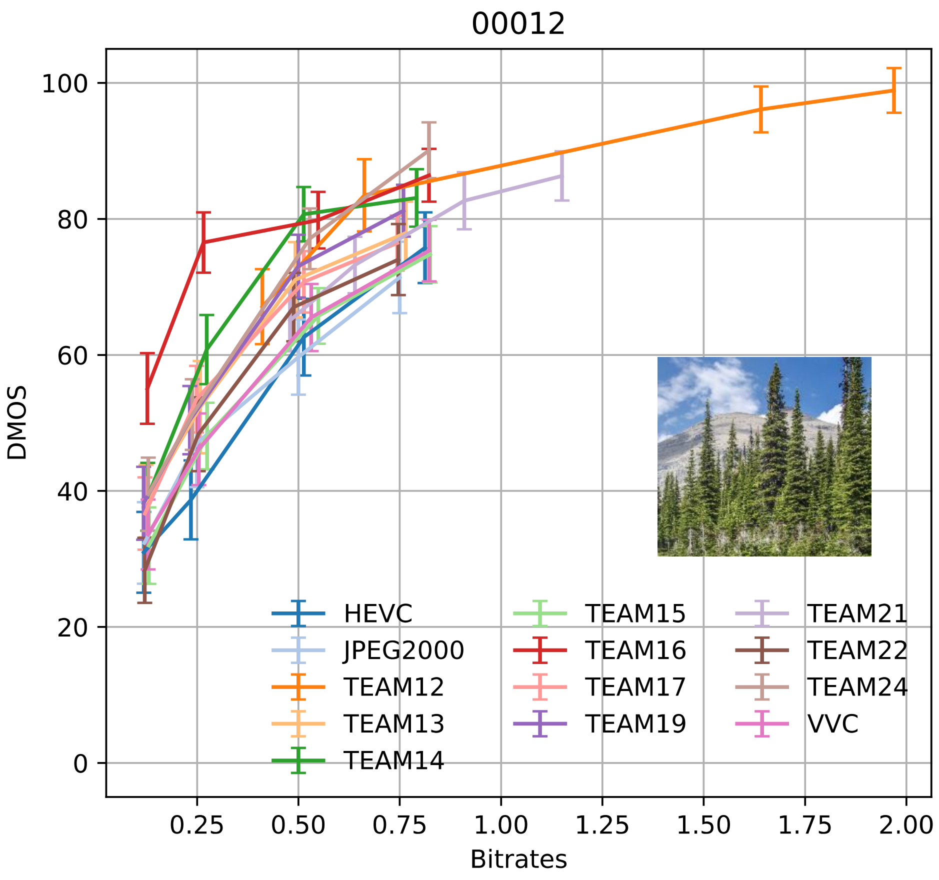
- For DMOS=80, ~ 60%(19) and 40% (20) of rate reduction can be observed for the best team in comparison to VVC Intra
- TEAM16, TEAM14 and TEAM24 have consistently better performance



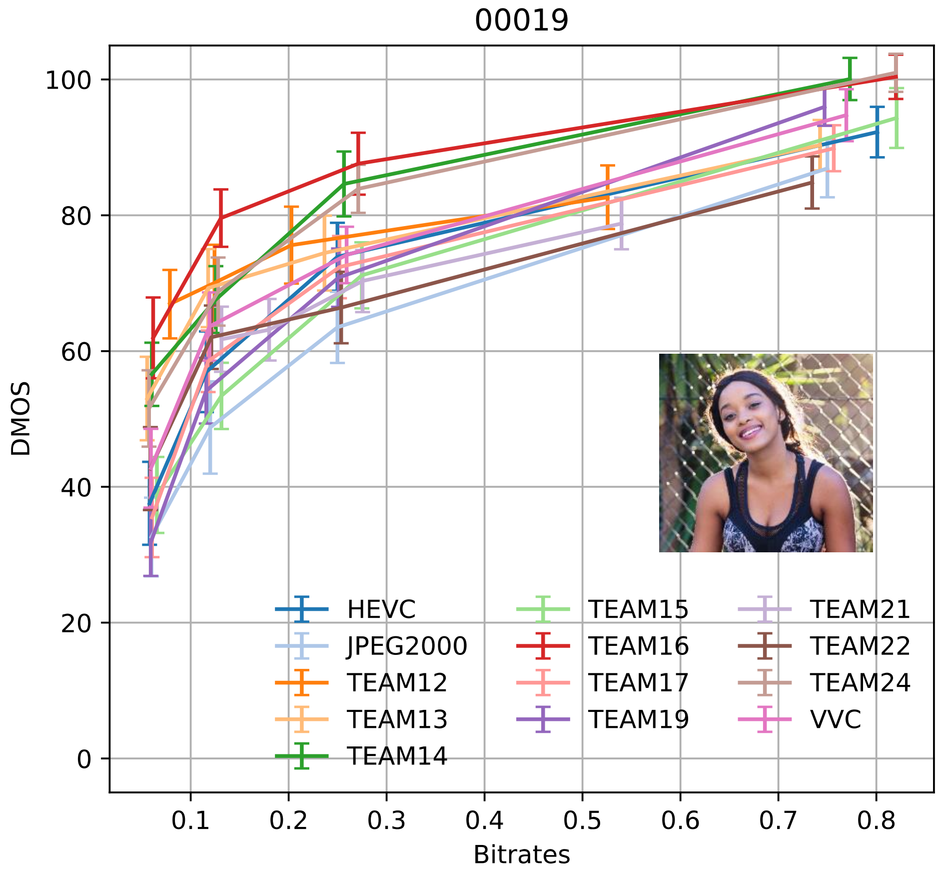


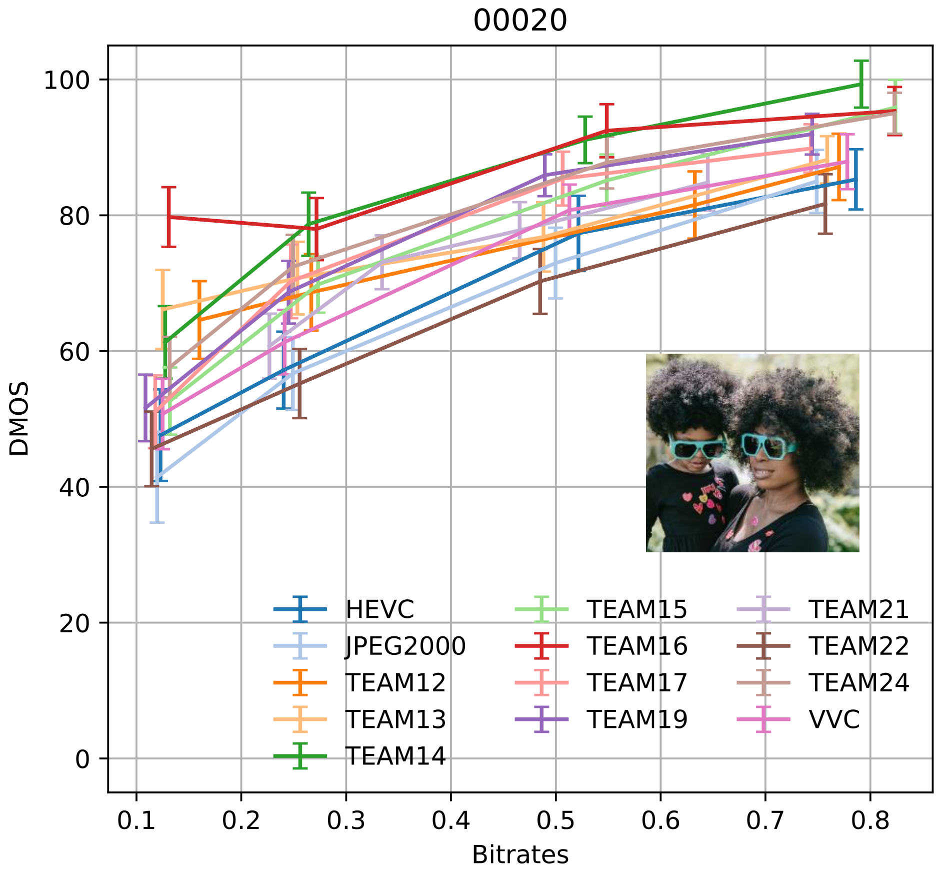
Call for Proposals Subjective Assessment Results





Call for Proposals Subjective Assessment Results





Objective Assessment of Learning-based Coding Solutions

VCIP, Klagenfurt, Austria, December 2025

Many, Many, Many Objective Quality Metrics …

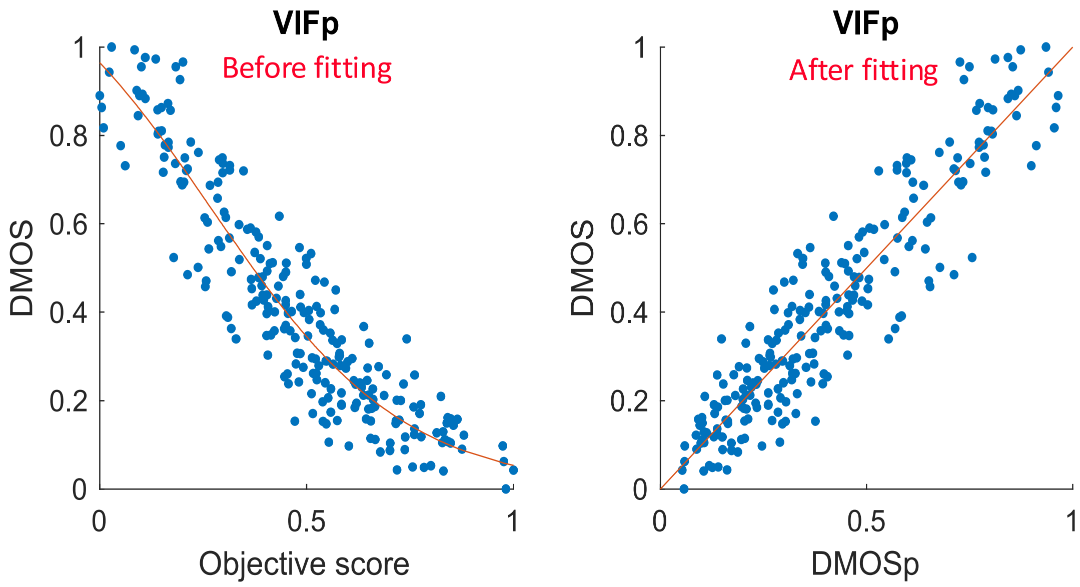
- Learning-based image codecs produce a set of artifacts much different from conventional image codecs
  - Contrast changes, color degradation, etc..
  - Significantly influenced by the loss function

Metric	Color Space	Short Description
CIEDE2000	*Lab*	*Difference between two colors in the CIELab color space.*
FID	*RGB*	*Designed to assess the quality of images generated with GANs.*
FSIM	*RGB*	*Assesses the quality through the Phase Congruency and Gradient Magnitude.*
IW-SSIM	*Y*	*Weights the SSIM by the amount of local information.*
LPIPS	*RGB*	*Measures the perceptual similarity by using deep neural network activations.*
MS-SSIM	*Y*	*Assesses the SSIM at multiple resolutions and viewing conditions*
NLPD	*Y*	*Decomposes images using the Laplacian pyramid*
PSNR	*Y*	*Measures the mathematical dissimilarity through the MSE.*
PSNR-HVS-M	*Y*	*A more advanced version of the PSNR, computed in the DCT domain.*
SSIM	*Y*	*Inspired by the HVS, combine luminance, contrast and structure of the image.*
VDP2	*RGB*	*Designed to be robust to different lighting conditions.*
VIFp	*Y*	*Measures the loss of information between a reference and a distorted image.*
VMAF	*YUV*	*The intra-frame quality estimation fuses VIFp with the Detail Loss Metric.*
WaDIQaM	*RGB*	*Metric based on an end-to-end deep neural network.*

Quality Metrics Performance Assessment

- Correlation measures:
  - Pearson Linear Correlation Coefficient (PLCC)
  - Spearman's Rank Correlation Coefficient (SROCC)
  - Kendall's Rank Correlation Coefficient (KROCC)
  - Root Mean Square Error (RMSE)
  - Outlier Ratio (OR)

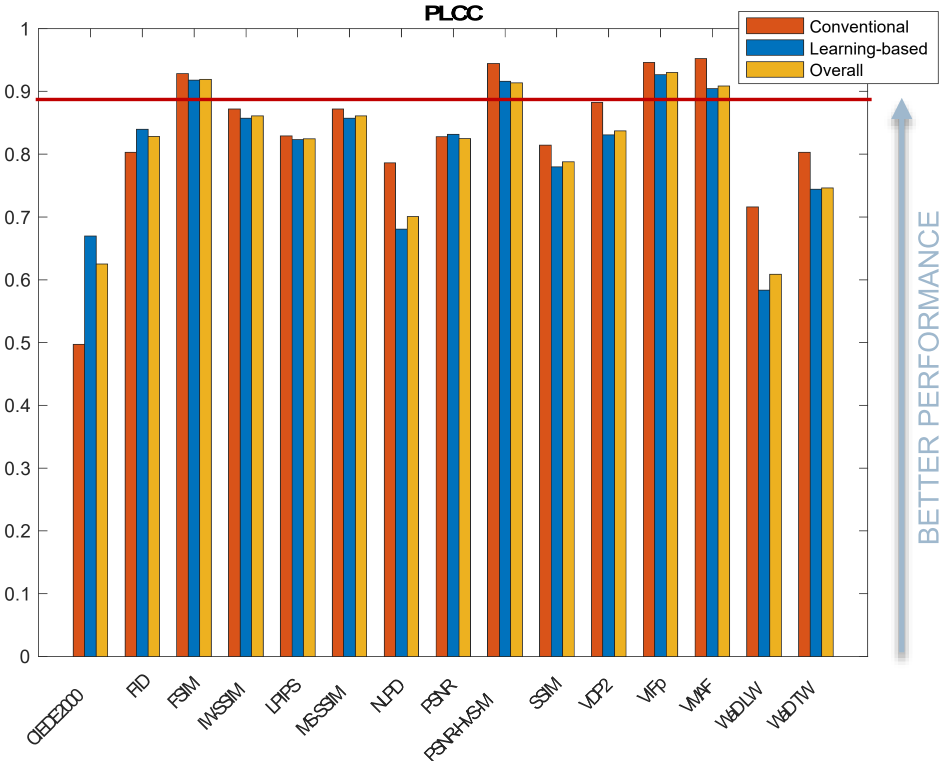
- Logistic fitting with least-squares regression is usually applied

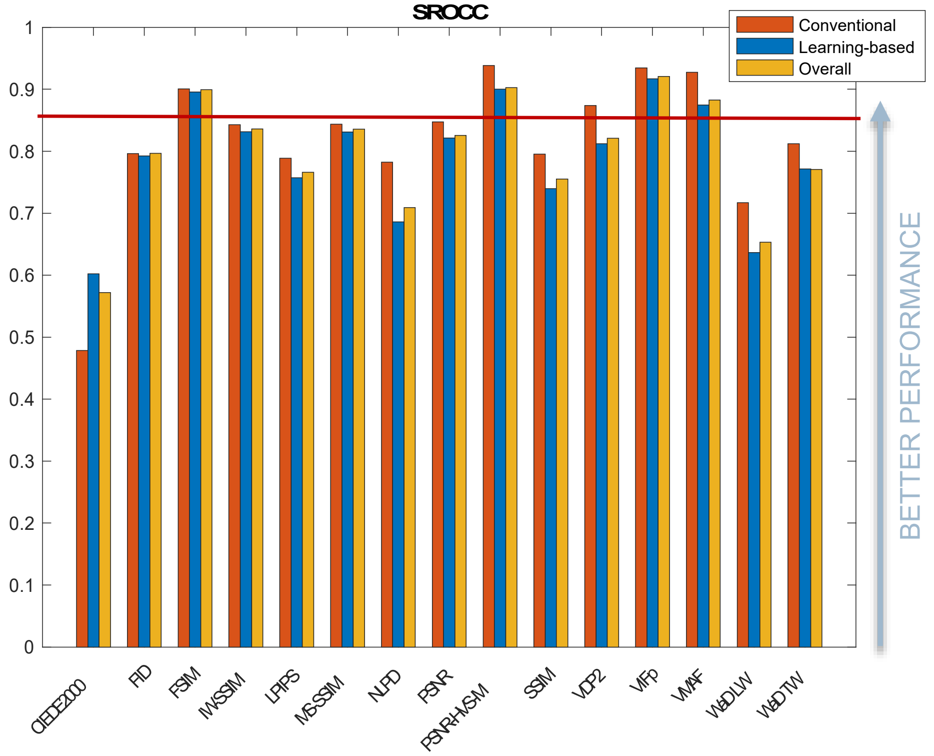


$$DMOS_p = \frac{\beta_1}{1 + \exp(-\beta_2 * (IQS - \beta_3))}$$

Subjective-Objective Correlation

- Using as reference, quality scores obtained from a subjective assessment test





Final Analysis

Metrics	PLCC			SROCC			KROCC		
	Conventional	*LB*	*Overall*	*Conventional*	*LB*	*Overall*	*Conventional*	*LB*	*Overall*
CIEDE2000	0.4973	0.6698	0.6251	0.4785	0.6021	0.5720	0.3373	0.4409	0.4143
FID	0.8029	0.8395	0.8281	0.7964	0.7922	0.7967	0.6012	0.5999	0.6001
FSIM	**0.9283**	**0.9177**	**0.9192**	**0.9004**	**0.8955**	**0.8992**	**0.7341**	**0.7177**	**0.7204**
IW-SSIM	0.8719	0.8574	0.8611	0.8427	0.8314	0.8359	0.6637	0.6428	0.6487
LPIPS	0.8292	0.8232	0.8243	0.7888	0.7570	0.7660	0.6011	0.5773	0.5850
MS-SSIM	0.8719	0.8574	0.8611	0.8427	0.8314	0.8359	0.6637	0.6428	0.6487
NLPD	0.7861	0.6807	0.7007	0.7825	0.6861	0.7090	0.5784	0.4962	0.5147
PSNR	0.8473	0.8062	0.8080	0.8697	0.8070	0.8168	0.6766	0.6123	0.6194

- Quality metrics with higher performance are **VIFp**, **VMAF**, **FSIM** and **PSNR-HSV-M**
- PSNR, WaDIQaM and CIEDE2000 have the lowest performance
- Quality metrics show show a **lower correlation for learning-based codecs** compared to conventional codecs

IW-SSIM	0.1243	0.1289	0.1276	0.6562	0.7273	0.7125
LPIPS	0.1418	0.1422	0.1420	0.7969	0.7443	0.7583
MS-SSIM	0.1242	0.1290	0.1276	0.6562	0.7273	0.7125
NLPD	0.1569	0.1835	0.1790	0.8125	0.7500	0.7583
PSNR	0.1424	0.1392	0.1418	0.7187	0.7159	0.7208
PSNR-HVS-M	**0.0834**	**0.1004**	**0.1020**	**0.5312**	**0.6023**	**0.6375**
SSIM	0.1472	0.1568	0.1545	0.7656	0.7273	0.7375
VDP2	0.1194	0.1395	0.1373	0.6562	0.7614	0.7583
VIFp	**0.0823**	**0.0940**	**0.0923**	**0.4375**	**0.6250**	**0.5958**
VMAF	**0.0775**	**0.1068**	**0.1047**	**0.6094**	**0.5739**	**0.6375**
WaD LW	0.1772	0.2044	0.1996	0.7969	0.7954	0.8125
WaD TW	0.1512	0.1673	0.1670	0.7812	0.7216	0.7583

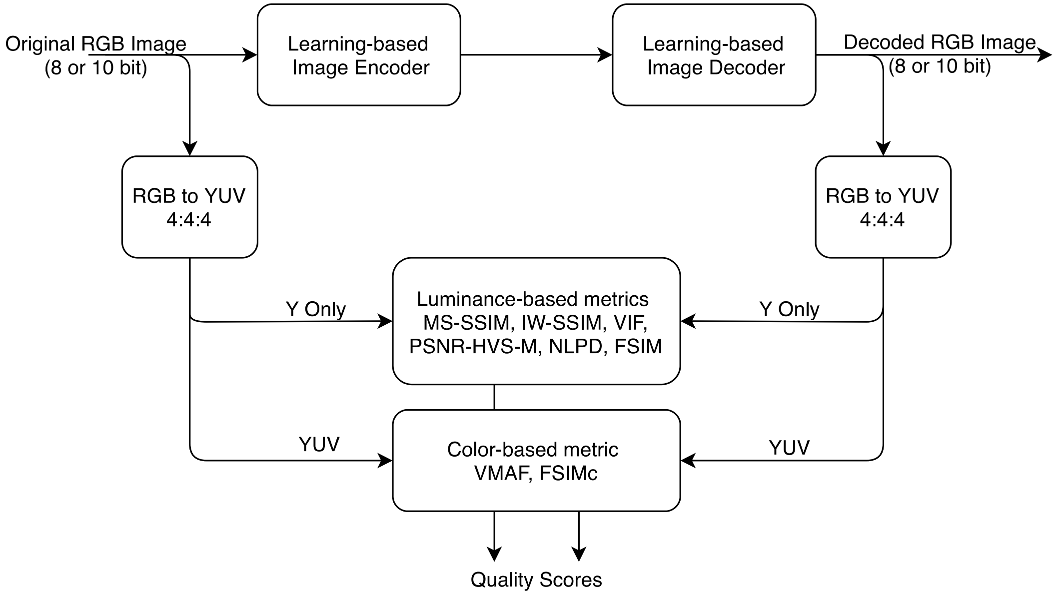
JPEG AI Objective Quality Assessment Framework

- Objective Quality Assessment Framework with reference implementation of all quality metrics defined
  - Cross-checked, including correlation performance assessment
  - Outputs CSV/TXT statistics
  - Supports several types of color conversions
  - Available in JPEG Gitlab https://gitlab.com/wg1/jpeg-ai/jpeg-ai-qaf
  - Summary file contains: name of the reconstructed image, BPP, MS-SSIM (by PyTorch), MS-SSIM (by IQA), PSNR Y, U, V, VIF, FSIM, NLPD, IW-SSIM, VMAF and PSNR-HVS.

- Complexity assessment of encoder and decoder operations
  - Model size, kMAC/px, model precision, running time, etc..

JPEG AI Objective Evaluation

- Objective quality evaluation using JPEG AI defined quality assessment metrics
  - MS-SSIM, IW-SSIM, VMAF, VIFP, PSNR-HVS-M, NLPD and FSIM
- Target bitrates for the objective evaluations include 0.03, **0.06, 0.12, 0.25, 0.50, 0.75**, 1.00, 1.50, and 2.00 bpp
- JPEG AI anchors were generated according to the JPEG AI CTTC:
  - HEVC Intra, VVC Intra, JPEG2000 and JPEG



Performance Relatively to VVC anchor

- Average BD-rate performance over all quality metrics
- Device interoperability pass

TEAMID	BD-rate vs VVC	Passed Cross-check
TEAM14	**-32.3%**	**YES**
TEAM24	**-29.9%**	**YES**
TEAM16	**-17.9%**	**YES**
TEAM12	**-3.1%**	NO
TEAM22	**7.2%**	NO
TEAM19	**8.6%**	**YES**
TEAM13	**10.6%**	**YES**
TEAM21	**13.8%**	NO
TEAM17	**32.0%**	NO
TEAM15	**51.2%**	NO

JPEG AI Common Training and Test Conditions

- Subjective quality evaluation for standard reconstruction
  - Double Stimulus Continuous Quality Scale (DSCQS) methodology
- Objective quality evaluation for standard reconstruction using JPEG AI defined quality assessment metrics
  - MS-SSIM, IW-SSIM, VMAF, VIFP, PSNR-HVS-M, NLPD and FSIM
- Complexity evaluation of both encoding and decoding process
  - Number of parameters, model precision, running time, MAC operations, etc.
- Device interoperability requirement
  - 0.5% BD rate mismatch between CPU and GPU

JPEG AI Anchors

- JPEG (ISO/IEC 10918-1 | ITU-T Rec. T.81)
  - JPEG XT reference software
- JPEG 2000 (ISO/IEC 15444-1 | ITU-T Rec. T.800)
  - Kakadu software
  - PSNR optimized and visually optimized
- HEVC Intra (ISO/IEC 23008-2 | ITU-T Rec. H.265)
  - HEVC Test Model (HM 16.20)
- VVC Intra (ISO/IEC 23090-3 | ITU-T Rec. H.266)
  - VVC Test Model (VTM 11.1)
- FFMPEG for the PNG (RGB) to YUV files conversion

JPEG AI Call for Proposals Dataset

- Training/validation dataset: 5264/350 images
- Proponents must use training dataset
- Test images are kept hidden until decoder submission, to avoid being used for training or validation
- Call for Proposals test set includes 21 images

Performance Relatively to VVC anchor

- Average BD-rate performance over all quality metrics
- Decoding run time relative to anchor using the same CPU (times)

TEAMID	BD-rate performance			CPU dec. time		
	J2K	HEVC	VVC	J2K	HEVC	VVC
TEAM12	**-39.3%**	**-13.2%**	**-3.1%**	601	606	484
TEAM13	**-31.5%**	**-2.1%**	**10.6%**	21	21	16
TEAM14	**-57.2%**	**-39.6%**	**-32.3%**	39	39	31
TEAM15	**-6.7%**	**33.6%**	**51.2%**	25	25	19
TEAM16	**-47.7%**	**-26.6%**	**-17.9%**	44	44	34
TEAM17	**-21.5%**	**15.4%**	**32.0%**	98	98	75
TEAM19	**-34.2%**	**-4.4%**	**8.6%**	21	21	16
TEAM21	**-33.4%**	**1.6%**	**13.8%**	153	153	118
TEAM22	**-32.6%**	**-4.9%**	**7.2%**	136	136	105
TEAM24	**-56.5%**	**-37.4%**	**-29.9%**	44	44	34

Decoding Complexity

- kMAC/pxl (amount of multiplication per one pixel of reconstructed image)
- Number of parameters for one stream decoding (the worst case), million of parameters
- Total number of parameters in all models, million of parameters
- GPU decoding submissions time with respect to HEVC and VVC (CPU)
  - GPU board: NVIDIA Tesla V100-SXM2-32GB

TEAMID	Decoding complexity			Decoding time GPU (HEVC)	Decoding time GPU (VVC)
	kMAC/pxl	Largest Model Size	Total Model Size		
TEAM12	no data	47	479	NA	NA
TEAM13	419	208	344	81.56%	61.52%
TEAM14	1266	38	152	800.61%	603.88%
TEAM15	1262	13	39	225.16%	169.83%
TEAM16	576	20	428	41.07%	30.90%
TEAM17	961	40	40	1665.04%	1255.41%
TEAM19	478	208	344	103.9%	78.37%
TEAM21	1348	25	59	NA	NA
TEAM22	281	4	17	NA	NA
TEAM24	593	20	326	40.58%	30.61%

Device Interoperability

- Decoding of submitted bitstreams was made in a cross-check fashion
  - Proponent A has decoded the bitstreams of proponent B and has measured the bitstream size and objective quality and vice-versa.
- Proposal passes the cross-check when the performance assessment results reported by proponent and cross-checker are very similar (BD-rate less than 0.5%) and no decoder crash was reported

TEAMID	BD-rate vs VVC	Passed Cross-check
TEAM14	**-32.3%**	**YES**
TEAM24	**-29.9%**	**YES**
TEAM16	**-17.9%**	**YES**
TEAM12	**-3.1%**	NO
TEAM22	**7.2%**	NO
TEAM19	**8.6%**	**YES**
TEAM13	**10.6%**	**YES**
TEAM21	**13.8%**	NO
TEAM17	**32.0%**	NO
TEAM15	**51.2%**	NO

JPEG AI Milestones

Overview of existing Technologies

Q-metrics selection

CfE
Call for Evidence

Exploration Experiments

CfP
Call for Proposals

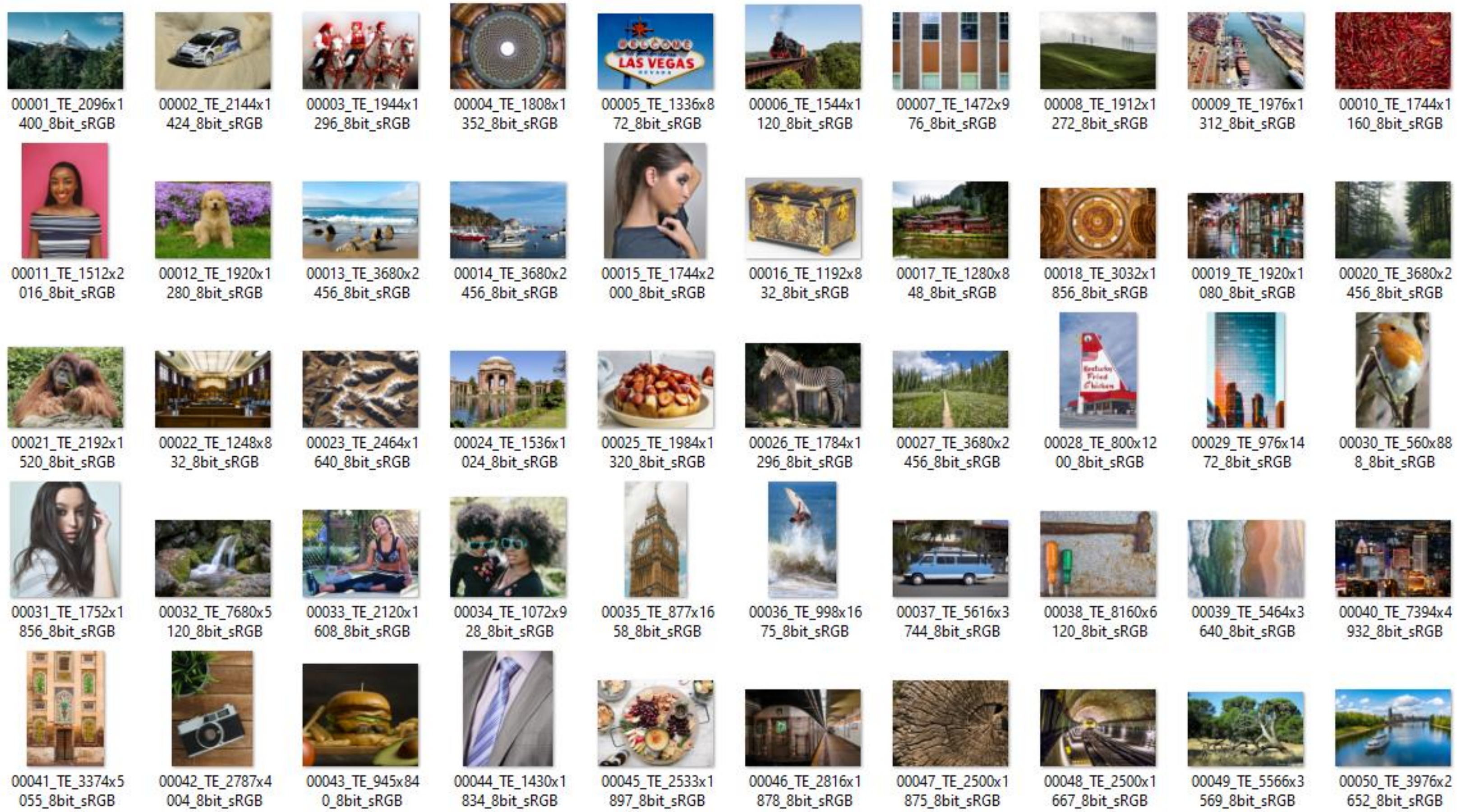
Decoder on smartphone

DIS
Draft International Standard

IS
International Standard

2019 | 2020 | 2021 | 2022 | 2023 | 2024 | 2025

How to Ensure Against Overfitting?



JPEG AI Test Set:
50 camera captured images

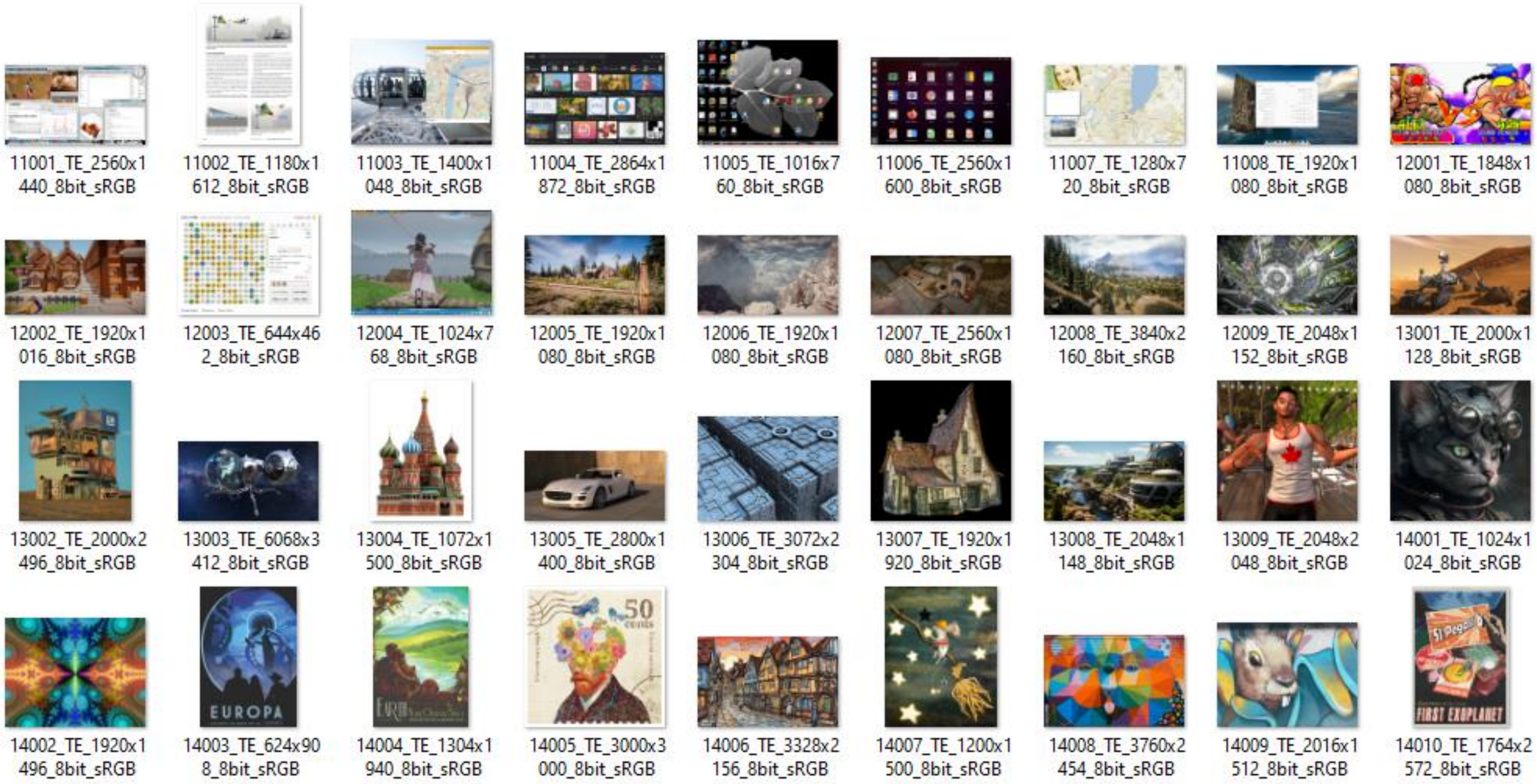
!=

Training Set:
5000+ images
Validation Set:
350+ images

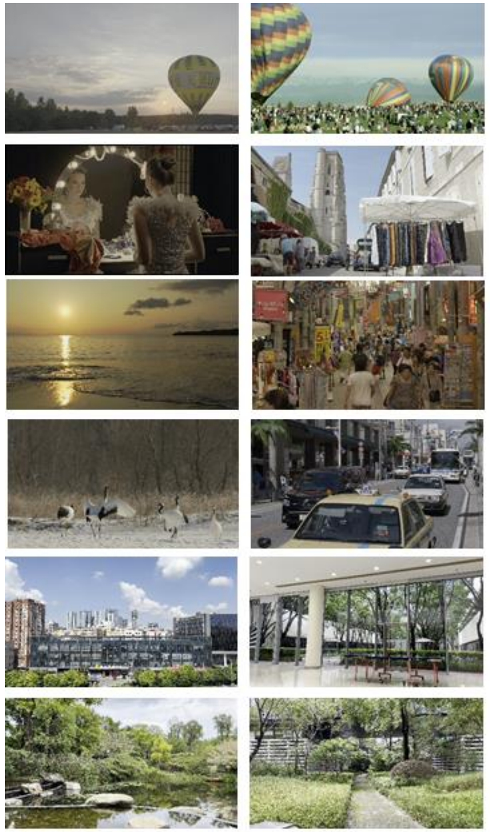
63

JPEG AI Additional Test Sets

36 Synthetic Images

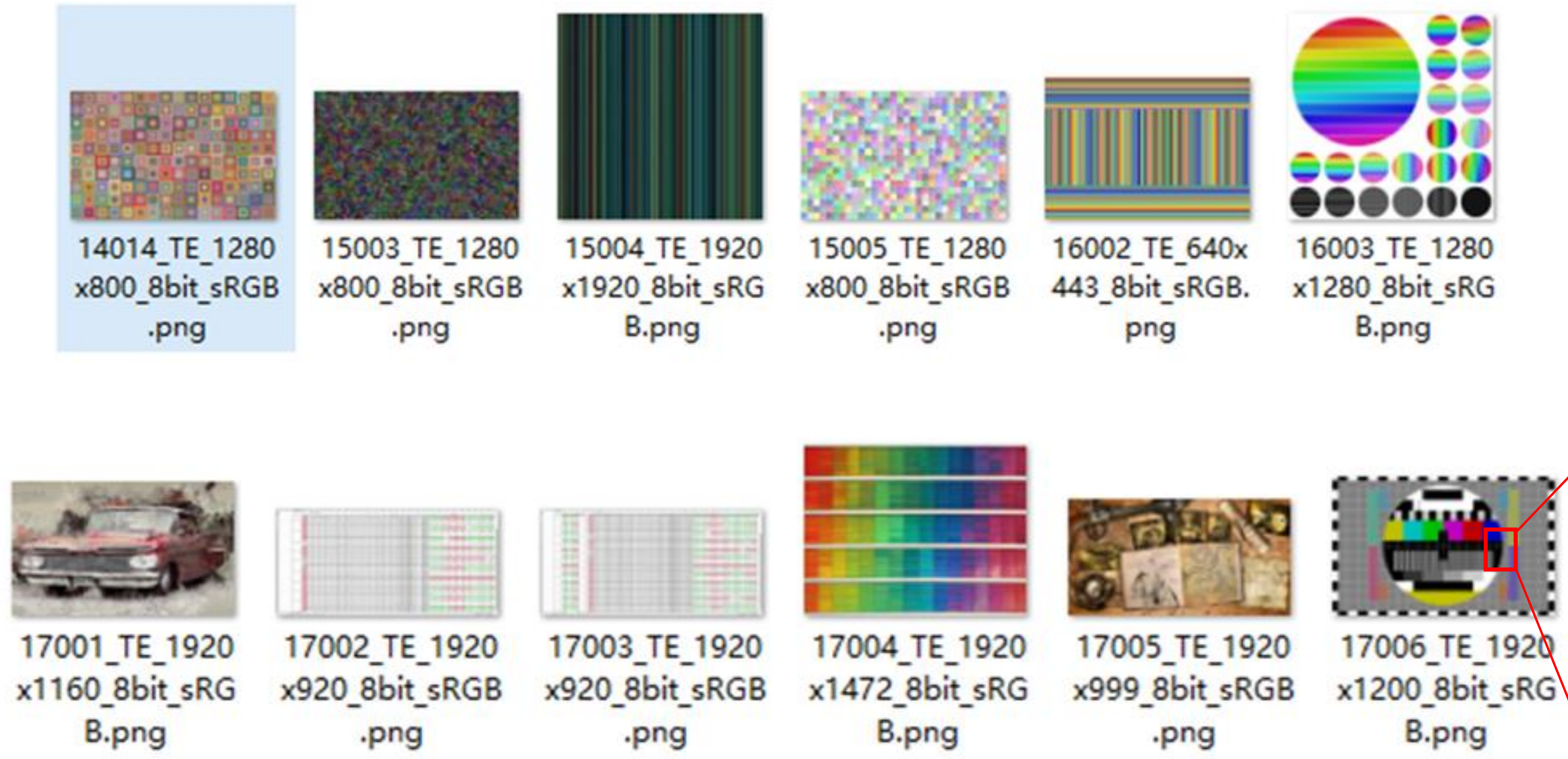


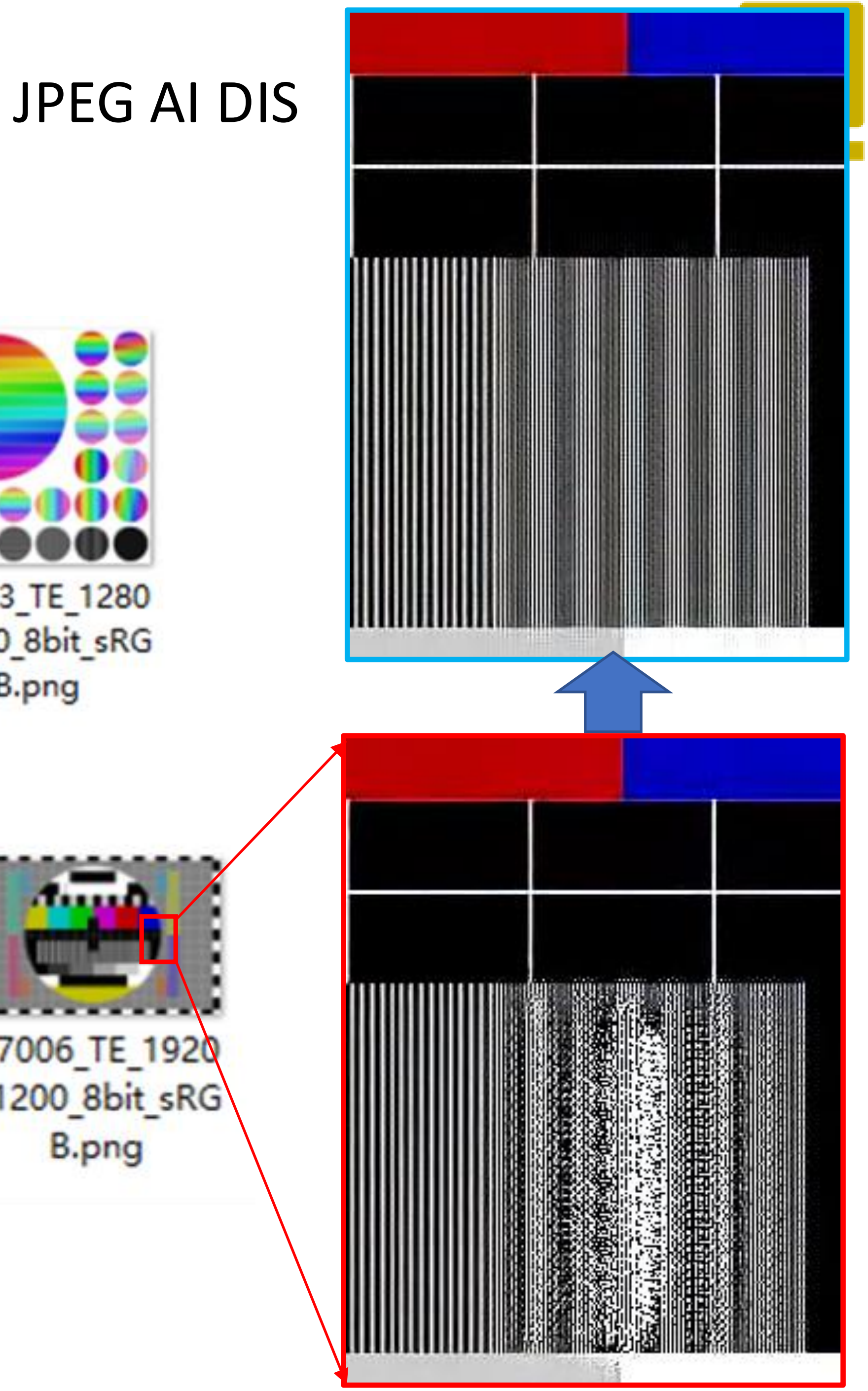
12 HDR Images



64

JPEG AI Crash Dataset





Profiling: Addressing Complexity Issues

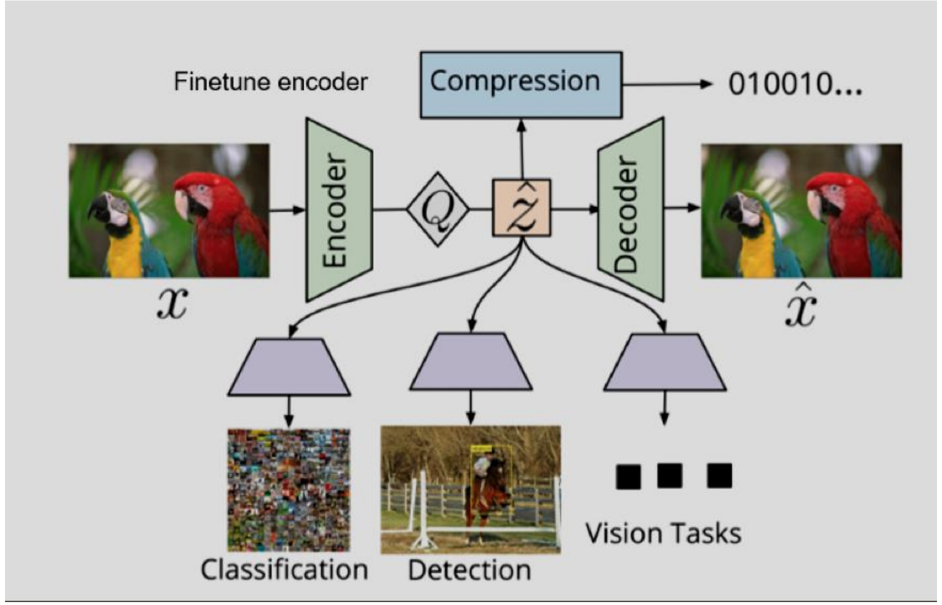
- Part 2 defines profiles and levels, i.e. set of constraints on the codestream and reconstruction process for efficient implementation across various applications.
- JPEG AI uses a nested profile structure with one stream profile and three decoder profiles with different complexity-quality trade-offs
  - A stream profile defines a subset of codestream syntax and their admissible values.
- The encoder can enable the use of one or more synthesis transforms for each decoder profile
  - CPU operating point targeting legacy devices
  - Base operating point medium to high capability mobile devices
  - High operating point for more powerful devices with no energy constraints


Compressed Domain Processing

VCIP, Klagenfurt, Austria, December 2025

JPEG AI Bitstream is Multi-purpose

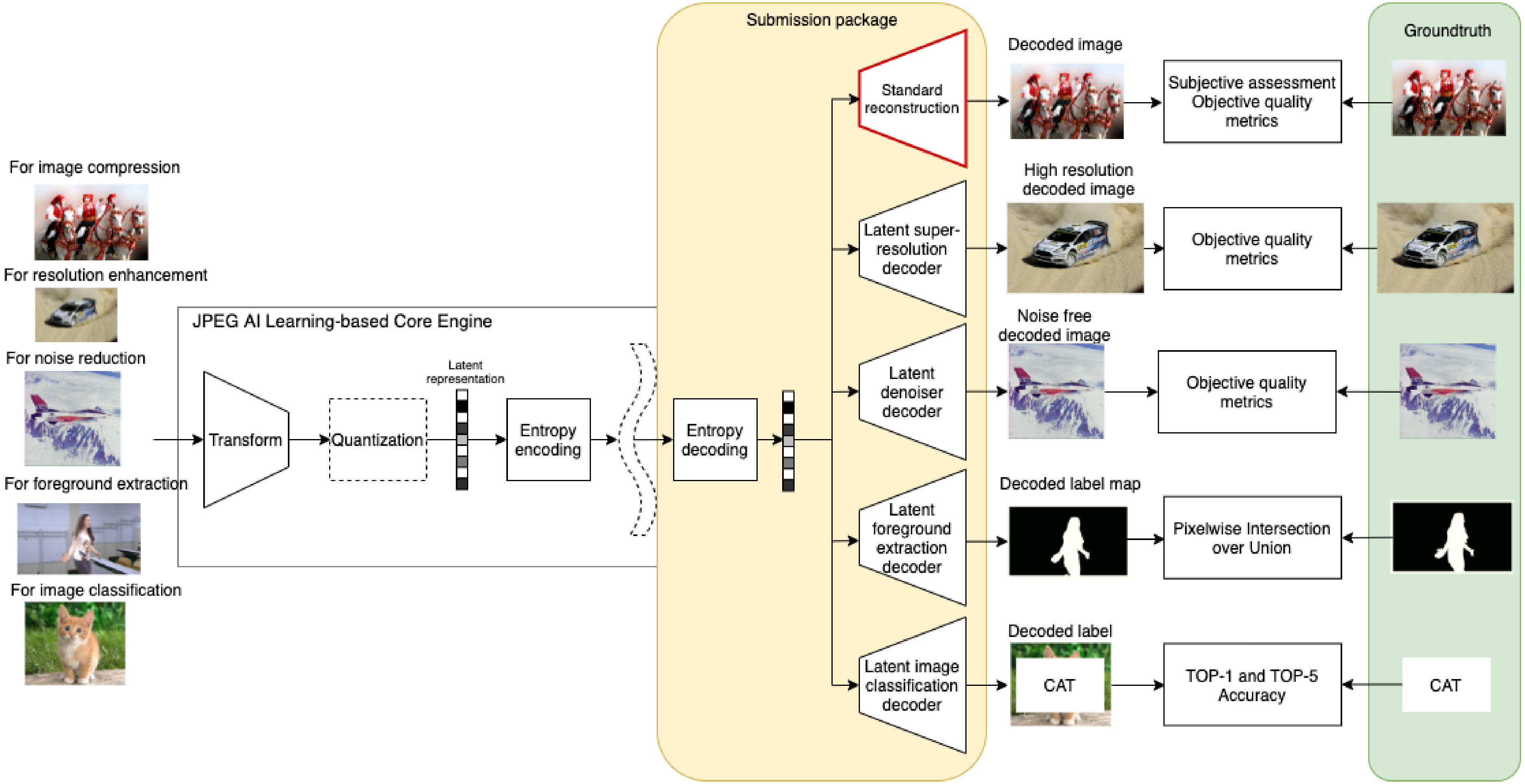
- Tasks under consideration:
  - Compressed Domain Image Classification
  - Compressed Domain Real-time Foreground Extraction
  - Compressed Domain Super-Resolution
  - Compressed Domain Denoising



- Following the vision of a multi-purpose bitstream !

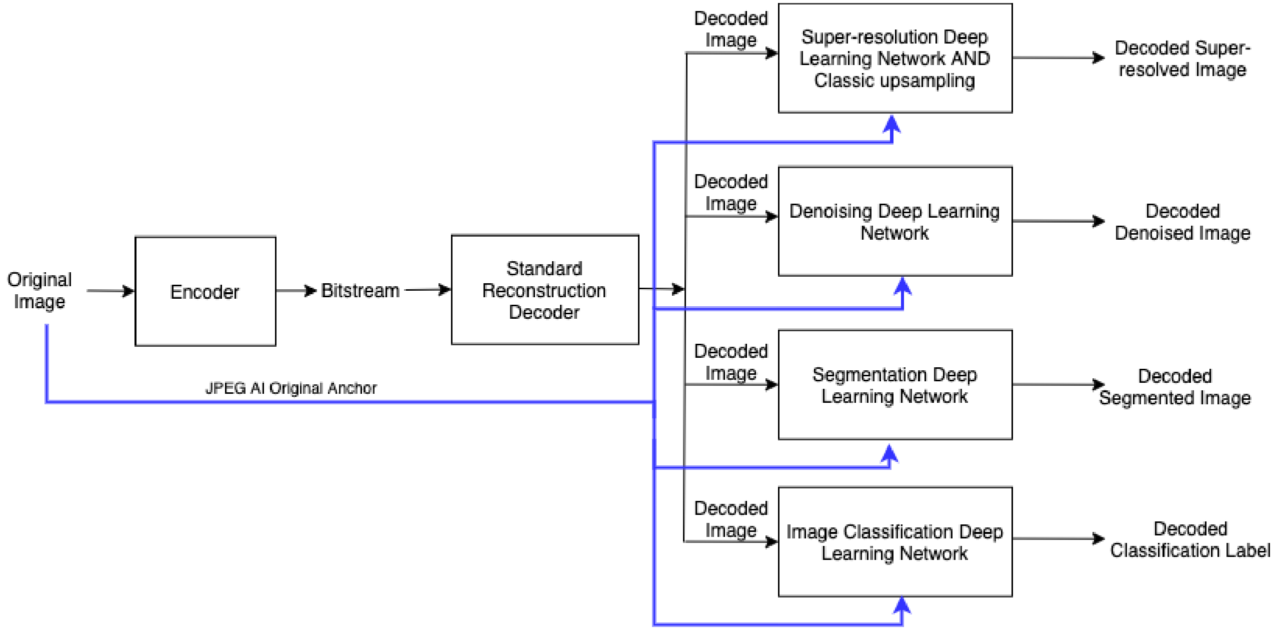
- All tasks performed directly on the latent representations produced by learning-based image codecs
  - Requires to perform entropy decoding only
  - Reduces the computational complexity needed to perform these tasks
  - Features extracted from the original are used instead from the lossy decoded images

JPEG AI Pipelines



JPEG AI Anchors

- Original anchor: Processing task is applied to the original images, before any compression, to assess the performance without any compression artifacts
- Decoded anchor: Processing task is applied to fully decoded RGB images, i.e., from the decoded pixel-wise representation

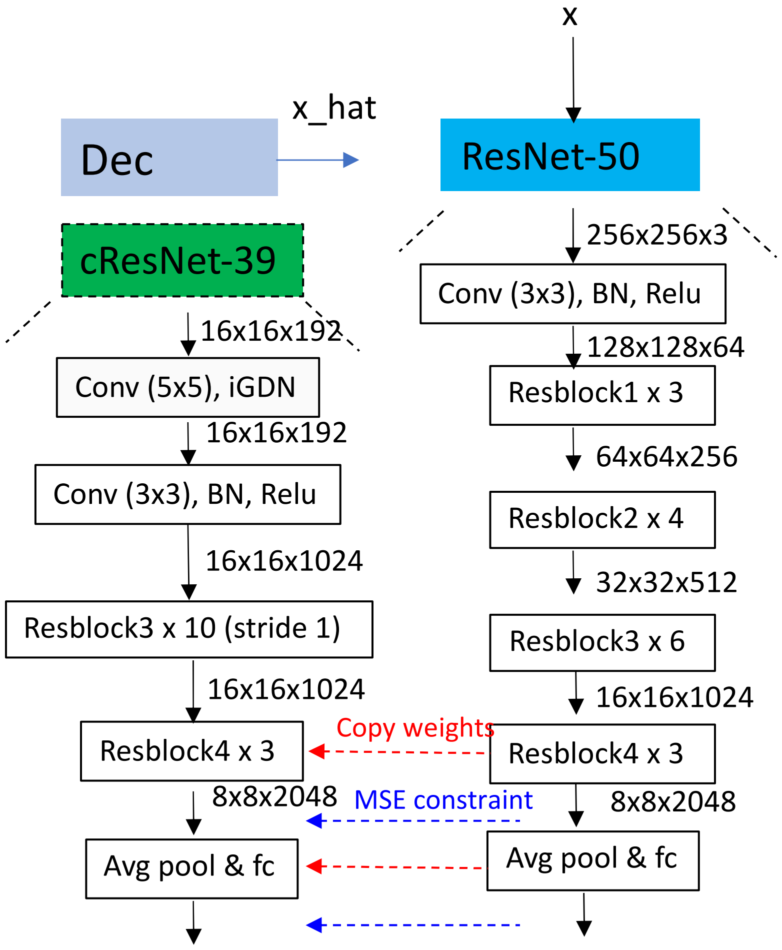


Compress Domain Image Classification

- Objective: Image classification performed directly on the latent representations produced by learning-based image codecs
    - Compute a label of an image from a pre-defined set of 1000 classes
- Learning-based image codec: proponent submission for standard reconstruction
- Anchor method: pre-trained Resnet-50
- Training dataset: ImageNet 2012 dataset
- Bitrates: 0.15 to 1.8 bpp
- Performance metrics:
    - Top-1 accuracy: probability of the label of the top-1 image (with highest confidence) being the true label
    - Top-5 accuracy: probability of the label of the top-5 images (with highest confidence) being the true label
- Complexity assessment similar to the standard reconstruction task

Compress Domain Image Classification

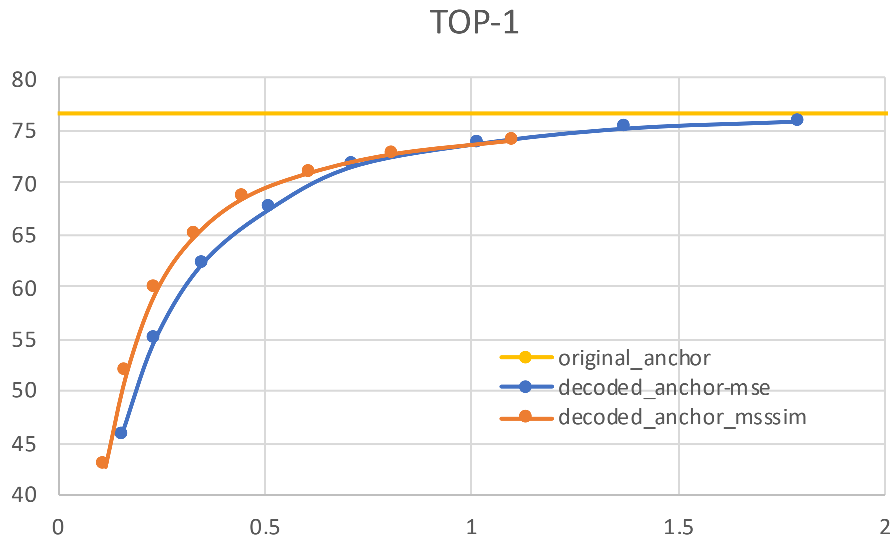
- Training procedure
  - Batch size: 256
  - Learning rate: 0.0005
  - Optimizer: Adam
  - Scheduler: OneCycleLR
  - Epochs: 50
  - Loss Func:
    - CrossEntropy + MSE(layer4) * w + MSE(fc)
  - Model weights init:
    - Layer 4 and fc layer weights were initialized as those in Resnet-50

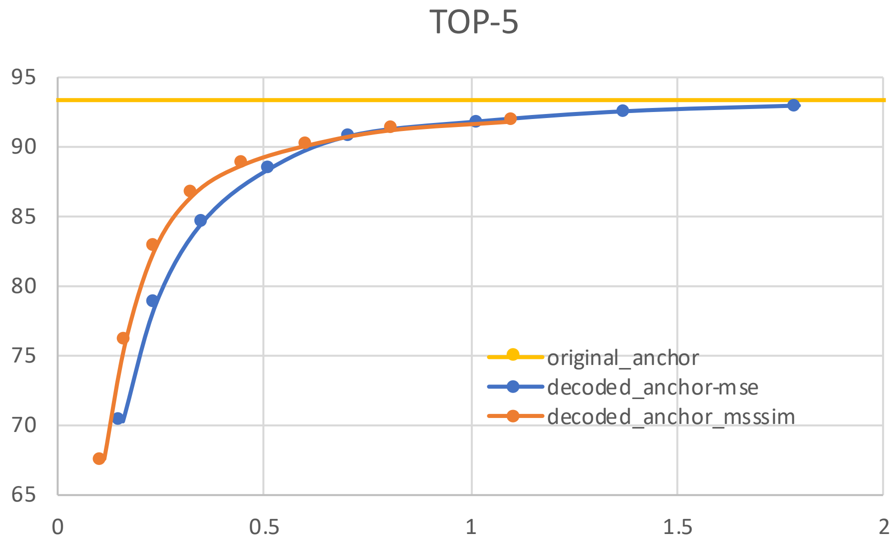


72

Rate-Accuracy Performance: Anchors

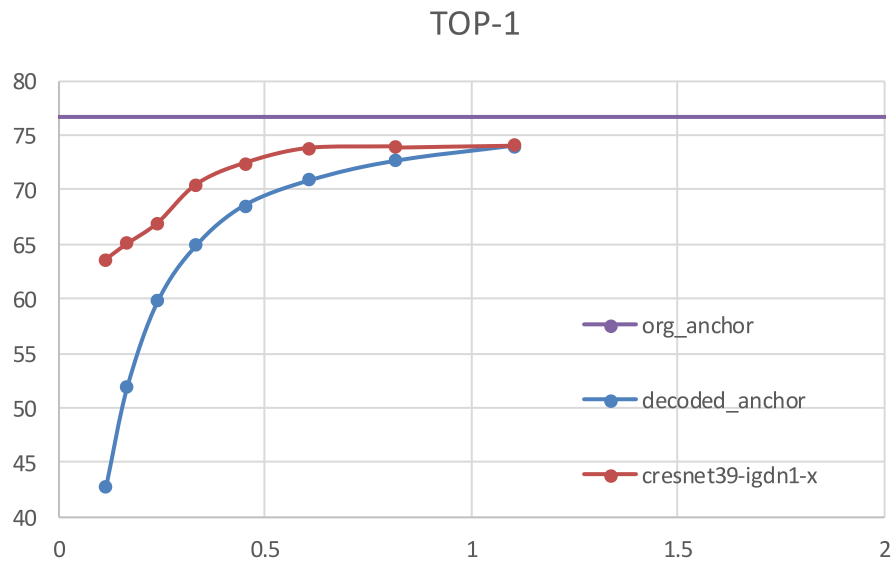
- Accuracy metrics:
  - **Top-1:** probability of the label of the top-1 image (with highest confidence) being the true label
  - **Top-5:** probability of the label of the top-5 images (with highest confidence) being the true label
- Learning-based decoded anchor optimized to MSE and MS-SSIM

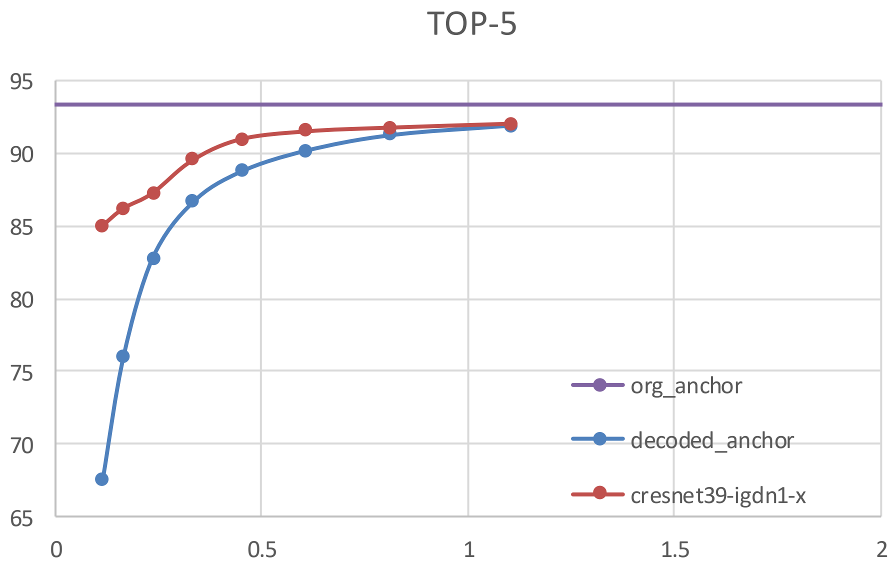




Rate Accuracy Performance

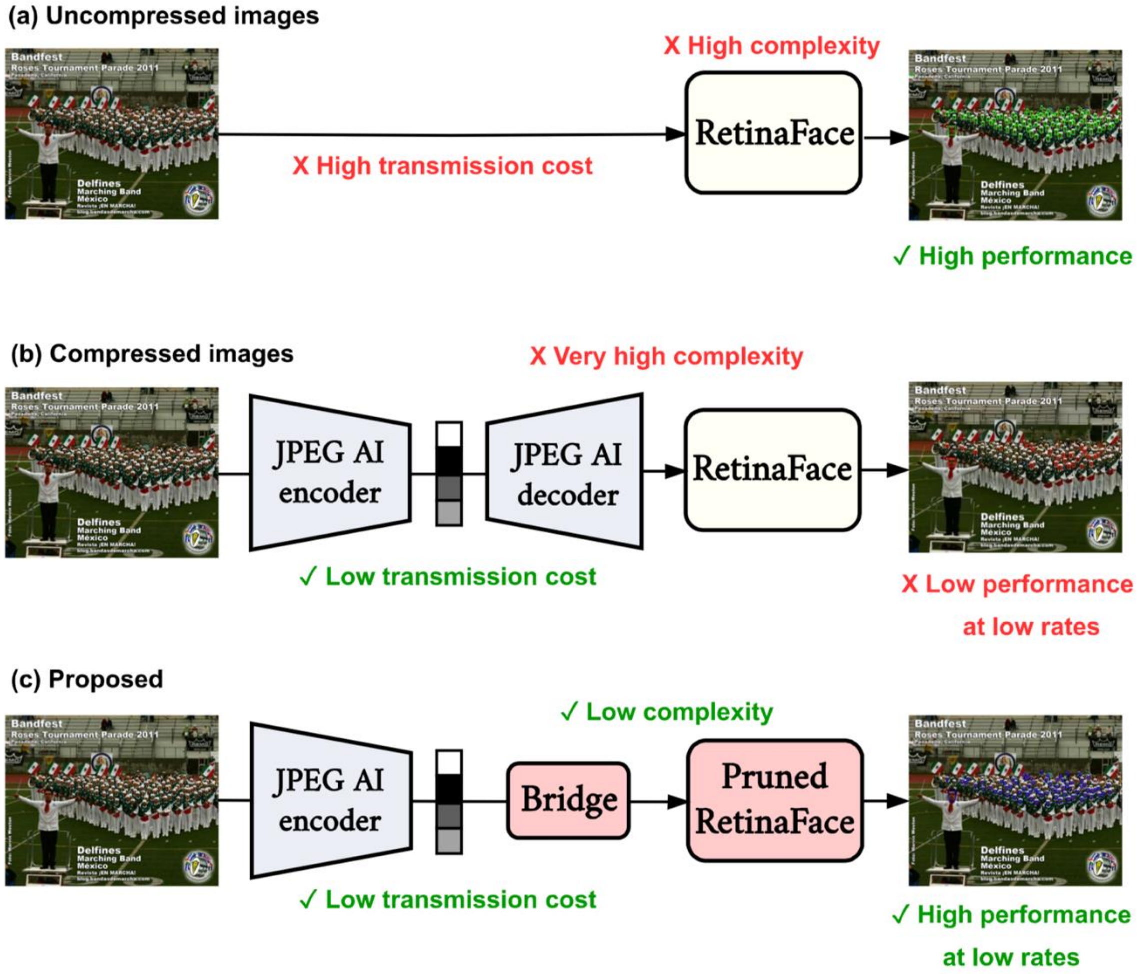
- JPEG AI Exploration Studies have showed great potential for this task:
  - High performance especially at low bitrates
  - Less complexity to perform image classification from the latent representation
- Performance results are for the MS-SSIM loss function using the Ballé et al. Hyperprior image codec





Reference Face Detection Model: RetinaFace

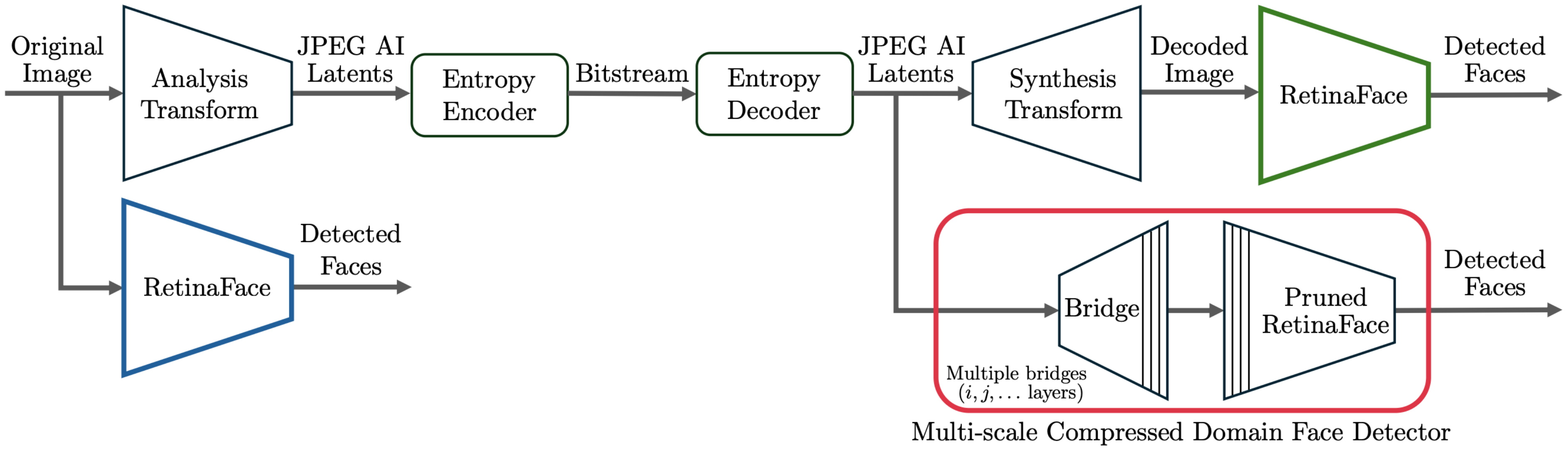
- Motivation: pixel-domain face detectors degrade significantly on JPEG AI decoded images, especially at low bitrates.
- Goal: perform face detection directly on JPEG AI latents, bypassing expensive image reconstruction.

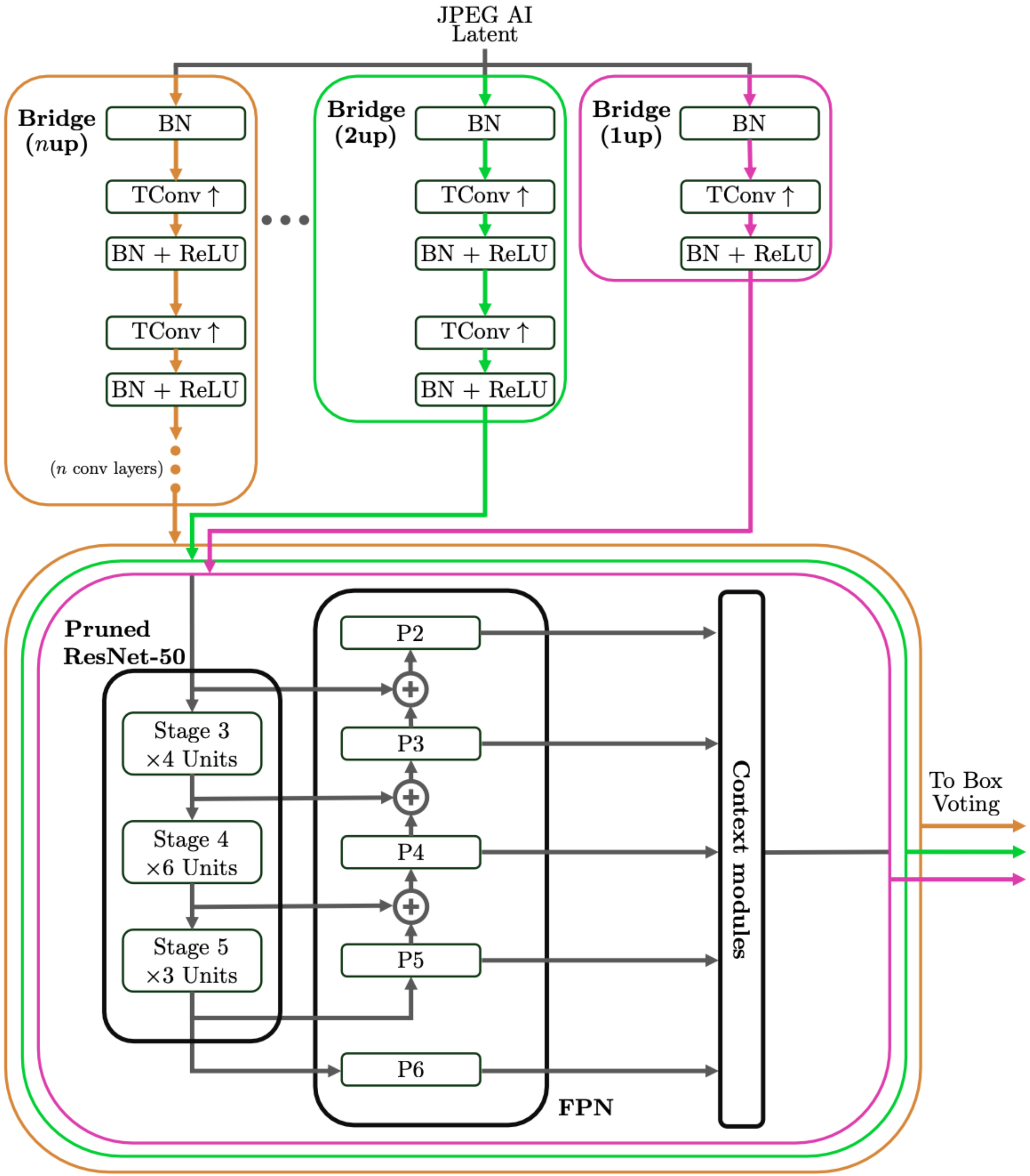


REF: Alkhateeb, A., Gnutti A., Guerrini F., Leonardi R., Ascenso J., Pereira F., "JPEG AI MMSP, West Lafayette, United States, October, 2024.

Multi-scale Bridging for Compressed-Domain RetinaFace

- Input: JPEG AI latent tensor, no decoded image required.
- Two innovations:
  - Variable-size latent processing: no fixed cropping; detector uses original image resolution.
  - Multi-scale bridging: multiple parallel bridges (2-up and 3-up) learn different spatial upsampling pathways.
- Each bridge is paired with a pruned RetinaFace backbone (stages 1–2 removed)
- Detection outputs from each scale fused via box voting




JPEG AI
Latent
Bridge
($n$up)
BN
TConv ↑
BN + ReLU
TConv ↑
BN + ReLU
(n conv layers)
Bridge
(2up)
BN
TConv ↑
BN + ReLU
TConv ↑
BN + ReLU
Bridge
(1up)
BN
TConv ↑
BN + ReLU
Pruned
ResNet-50
Stage 3
×4 Units
Stage 4
×6 Units
Stage 5
×3 Units
P2
P3
P4
P5
P6
FPN
Context modules
To Box
Voting

WIDER FACE dataset

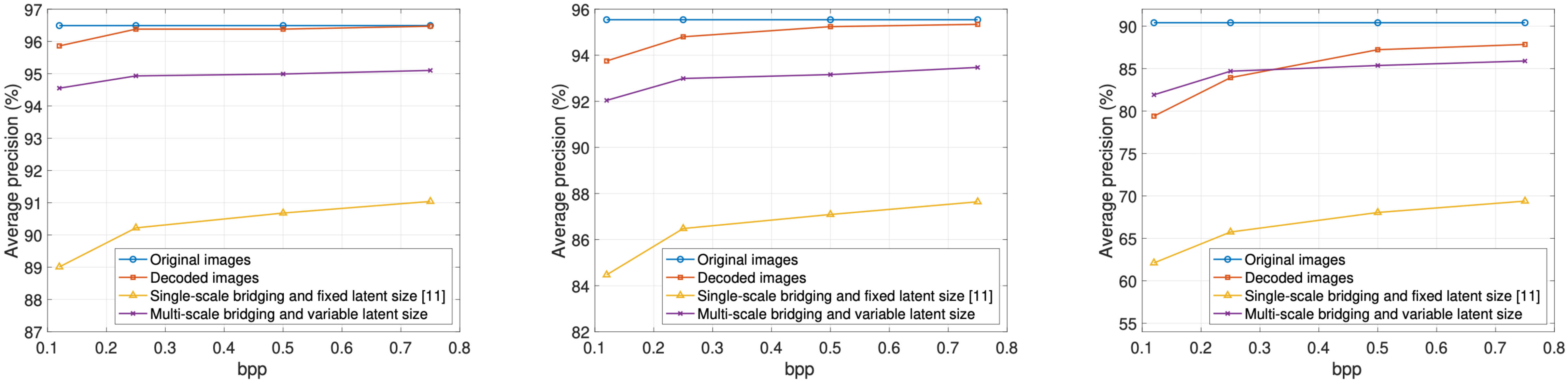
Easy level

Medium level

Hard level

JPEG AI Face Detection Performance Comparison

- +4–8% AP improvement (easy/medium) over single-scale
- Up to +20% AP on hard subset, especially tiny faces
- Nearly matches RetinaFace on decoded images; sometimes surpasses it at low bitrate



(a) Easy level. (b) Medium level. (c) Hard level.

Complexity Assessment

- JPEG AI decoding + RetinaFace ≈ 536 GMACs
- Multi-scale 2+3-up ≈ 170 GMACs → ≈32% of pixel-domain cost
- Conclusion: Multi-scale bridging provides high accuracy with dramatic complexity savings; ideal for on-device and large-scale vision systems

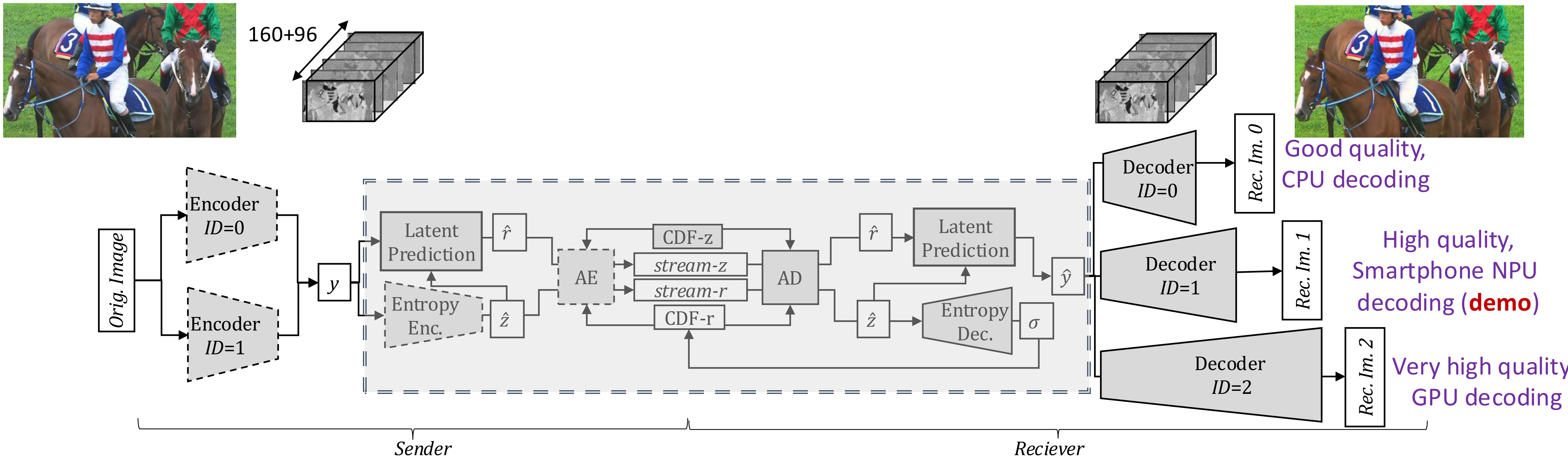
Table I: Complexity (in GMACs) of the face detection process in various configurations for 640×640 input image size.

Module	**Complexity (GMAC)**	**Approach**					
		Decoded	1up	2up	3up	MS (2+3up)	MS (1+2+3up)
JPEG AI decoder	90.52	1x					
RetinaFace (per scale)	44.56	10x					
Bridge 1/2/3 up	2.49		1x				1x
	15.69			1x		1x	1x
	78.50				1x	1x	1x
Pruned RetinaFace	37.97		1x	1x	1x	2x	3x
Total complexity (GMAC)		**536.12**	**40.46**	**53.66**	**116.47**	**170.13**	**210.49**

JPEG AI design

principles

JPEG AI: single stream multiple decodes



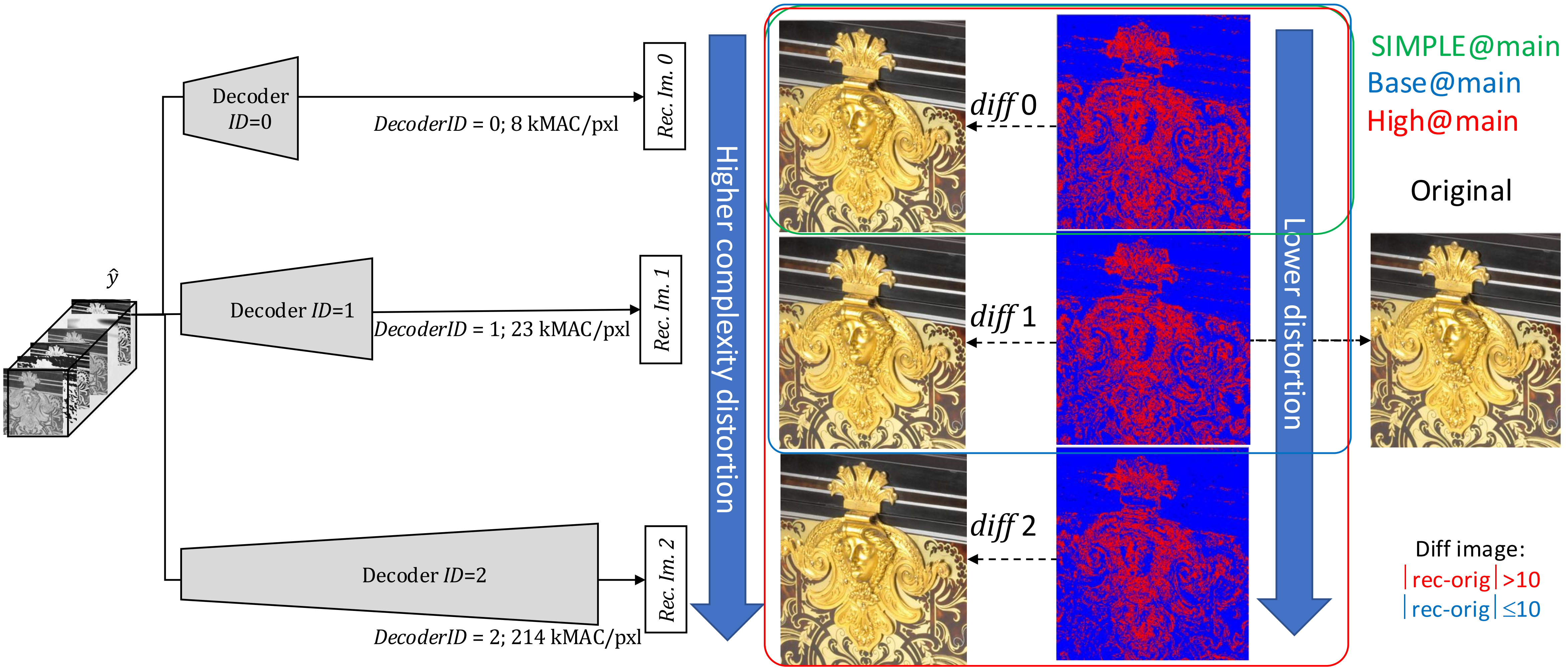
Run time: **encoder0** JPEG AI (GPU) = JPEG (CPU)

encoder1 JPEG AI (GPU) = 1/2000 · VTM (CPU)

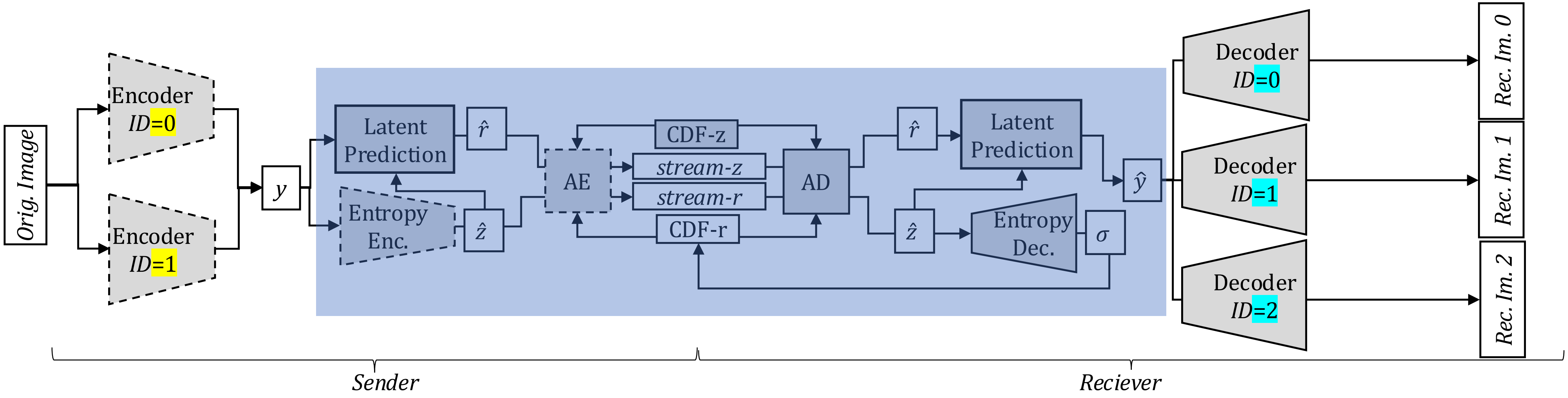
decoder1 JPEG AI (GPU) = ½ · VTM (CPU)

VTM – Reference SW for VVC/H.266

JPEG AI: single stream multiple decodes

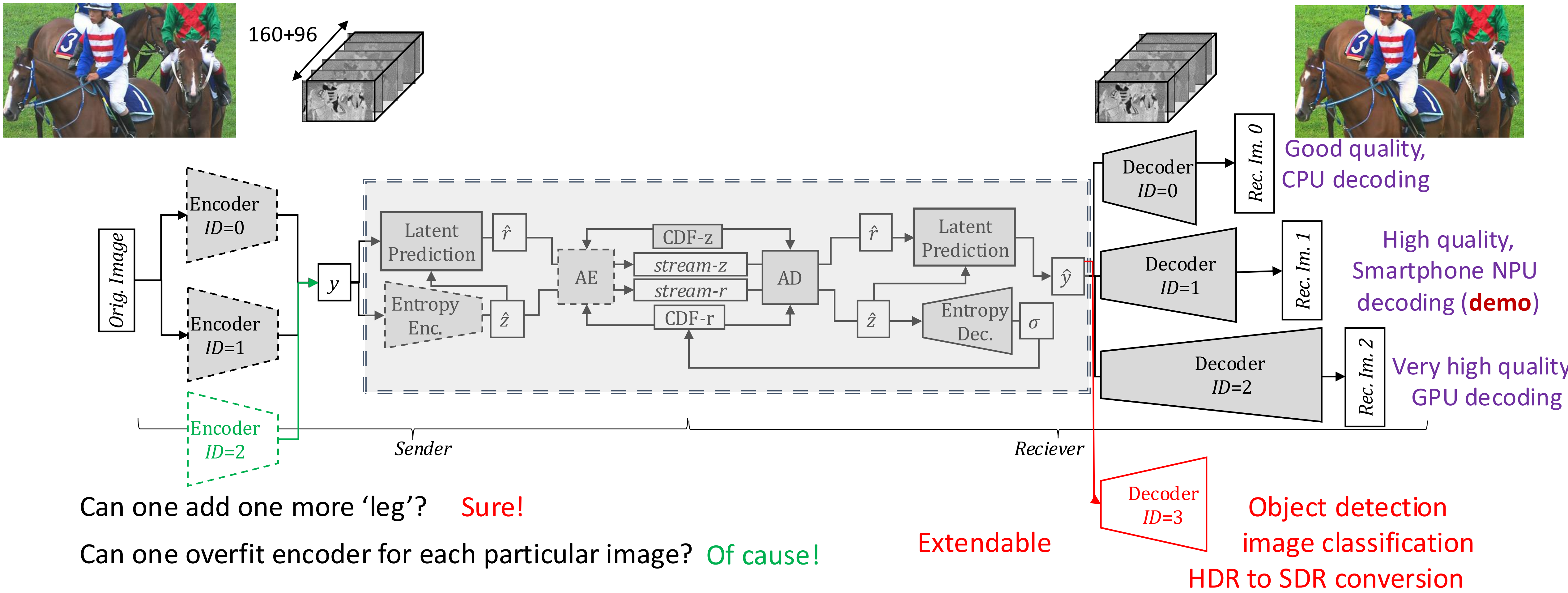


JPEG AI: single stream multiple decodes



How is it possible?... Just train 'multi-legs' NN.

JPEG AI: what else can we do?



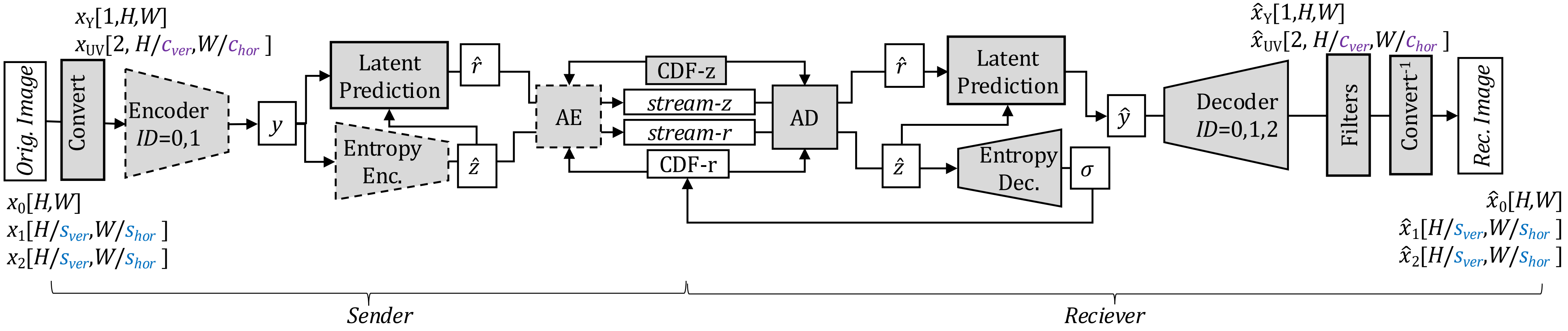
Can one add one more 'leg'? Sure!

Can one overfit encoder for each particular image? Of cause!

JPEG AI encoder & decoder

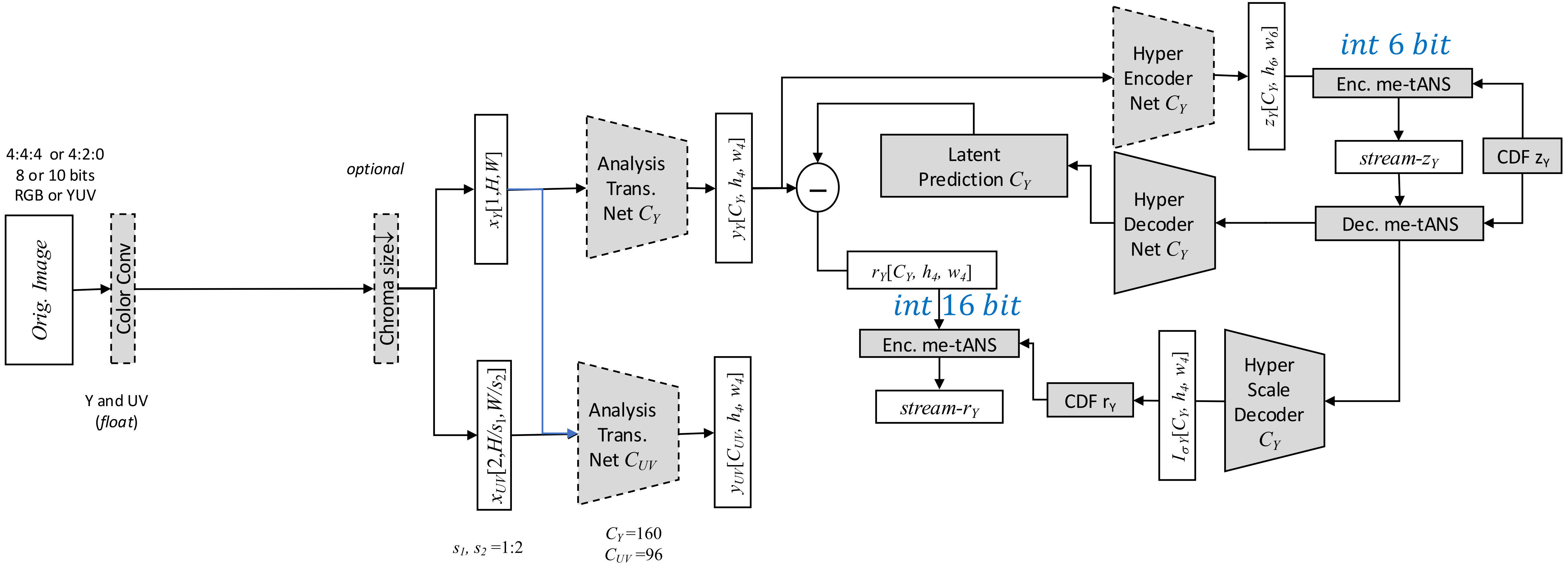
Step by step

Supported combinations of output picture format and scaling factors

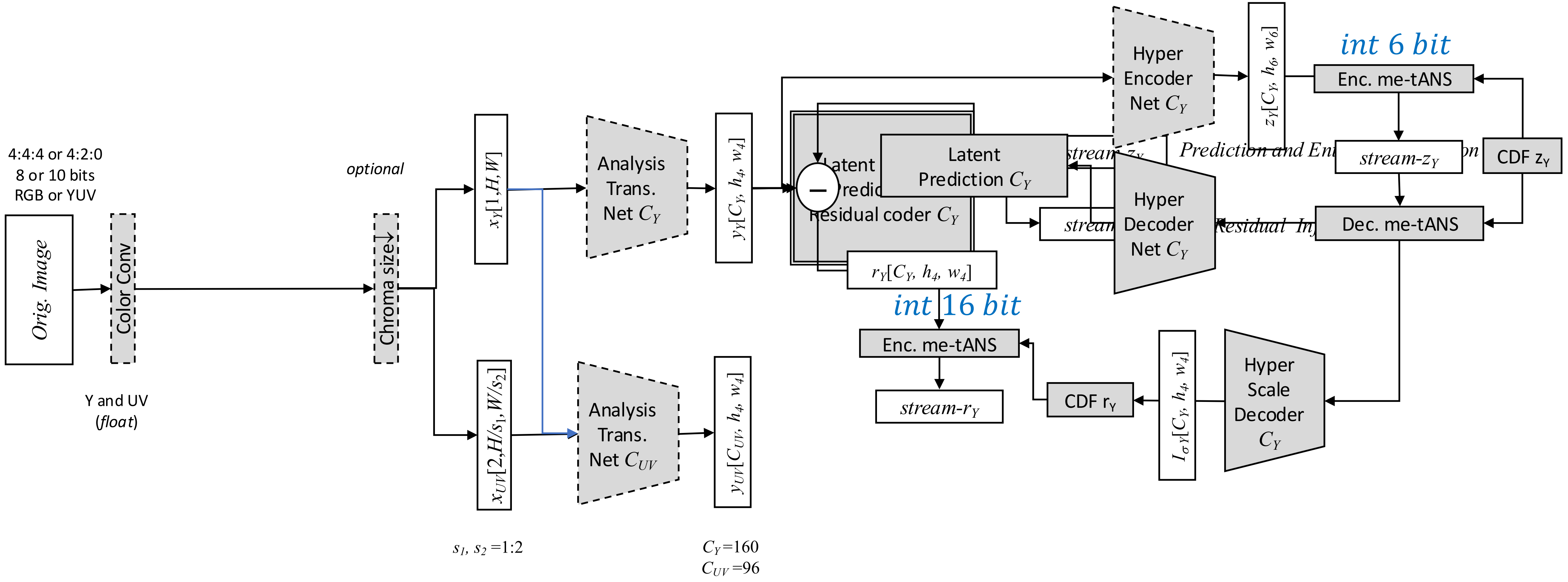


<table>
<tr><th colspan="5">Input/output image representation</th><th colspan="5">Coded image representation</th></tr>
<tr><th colspan="10">Color sampling mode</th></tr>
<tr><td>format</td><td>s_{hor}</td><td>s_{ver}</td><td>H_{UV}</td><td>W_{UV}</td><td>format</td><td>c_{hor}</td><td>c_{ver}</td><td>cH_{UV}</td><td>cW_{UV}</td></tr>
<tr><td rowspan="3">4:4:4</td><td rowspan="3">1</td><td rowspan="3">1</td><td rowspan="3">H</td><td rowspan="3">W</td><td>4:4:4</td><td>1</td><td>1</td><td>H</td><td>W</td></tr>
<tr><td>4:2:2</td><td>2</td><td>1</td><td>H</td><td>W/2</td></tr>
<tr><td>4:2:0</td><td>2</td><td>2</td><td>H/2</td><td>W/2</td></tr>
<tr><td rowspan="2">4:2:2</td><td rowspan="2">2</td><td rowspan="2">1</td><td rowspan="2">H</td><td rowspan="2">W/2</td><td>4:2:2</td><td>2</td><td>1</td><td>H</td><td>W/2</td></tr>
<tr><td>4:2:0</td><td>2</td><td>2</td><td>H/2</td><td>W/2</td></tr>
<tr><td>4:2:0</td><td>2</td><td>2</td><td>H/2</td><td>W/2</td><td>4:2:0</td><td>2</td><td>2</td><td>H/2</td><td>W/2</td></tr>
<tr><th colspan="10">Color space</th></tr>
<tr><td colspan="5">RGB</td><td colspan="5">YUVbt709</td></tr>
<tr><td colspan="5">YUVbt709</td><td colspan="5">YUVbt709</td></tr>
<tr><td colspan="5">any</td><td colspan="5">any</td></tr>
</table>

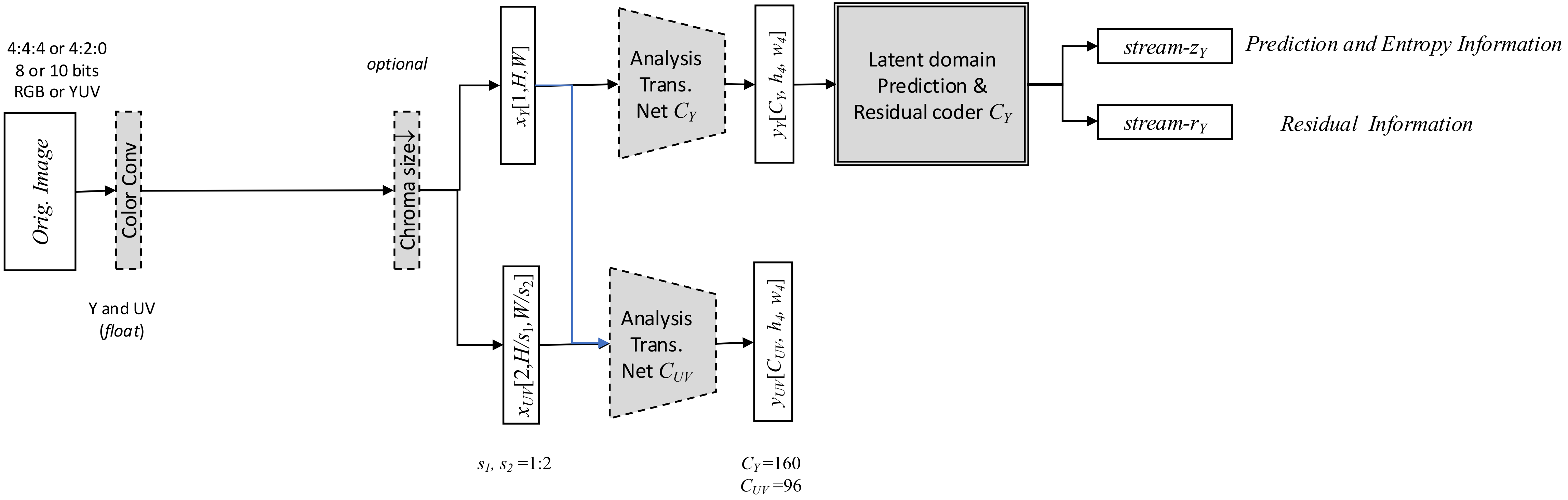
High level encoder diagram



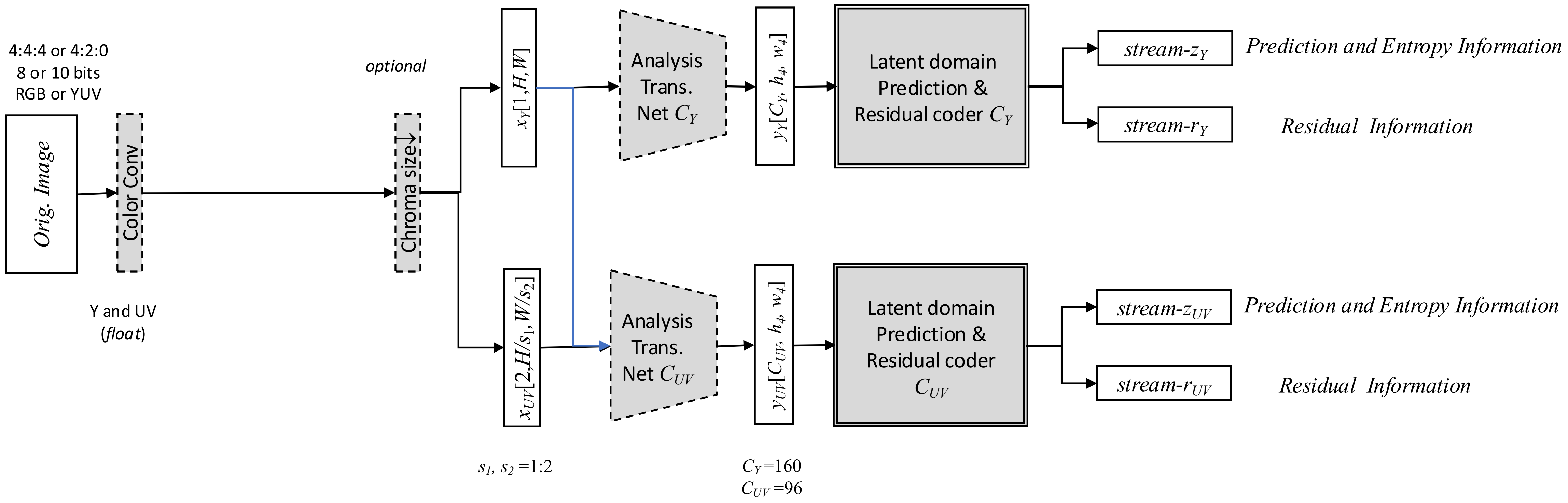
High level encoder diagram



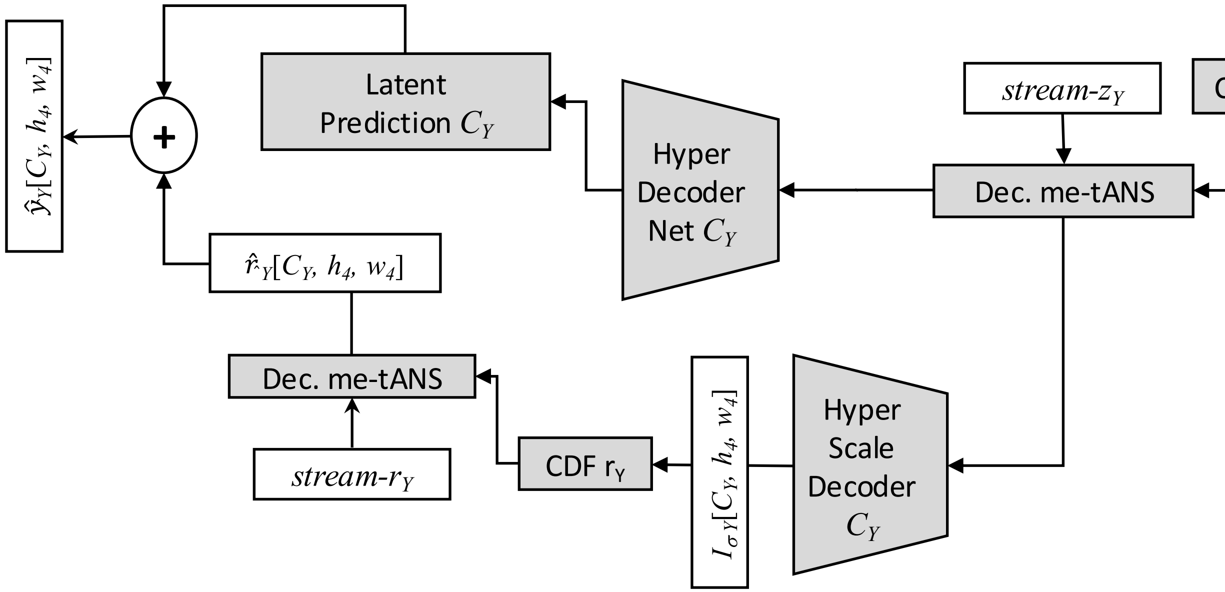
High level encoder diagram



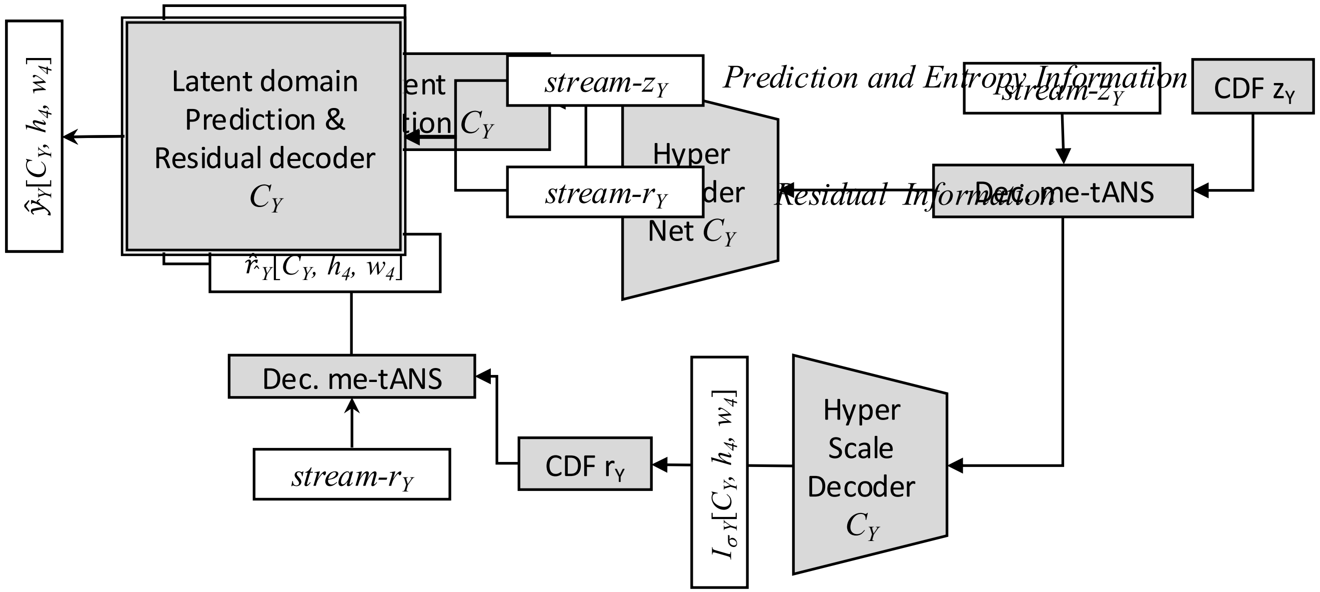
High level encoder diagram



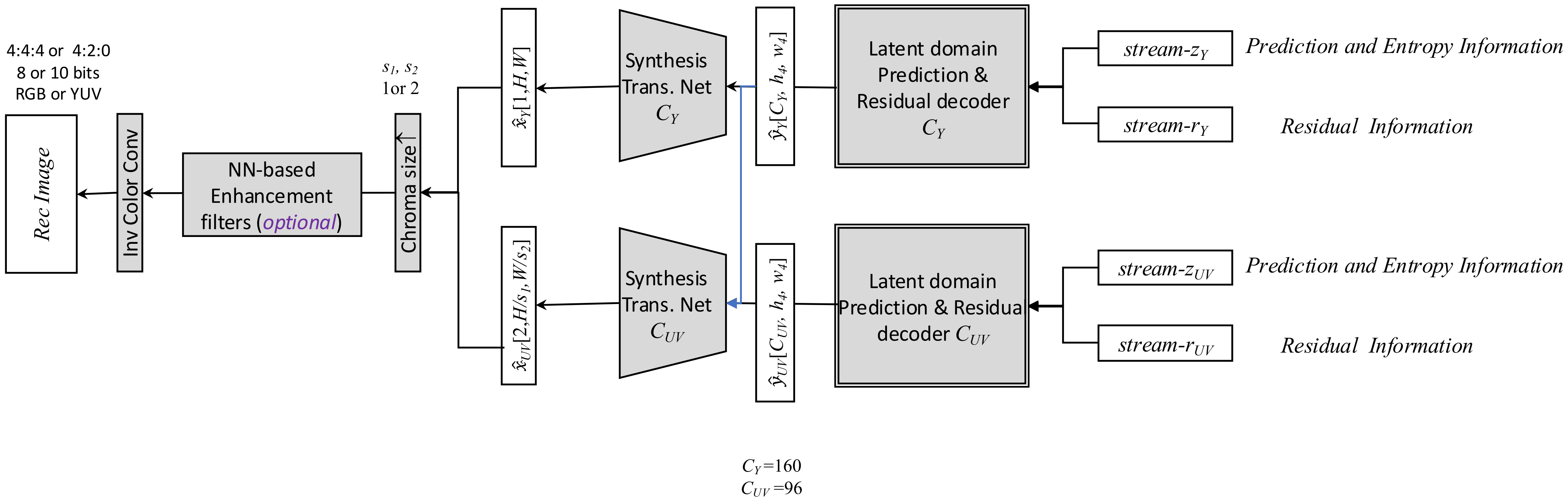
High level decoder diagram



High level decoder diagram



High level decoder diagram



JPEG AI stream structure and partitioning

JPEG AI stream structure

JPEG AI code stream

Start

Picture header

r_{UV} *-stream* | *Tools Header*

r_Y *-stream* | *Rendering Info*

z –stream $(z_Y \& z_{UV})$ | *Q-map*

User Defined

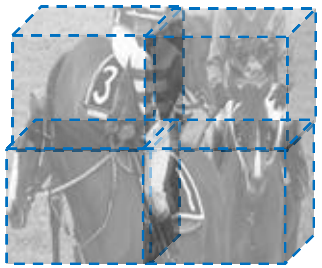
End

code stream segment

Marker	*Length*	*data*

code stream segments

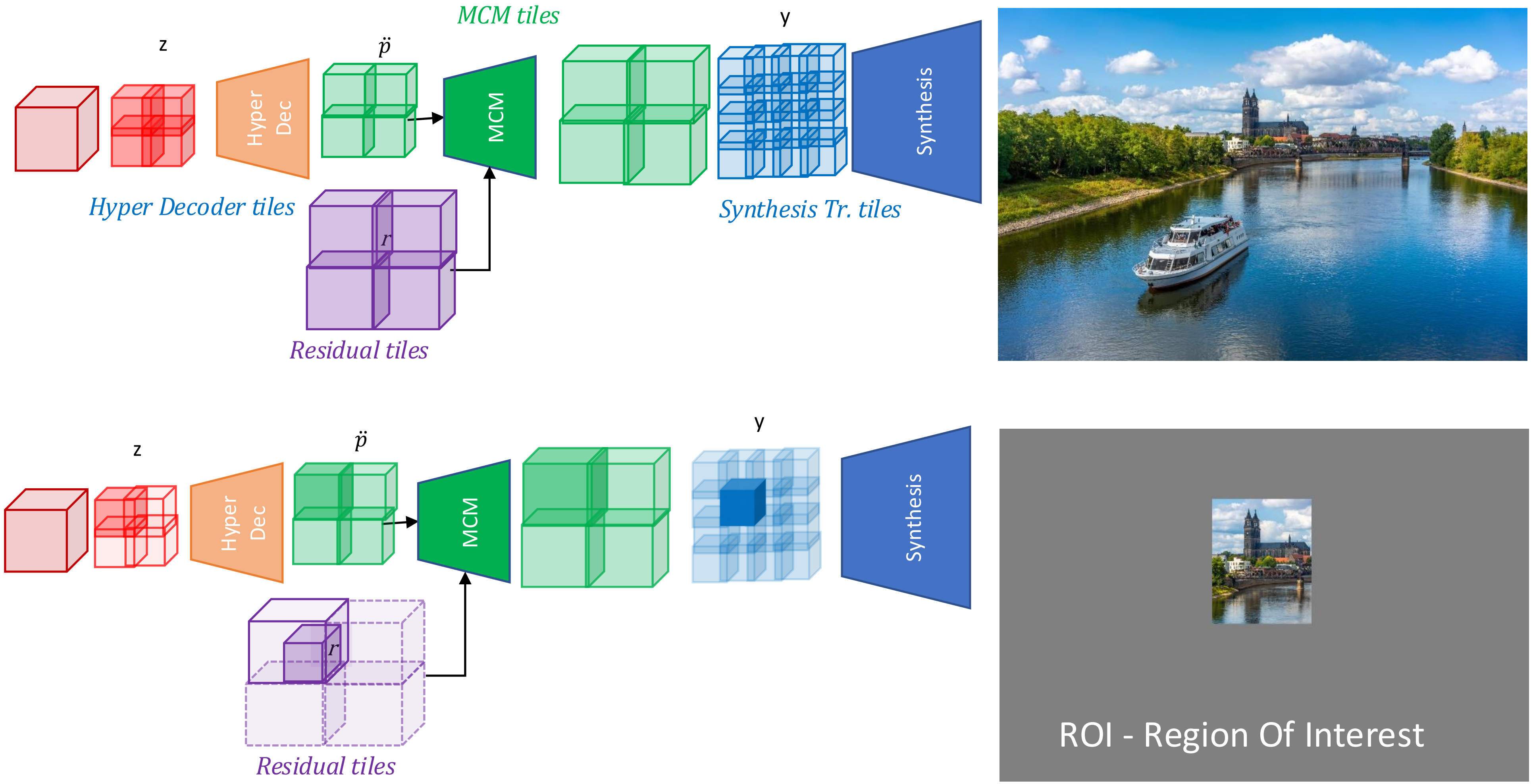
mandatory | optional | partitioned



partitioned code stream segment

Marker	*Length*	*N tiles*	*TilesLengths*	*Tiles0 data*	…	*N-1 Tile data*

Latent domain tiles and ROI decoding



Random access inside picture is possible ***after*** ***<u>parital</u> residual*** parsing

me-tANS: memory efficient tabulated Asymmetric Numeral Systems

Just one specific variant of ANS

Encoding / decoding

Alice send 'AAABA' to Bob with fewest bits

Basic method

Alice

Encode: AAABA → 00 00 00 01 00

Symbol	Code
A	00
B	01
C	10
D	11

Shared knowledge

0000000100 (10 bits)

Bob

Decode: 00 00 00 01 00 → AAABA

Improved

Alice

Encode: AAABA → 0 0 0 10 0

Symbol	Code
A	**0**
B	**10**
C	**110**
D	**111**

Shared knowledge

000100 (6 bits)

Bob

Decode: 0 0 0 10 0 → AAABA

Basic idea: higher frequency → less bits

Entropy

Optimal expected code length: **Entropy**

$$H(X) = -\sum_i p_i \log_2 p_i$$

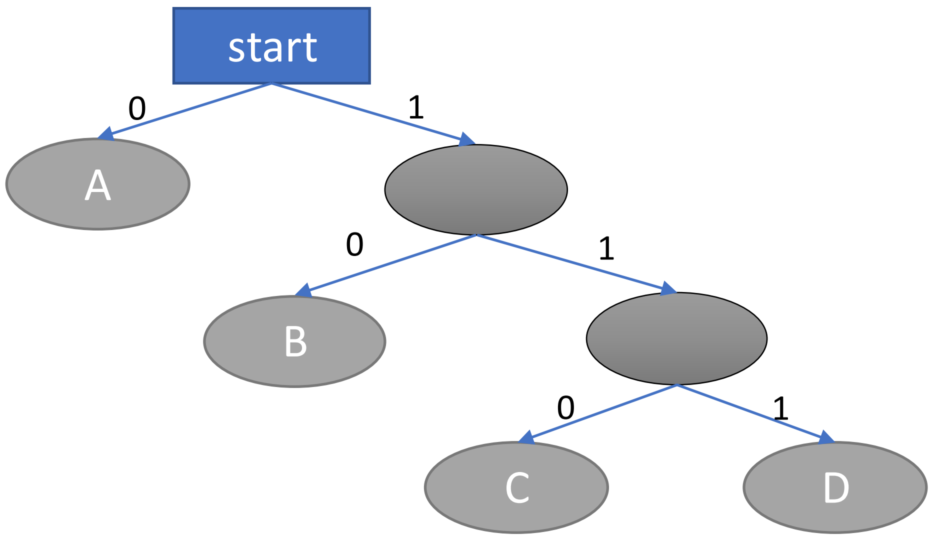
Optimal way:

$p_i \rightarrow -\log_2 p_i$ bits

Closer to optimal → shorter code

Symbol	Prob	Method 1 bits	Method 2 bits	Optimal bits
A	0.7	2	1	0.51
B	0.1	2	2	3.32
C	0.1	2	3	3.32
D	0.1	2	3	3.32
Expected		**2**	**1.5**	**1.36**

Huffman tree



Each symbol requires integer number of bits

Can we do better?

Asymmetric Numeral Systems (ANS)

Before encoding / decoding

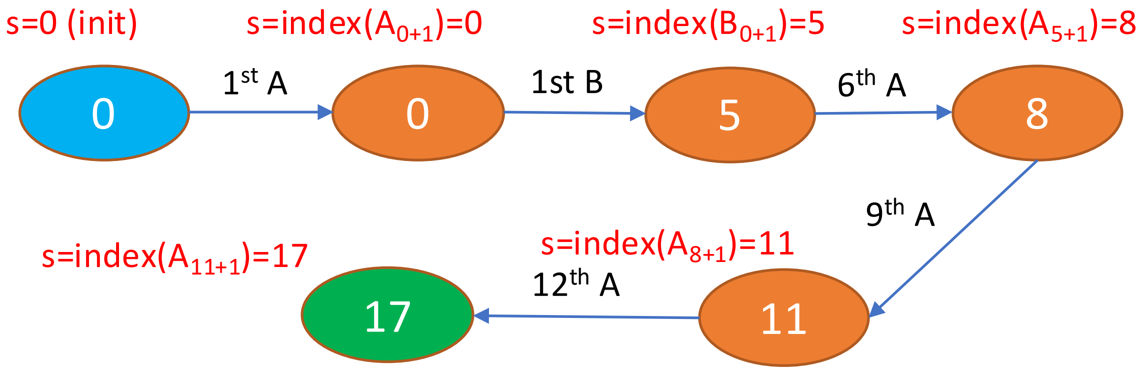
1. Quantize probability
2. Construct table (infinite)

	Prob	quantized
A	0.7	**5/8**
B	0.1	**1/8**
C	0.1	**1/8**
D	0.1	**1/8**

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**...**
Symbol	A_1	A_2	A_3	A_4	A_5	B_1	C_1	D_1	A_6	A_7	A_8	A_9	A_{10}	B_2	C_2	D_2	A_{11}	A_{12}	...

Encode:

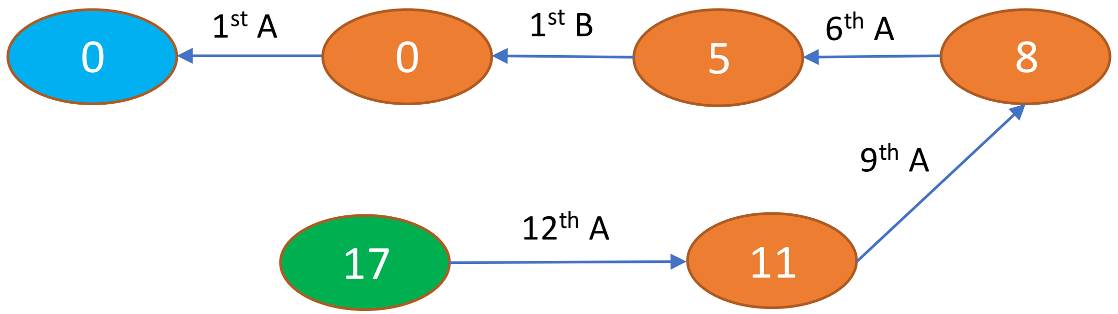
- Init s = 0
- To encode x: s → index of (s+1)th x



Encode AAABA (reversed), codeword=10001 (17)

Decode s:

- x: Index s → which symbol
- s → #x before s



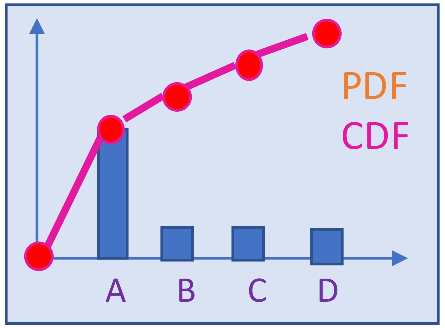
Decode 10001 (17) to AAABA

Ranged ANS (rANS)

Before encoding / decoding

1. Quantize probability (denominator M)
2. Construct table (infinite)
3. Calculate PDF (P) & CDF (C)

	Prob	quantized	PDF (P)	CDF (C)
A	0.7	5/8	**5**	**0**
B	0.1	1/8	**1**	**5**
C	0.1	1/8	**1**	**6**
D	0.1	1/8	**1**	**7**



Encode x:

$$s \leftarrow \lfloor s/PDF(x) \rfloor * M + CDF(x) + s \bmod PDF(x)$$

x	Calculation	s
		0
A	0/5*8 + 0 + 0%5	0
B	0/1*8 + 5 + 0%5	5
A	5/5*8 + 0 + 5%5	8
A	8/5*8 + 0 + 8%5	11
A	11/5*8 + 0 + 11%5	17

Encode AAABA (reversed), codeword=10001 (17)

Decode from s:

$$x \leftarrow CDF(x) \leq s \bmod M < CDF(x) + PDF(x)$$
$$s \leftarrow \lfloor s/M \rfloor * PDF(x) + s \bmod M - CDF(x)$$

s	s mod M	x	Calculation (new s)
17	1	A	17/8*5 + 17%8 – 0
11	3	A	11/8*5 + 11%8 – 0
8	0	A	8/8*5 + 8%8 – 0
5	5	B	5/8*5 + 5%8 – 5
0	0	A	

Streamed rANS

Problem with original rANS:

$s \leftarrow \lfloor s/PDF(x) \rfloor * M + CDF(x) + s \bmod PDF(x)$

Long message → s veeeeeeeery large

- s: Big Integer
- **Time complexity: O(n²)**

In real use:

- Codeword: $s \in [2^i, 2^{2i})$, and memory (stack)
- When $s \geq 2^{2i}$, push lower i bits to memory
- Larger i → shorter codeword
- Usually, i = 16 or 32 (s fit in uint32/uint64)
- Init: $s \leftarrow 2^i$, memory ← empty

Encode x:

if $s * M \geq PDF(x) * 2^{2i}$

- memory.push(s % 2^i)
- $s \leftarrow s/2^i$

$s \leftarrow \lfloor s/PDF(x) \rfloor * M + CDF(x) + s \bmod PDF(x)$

Decode from s:

$x \leftarrow CDF(x) \leq s \bmod M < CDF(x) + PDF(x)$

$s \leftarrow \lfloor s/M \rfloor * PDF(x) + s \bmod M - CDF(x)$

if $s < 2^i$

- $s \leftarrow s * 2^i$ + memory.pop()

	Prob	quantized	PMF (P)	CDF (C)
A	0.7	5/8	**5**	**0**
B	0.1	1/8	**1**	**5**
C	0.1	1/8	**1**	**6**
D	0.1	1/8	**1**	**7**

Tabled ANS (tANS)

Streamed rANS
- O(n), much faster than original rANS
- < 100 MB/s
  - Encoding: division/modulo
  - Decoding: binary search

Tabled ANS (tANS)
- **Minimize calculation with table**

(*Optional) Table construction:
- (Decoding) Build s_enc → x
- (Encoding) Each s + x → s_enc
  - (s_enc, pushed) different
  - Same s_enc → same nbits
  - Same x → nbits try to be similar
- (Decoding) Other parts consistent with encoding

	Prob	quantized	PMF (P)	CDF (C)
A	0.7	5/8	**5**	**0**
B	0.1	1/8	**1**	**5**
C	0.1	1/8	**1**	**6**
D	0.1	1/8	**1**	**7**

Encode x to s:
- memory.push(getPushed(s,x), getBits(s,x))
- s ← getSEnc(s,x)

x	s	nbits	pushed	S_enc
A	0	0		0
	1	0		1
	2	1	0	2
	3	1	1	2
	4	1	0	3
	5	1	1	3
	6	1	0	4
	7	1	1	4
B,C,D	0	3	000	B → 5 C → 6 D → 7
	1	3	001	
	2	3	010	
	3	3	011	
	4	3	100	
	5	3	101	
	6	3	110	
	7	3	111	

Table for encoding

Decode s_enc:
- x ← getX(s_enc)
- poped ← memory.pop(getBits(s_enc))
- s ← getS(s_enc, poped)

s_enc	x	nbits
0	A	0
1	A	0
2	A	1
3	A	1
4	A	1
5	B	3
6	C	3
7	D	3

s_enc	popped	s
0		0
1		1
2	0	2
	1	3
3	0	4
	1	5
4	0	6
	1	7
5,6,7	000	0
	001	1
	010	2
	011	3
	100	4
	101	5
	110	6
	111	7

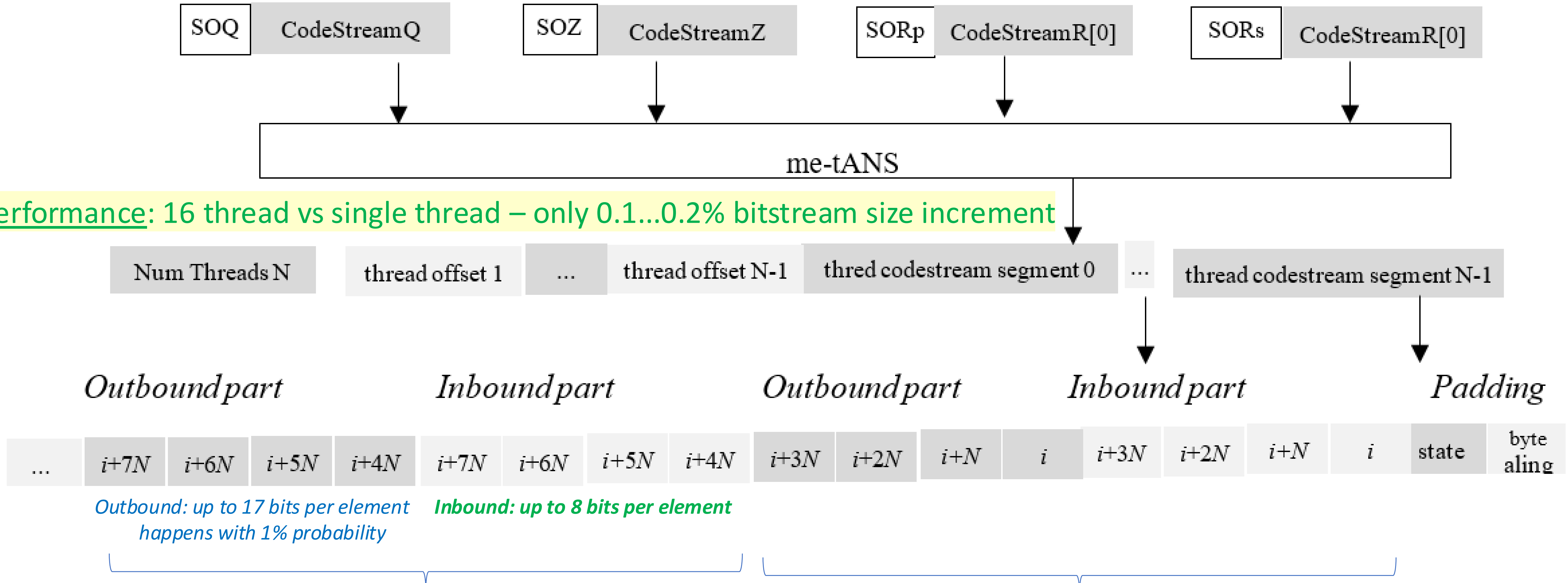
Table for decoding

ANS - a new class of entropy coders

ANS		Algorithm	operations	memory
streamed rANS	Encoder	**if** $s * M \geq P(x) * 2^{2i}$ • memory.push(s % 2^i) • $s \leftarrow s/2^i$ $s \leftarrow \lfloor s/P(x) \rfloor * M + C(x) + s \, mod \, P(x)$	1~2 div / mod, 1 branch Several add, bit op	O(1)
tANS		memory.push(getPushed(s,x), getBits(s,x)) s ← getSEnc(s,x)	3 table lookups, 2 bit ops, 1 add	O(XM)
me-tANS		$x \leftarrow \Theta[s]$ $s \leftarrow N[s]$ + memory.pop($B[s]$)	2 table lookups, 7 bit ops, 3 adds	O(M)
streamed rANS	Decoder	$x \leftarrow C(x) \leq s \, mod \, M < C(x) + P(x)$ $s \leftarrow \lfloor s/M \rfloor * P(x) + s \, mod \, M - C(x)$ **if** $s < 2^i$ $s \leftarrow s * 2^i$ + memory.pop()	1 binary search → O(log X) 1 branch, 1 mult Several adds, bit ops	O(1)
tANS		x ← getX(s_enc) poped ← memory.pop(getBits(s_enc)) s ← getS(s_enc, poped)	3 table lookups, 3 bit ops, 1 add	O(M²)
me-tANS		$x \leftarrow \Theta[s]$ $s \leftarrow N[s]$ + memory.pop($B[s]$)	1 table lookups, 5 bit ops, 1 adds	O(M)

- https://kedartatwawadi.github.io/post--ANS/

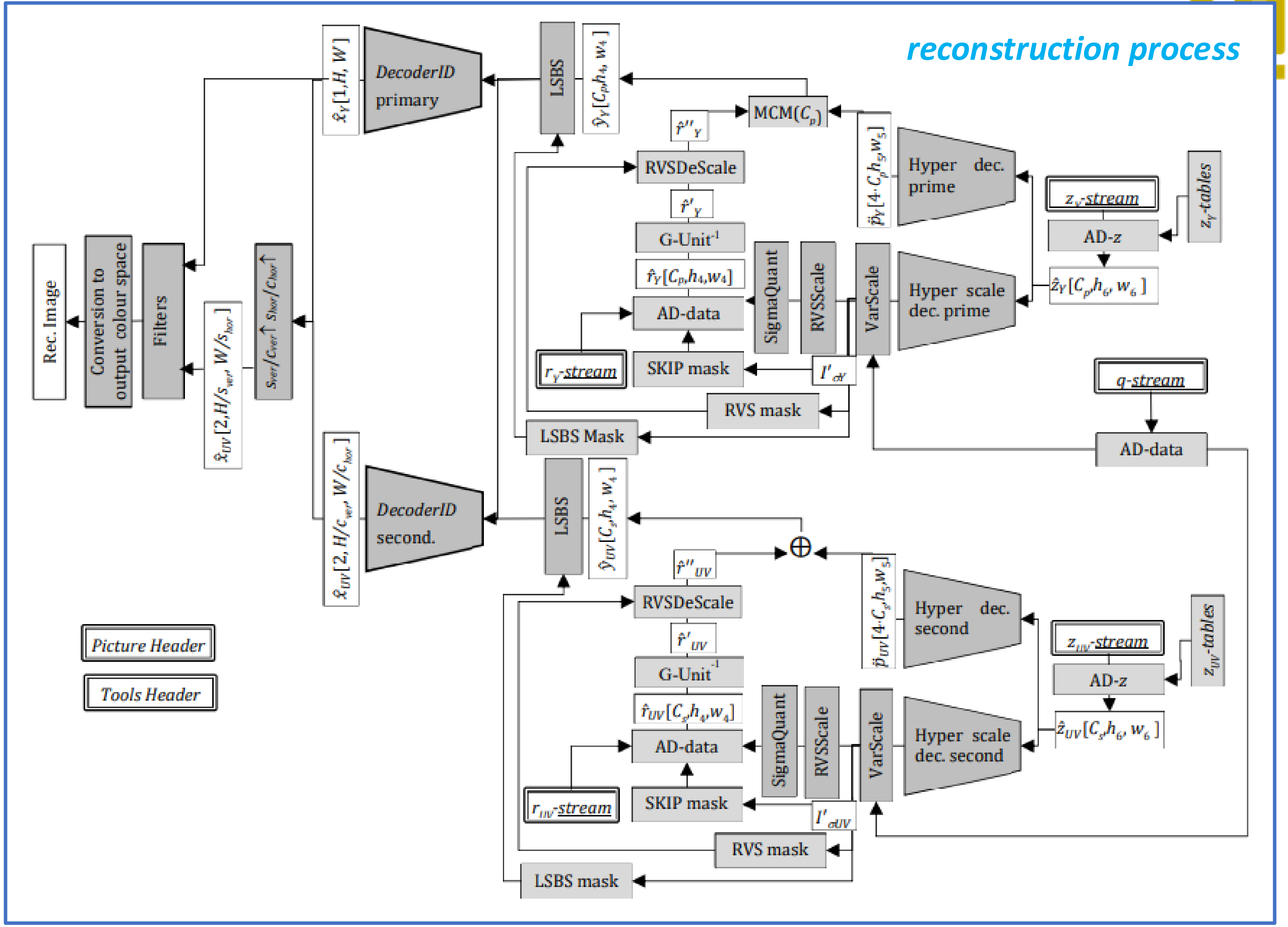
Multi-thread coding with me-tANS



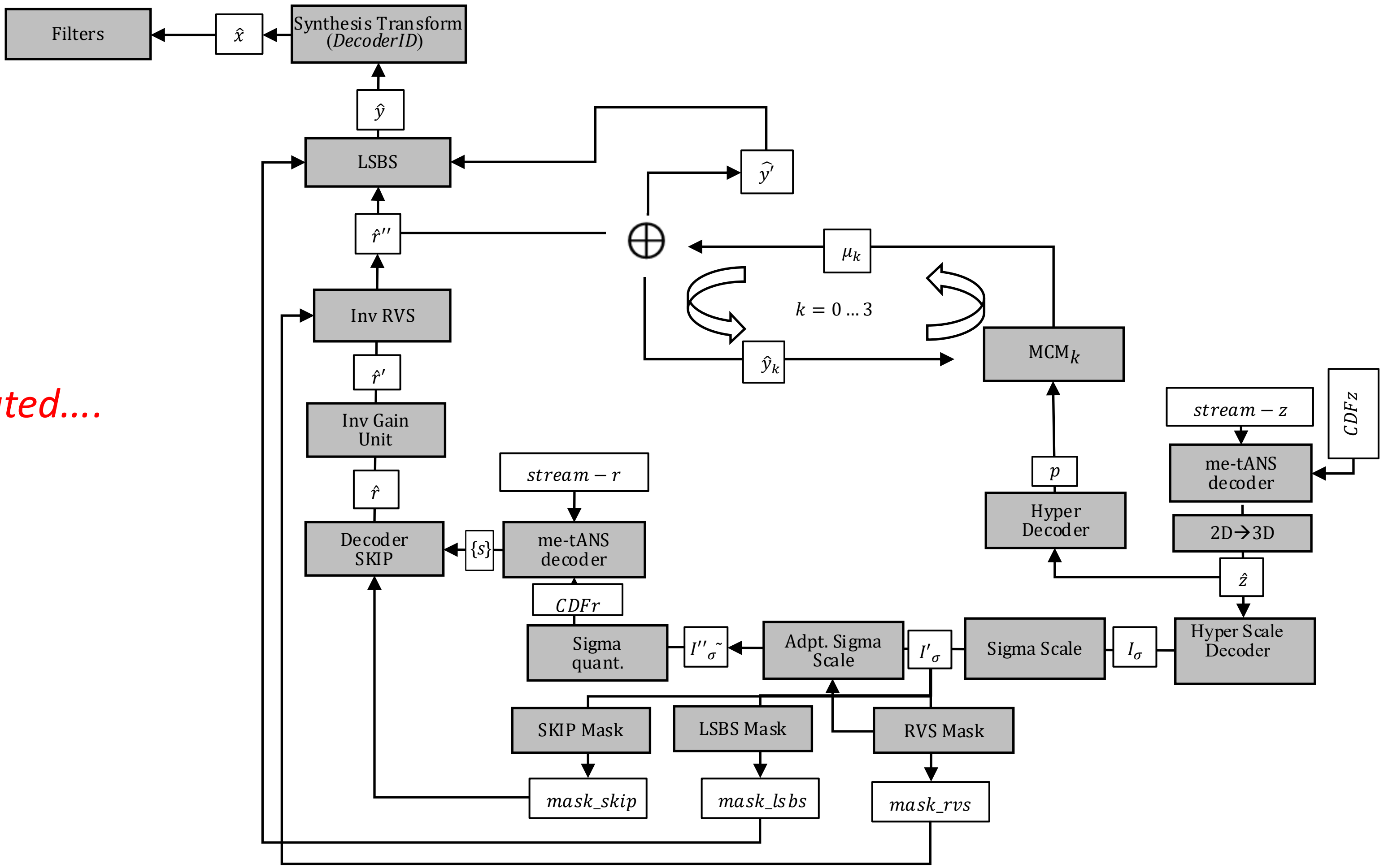
Performance: 16 thread vs single thread – only 0.1...0.2% bitstream size increment

4 elements in each ‚round‘ , inboudn part can be efficiently ‚packed‘ into 32 bits register

JPEG AI decoder



Decoding process for one component

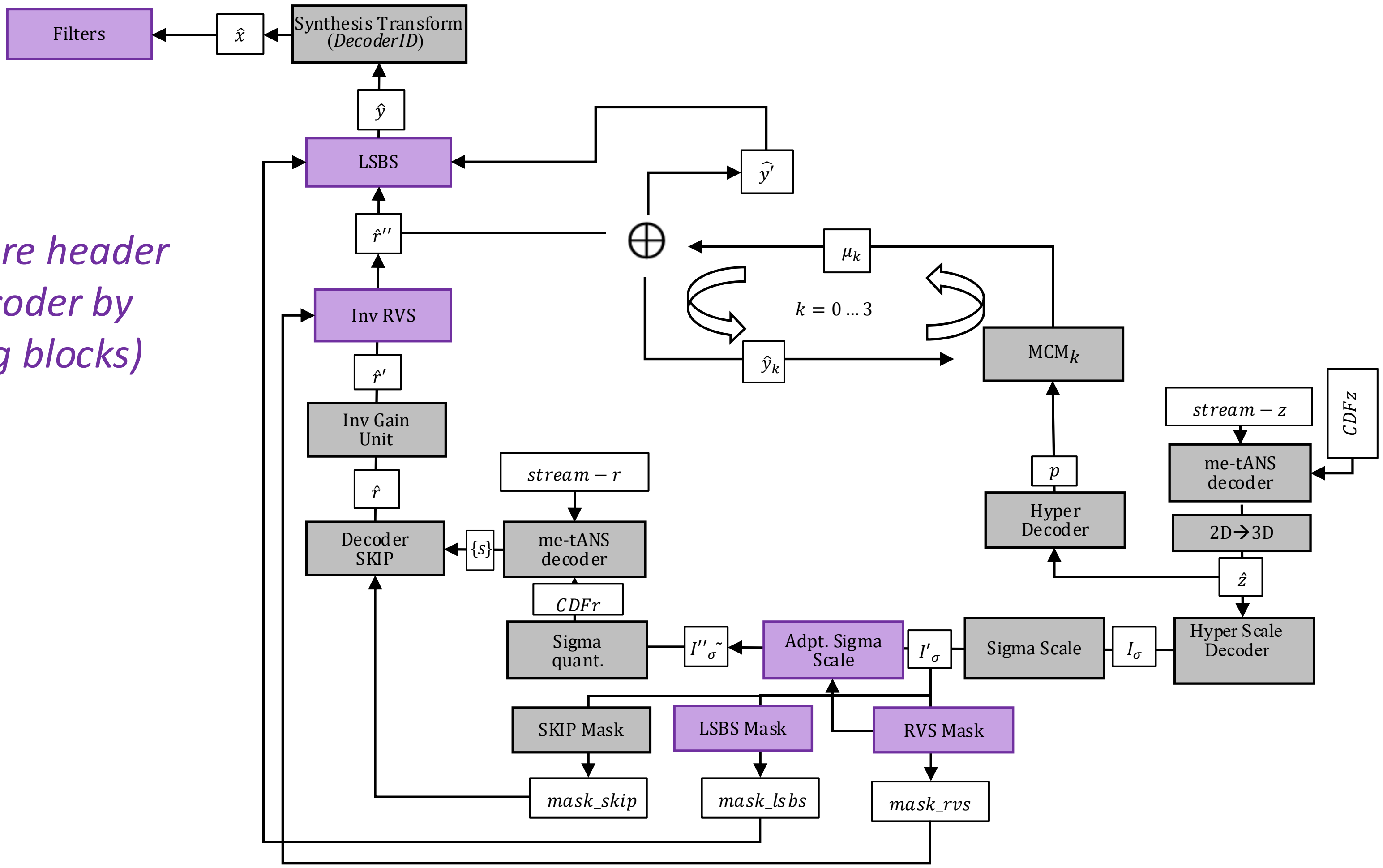


...looks complicated....

let's explain!!!

Optional tools

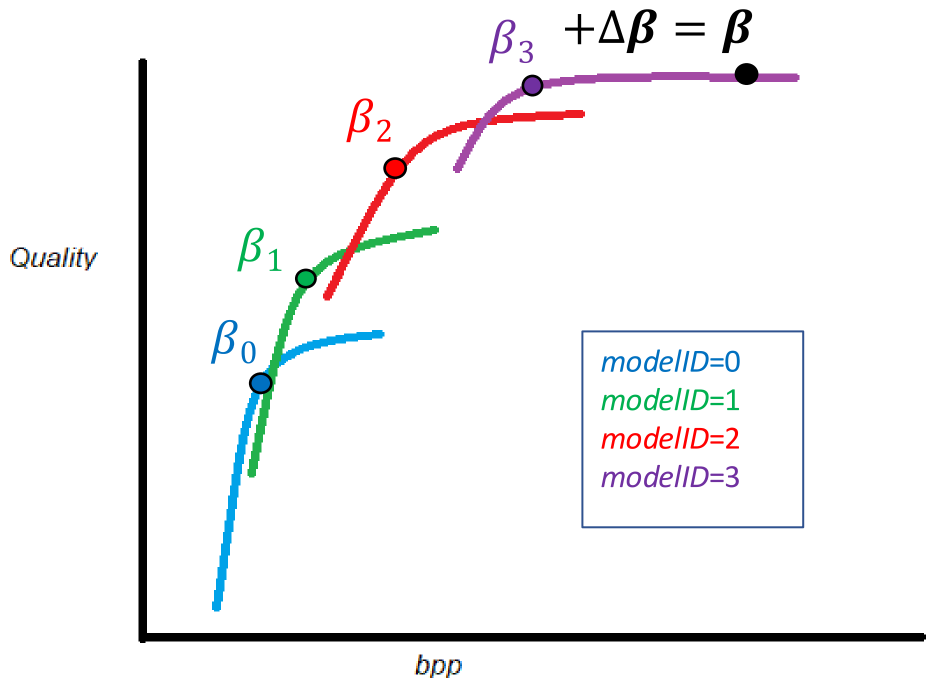
Enabled by flag in picture header (if disabled then decoder by passes corresponding blocks)

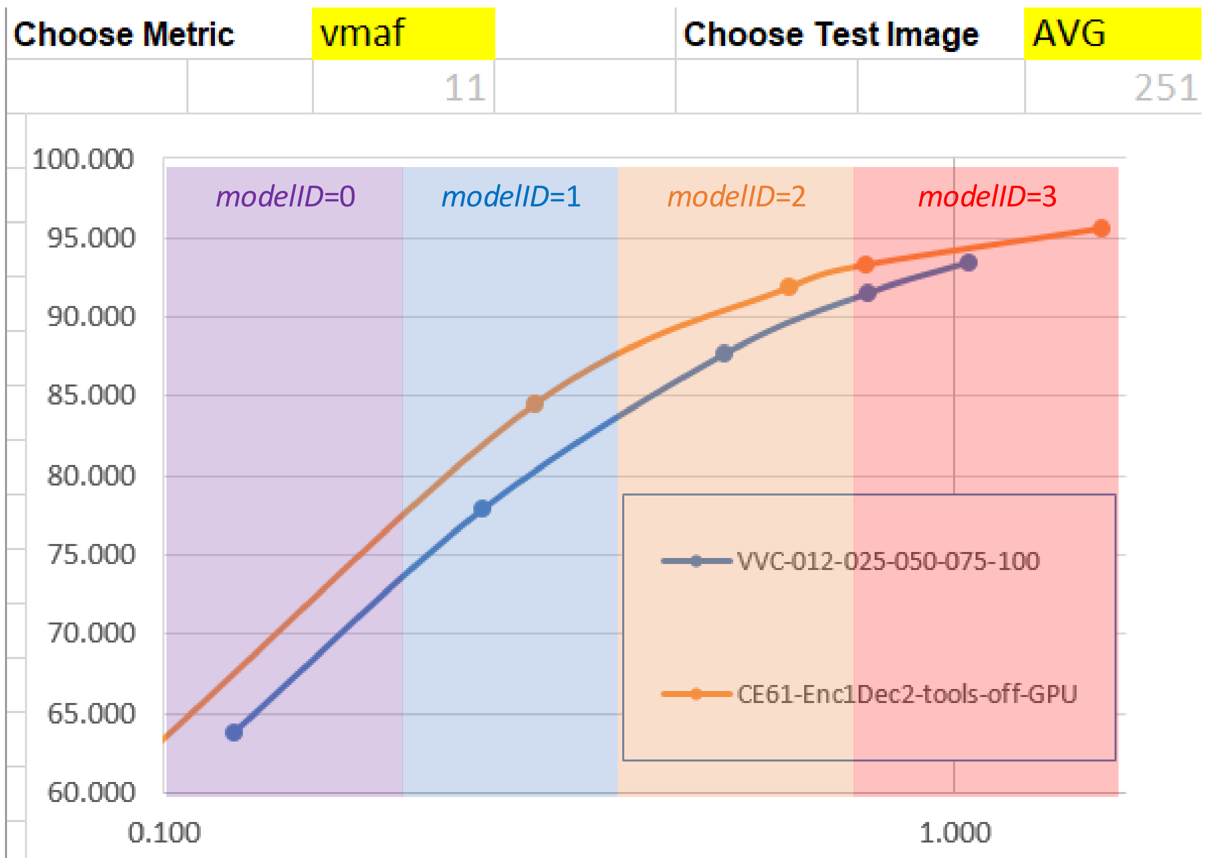


Variable rate coding

Not only Rate control

Variable rate coding





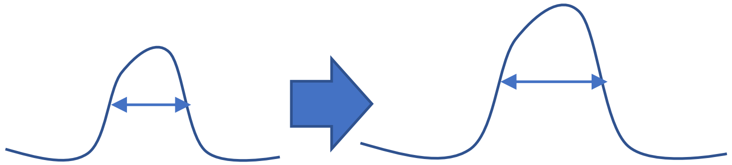
Reciever have to support:

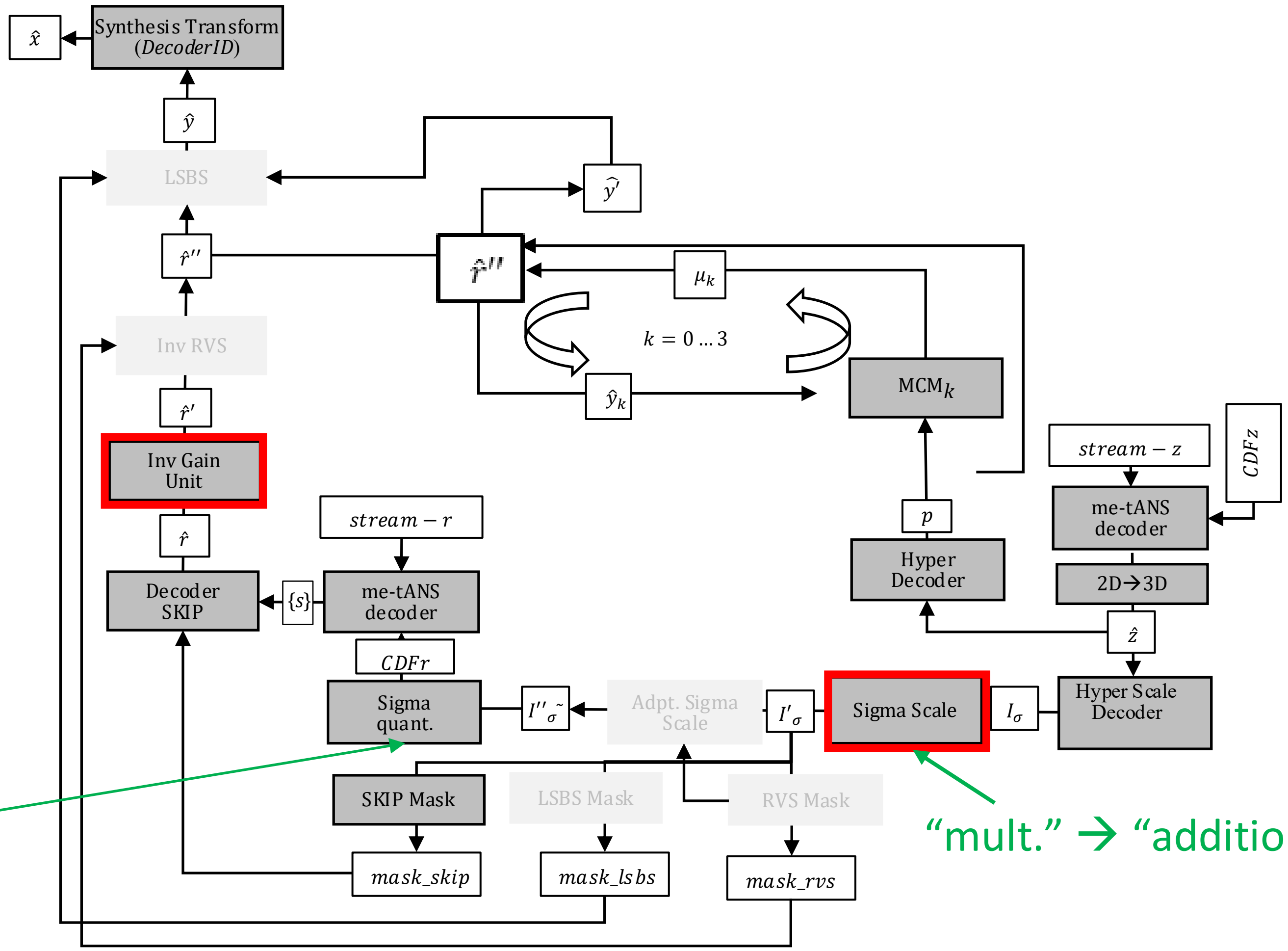
4 sets of model parameters * **1** decoder NN structures

Variable rate coding

Model parameters defined by ***modelID*** (***signaled***)

Gain Unit == "Per channel gain" factor for residual & variance (**pretrained**), gain defined by $\Delta\boldsymbol{\beta}$ (***signaled***)



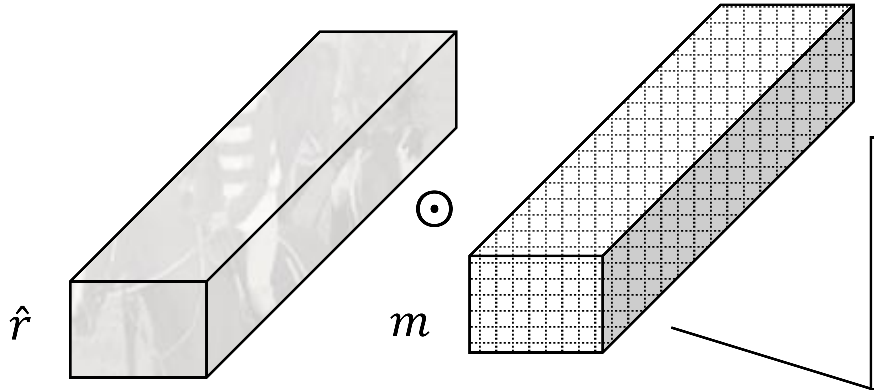


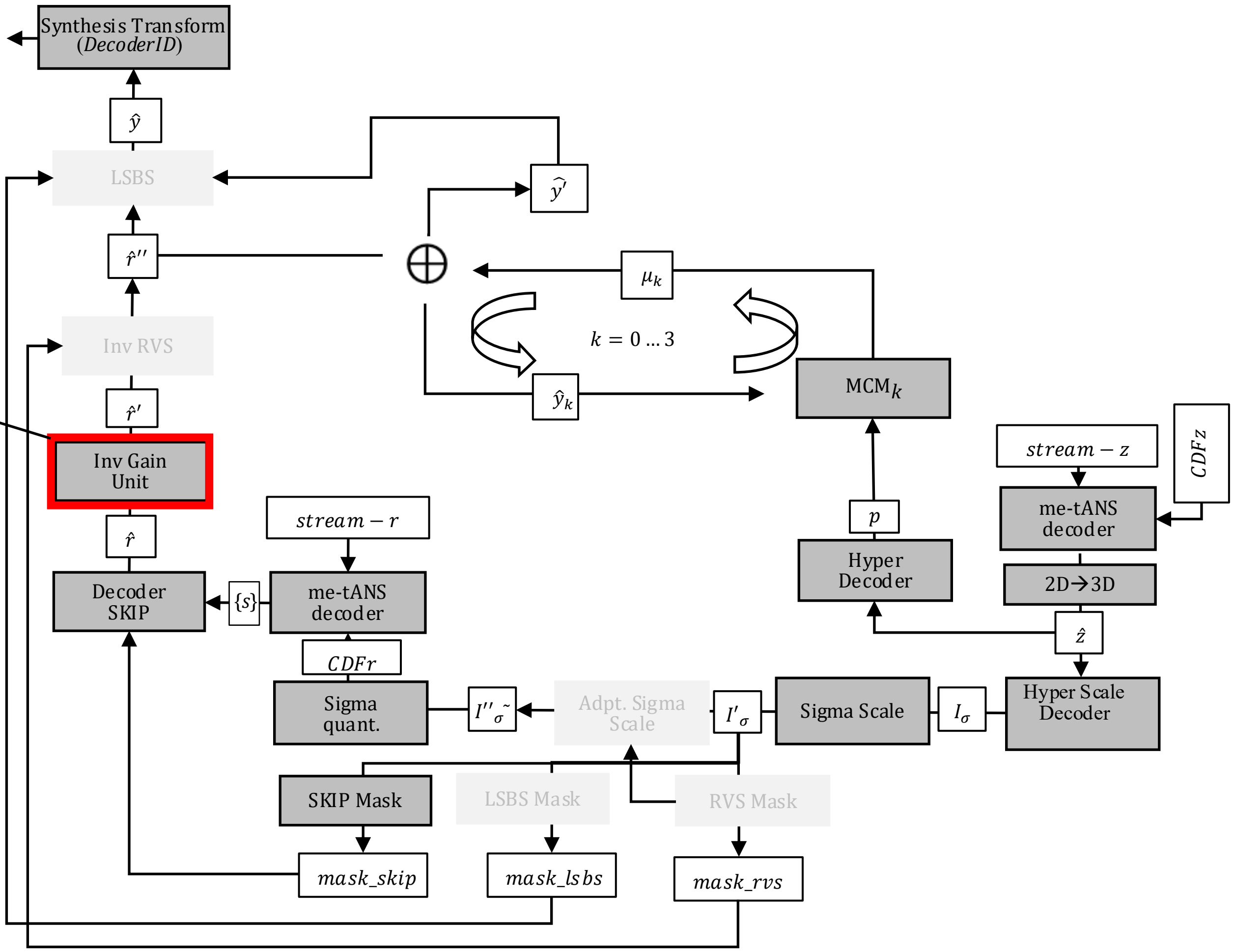
Hint: Hyper scale decoder outputs I_σ - log domain "σ"

LUT search → rounding

"mult." → "addition"

Variable rate coding ++





$m[c,i,j] = m_{modelID}[c,:,:]$

- $\Delta\beta$ ~ *per picture "QP"*
- $gfrs[c]$ ~*per channel amplifier*
- $3Dgain[c,i,j]$ ~ *local quality control*

optional, selected by encoder

How Q-map helps?

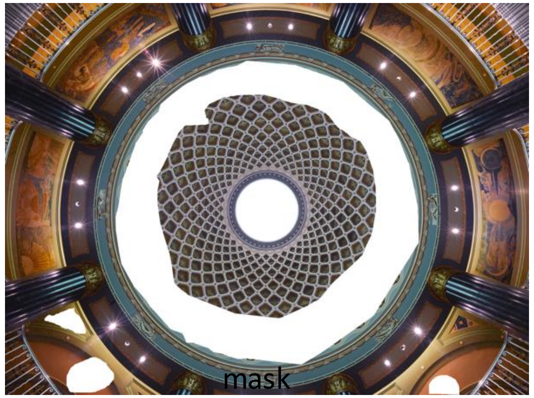
W/o Q-map

Original:

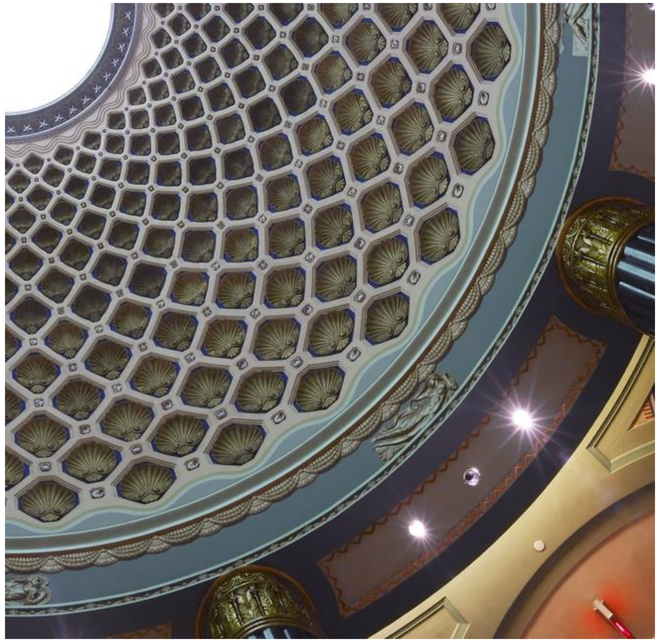
DecoderID = 1 *DecoderID* = 2

W/ Q-map

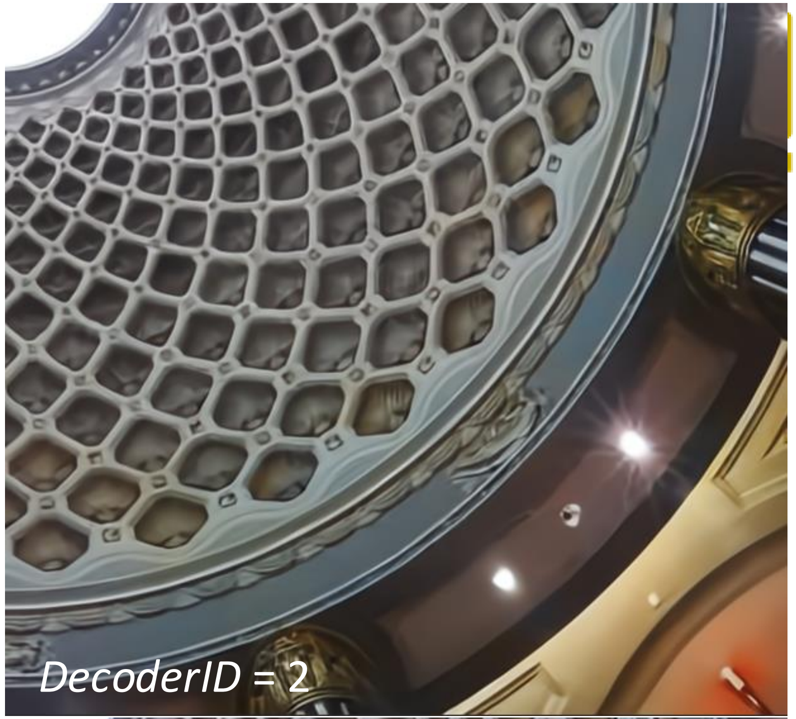
W/o Q-map



Original:



W/ Q-map



Entropy network

Ensures bit-exact behavior

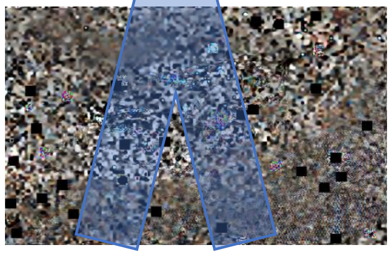
Device interoperability

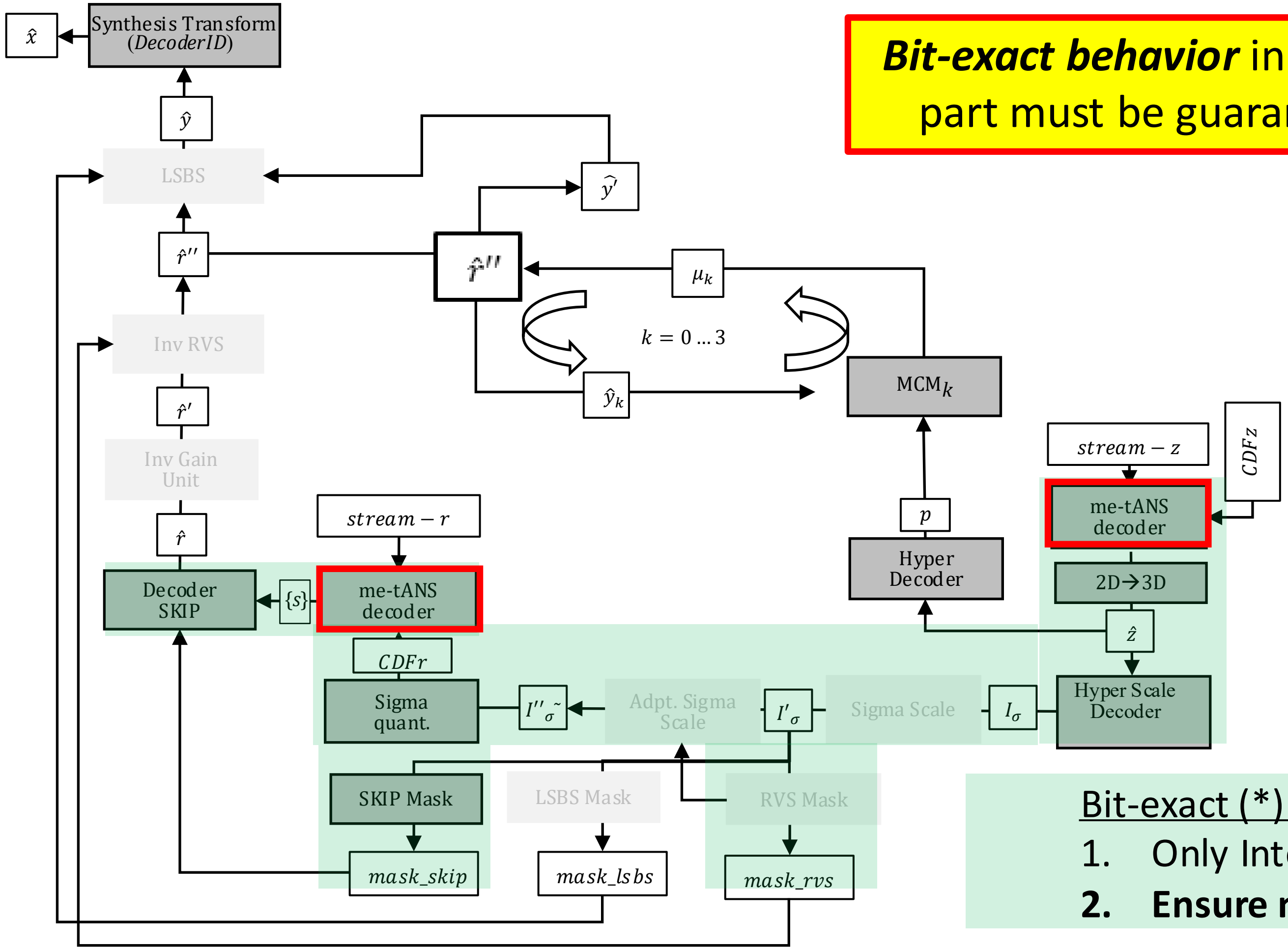
Even ***minor error*** in CFD leads to ***wrong interpretation*** of parsed symbol in arithmetic coder.

How does effect look like?

Encoded decoded on same device

Encoded decoded on different devices





Bit-exact behavior in entropy part must be guaranteed!

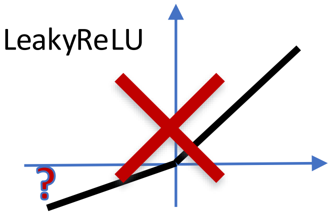
Bit-exact (*):

1. Only Integer arithmetic
2. **Ensure no-overflow**

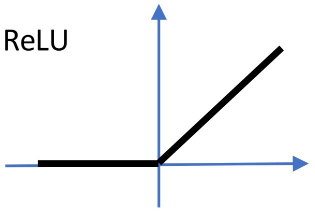
Entropy network with bit-exact behavior

$\hat{z}[C, h_6, w_6]$
$qCONV(1{\times}1, C, C, d_1, p_1)$
$ReLU()$
$qCONV(3{\times}3, C, C, d_2, p_2)$
$ReLU()$
$qCONV(1{\times}1, C, 4{\cdot}4{\cdot}C, d_3, p_3)$
$Shuffle\ (4,4)$
$Crop(\ h_4, w_4)$
$abs()$
$clip(0,\ ((N_\sigma - 1) \ll sigmaPrecision)) - 1)$
$I_\sigma[C, h_4, w_4]$

LeakyReLU



ReLU



convolution layer $CONV$

…

$$out[c_{out}, i, j] = bias[c_{out}] + \sum_{c_{in}=0}^{C_{in}} weigth[c_{in}, c_{out}] \star input[c_{in}, s \cdot i, s \cdot j];$$

$$i = 0, \dots, h_{out} - 1; j = 0, \dots, w_{out} - 1; c_{out} = 0, \dots, C_{out} - 1$$

where "$\star$" is 2D **cross-correlation operator** with kernel size $K_{ver} \times K_{hor}$

….

quantized convolution layer $qCONV$

… three-steps operation:

$$temp[c_{in}, i, j] = clip(-d, d-1, input[c_{in}, i, j]),$$
$$i = 0, \dots, h_{in} - 1; j = 0, \dots, w_{in} - 1; c_{in} = 0, \dots, C_{in} - 1;$$

$$R[c_{out}, i, j] = bias[c_{out}] + \sum_{c_{in}=0}^{C_{in}-1} weigth[c_{in}, c_{out}] \star temp[c_{in}, s \cdot i, s \cdot j];$$

where "$\star$" is 2D **cross-correlation operator** with kernel size $K_{ver} \times K_{hor}$.

$$out[c_{out}, i, j] = (R[c_{out}, i, j]) \gg p[c_{out}];$$
$$i = 0, \dots, h_{out} - 1; j = 0, \dots, w_{out} - 1; c_{out} = 0, \dots, C_{out} - 1.$$

The tensor $weigth$ of shape $[C_{in}, C_{out}, K_{ver}, K_{hor}]$ contains learnable ***8-bit integer*** weights, the tensor $bias$ of shape $[C_{out},]$ contains learnable ***31-bit integer*** biases. All parameters $weigth$ and $bias$ are part of learnable quantized model.
The combination of clipping value *d*, de-scaling shifts $p[c_{out}]$ and magnitude for the quantized model parameters allows control over bit depth of register $R[c_{out}, i, j]$ (guaranteed to be within ***32 bits***).
…

Overflow aware model quantization

- $temp[c_{in}, i, j] = clip(-d, d-1, input[c_{in}, i, j]),\ i = 0, \dots, h_{in}-1; j = 0, \dots, w_{in}-1; c_{in} = 0, \dots, C_{in}-1;$
  - $\boxed{R[c_{out}, i, j] = bias[c_{out}] + \sum_{c_{in}=0}^{C_{in}-1} weigth[c_{in}, c_{out}] \star temp[c_{in}, s \cdot i, s \cdot j];}$ where "$\star$" is 2D **cross-correlation operator** with kernel size $K_{ver} \times K_{hor}$.
    - $out[c_{out}, i, j] = (R[c_{out}, i, j]) \gg p[c_{out}];\ i = 0, \dots, h_{out}-1; j = 0, \dots, w_{out}-1; c_{out} = 0, \dots, C_{out}-1.$

Overflow can happen here

$$R = \sum_{n=1}^{N_{param}} model_n \cdot temp_n$$

$$Given: d \le temp \le d \quad To\ find: \min(R) =?\ \max(R) =?$$

$$\max(R) = \sum_{n=1}^{N_{param}} (model_n > 0?\ (model_n \cdot d) : (-model_n \cdot d) = d \sum_{n=1}^{N_{param}} |model_n|$$

$$\min(R) = \sum_{n=1}^{N_{param}} (model_n > 0?\ (-model_n \cdot d) : (model_n \cdot d) = -d \sum_{n=1}^{N_{param}} |model_n|$$

$$|R| \le d \sum_{n=1}^{N_{param}} |model_n|$$

For example, $d = 2^{16}$ and required $|R| < 2^{32} \implies \sum_{n=1}^{N_{param}} |model_n| < 2^{32}/d$

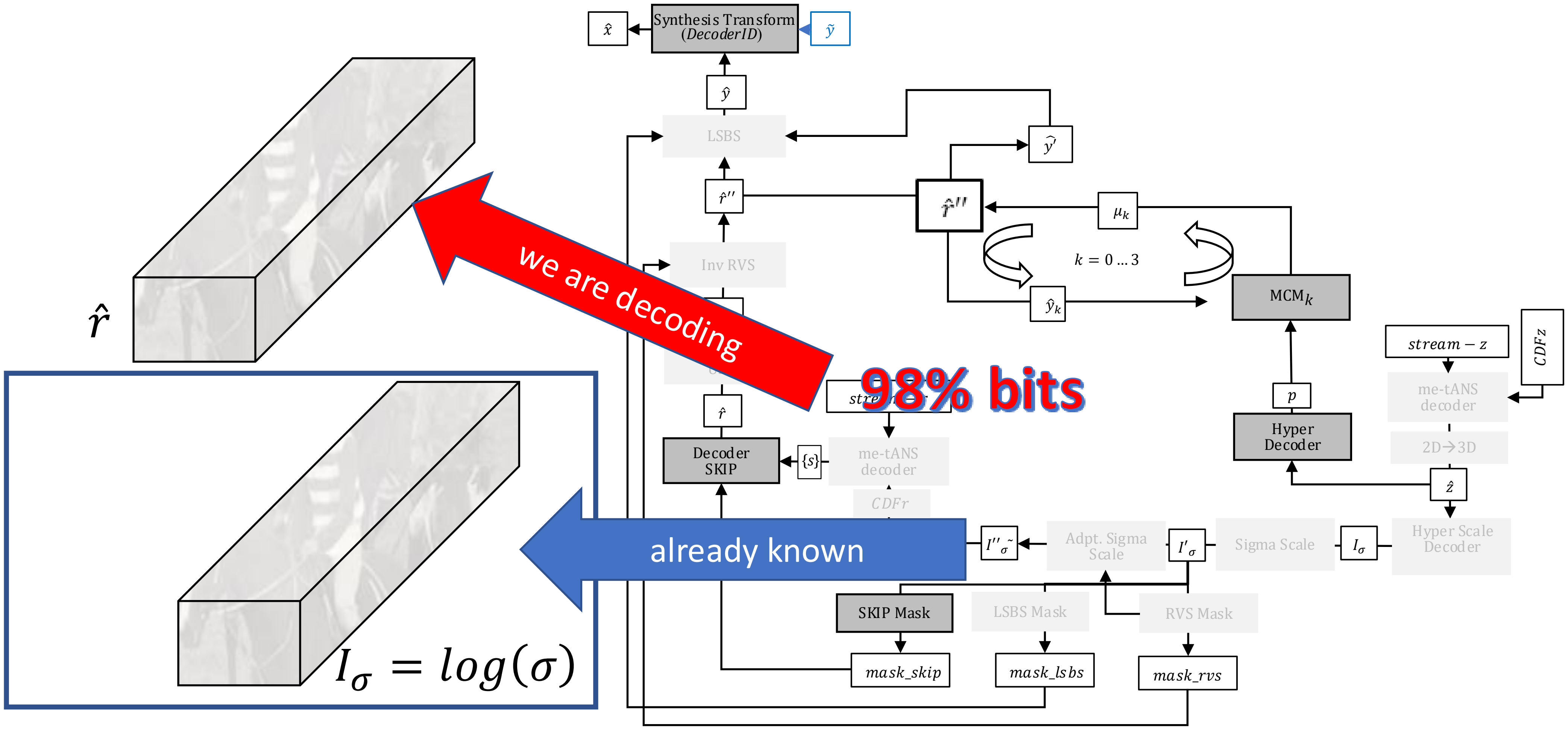
$$\boxed{\sum_{n=1}^{N_{param}} |model_n| < 2^{16}}$$

Training procedure

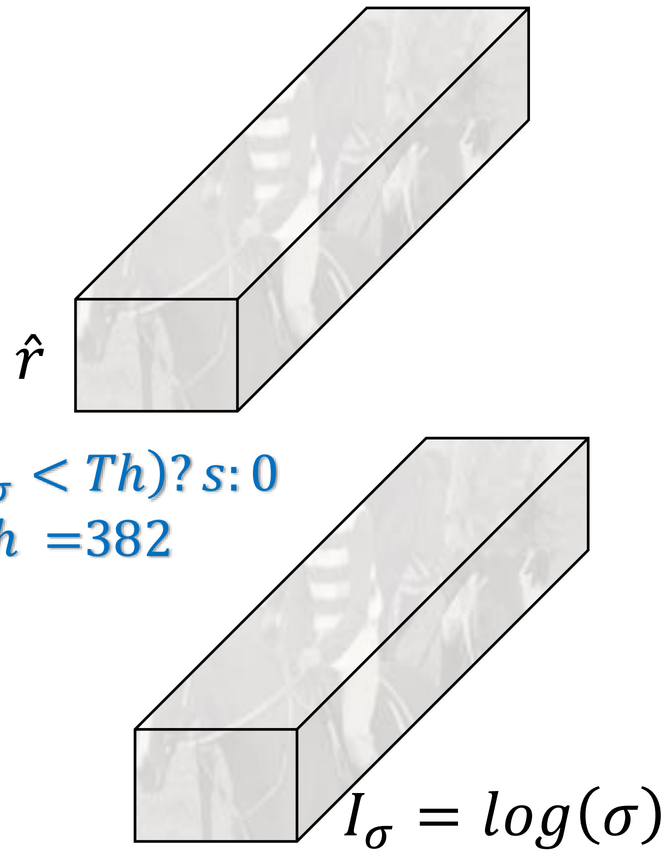
1. MSE-loss; all modules; fixed rate
2. ***Mixed loss***; all modules; fixed rate
   - Mixed distortion = (1.0 – *a*) (mse_loss_Y + mse_loss_U + mse_loss_V)+ *a* * 1000 * mssim_loss_Y
   - *a* = 0.5 (lossy coding), *a* = 0.25 (nearly lossless quality)
3. Mixed loss; ***entropy part only***; fixed rate
4. Mixed loss; all module; ***variable rate***
5. ***Quantize*** of entropy sub-network (device interoperability)
6. ***Reorder*** channels (progressive decoding)
7. Encoder ***dynamic*** range control (artifacts prevention)

Residual coding

Help arithmetic coder

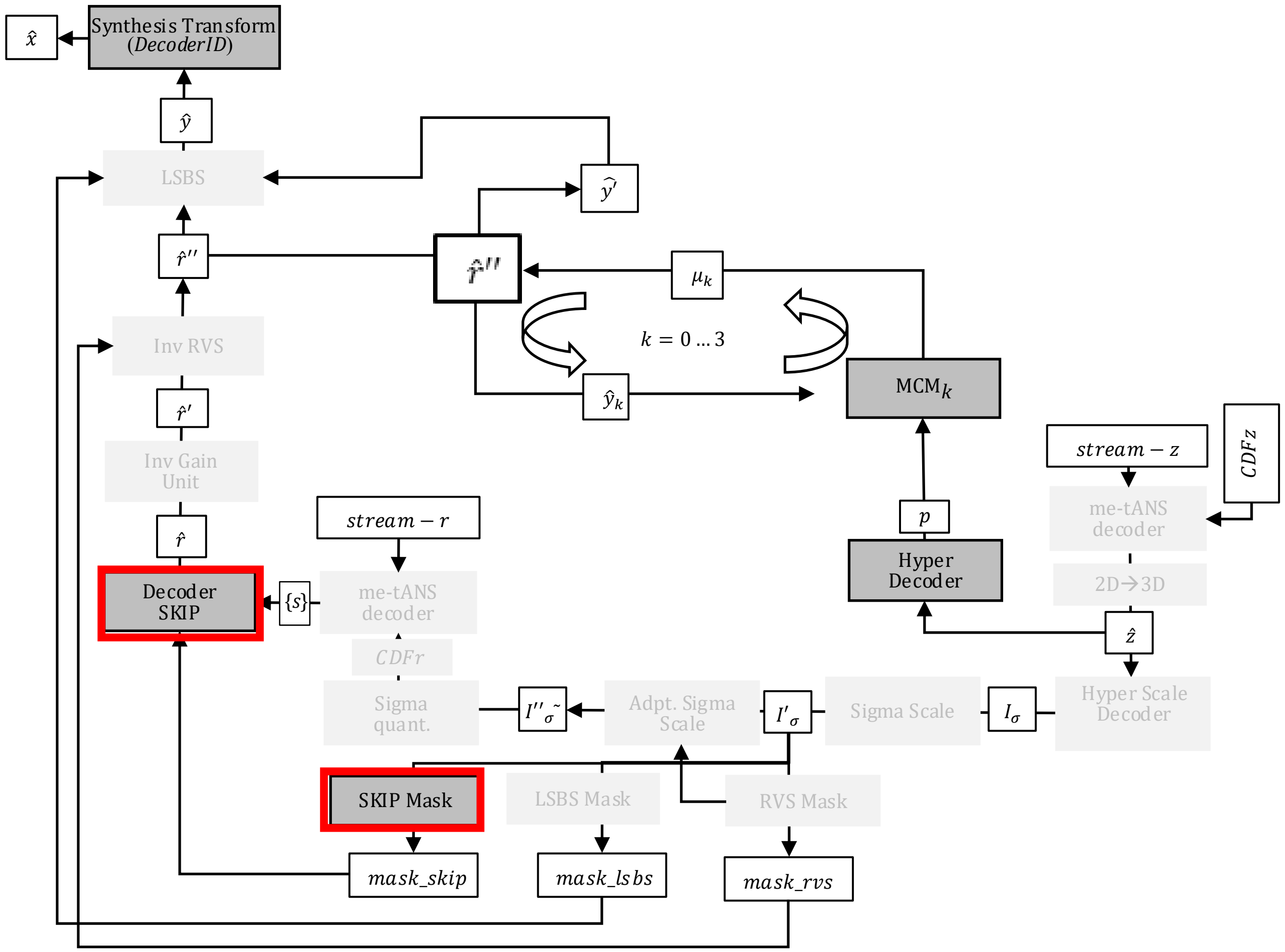


Help arithmetic coder

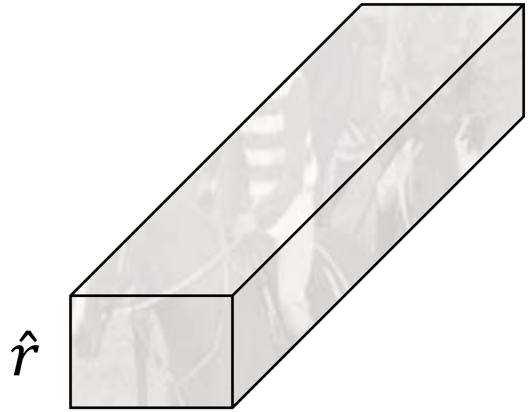


$\hat{r} = (I_\sigma < Th)?\, s : 0$

$Th\ = 382$



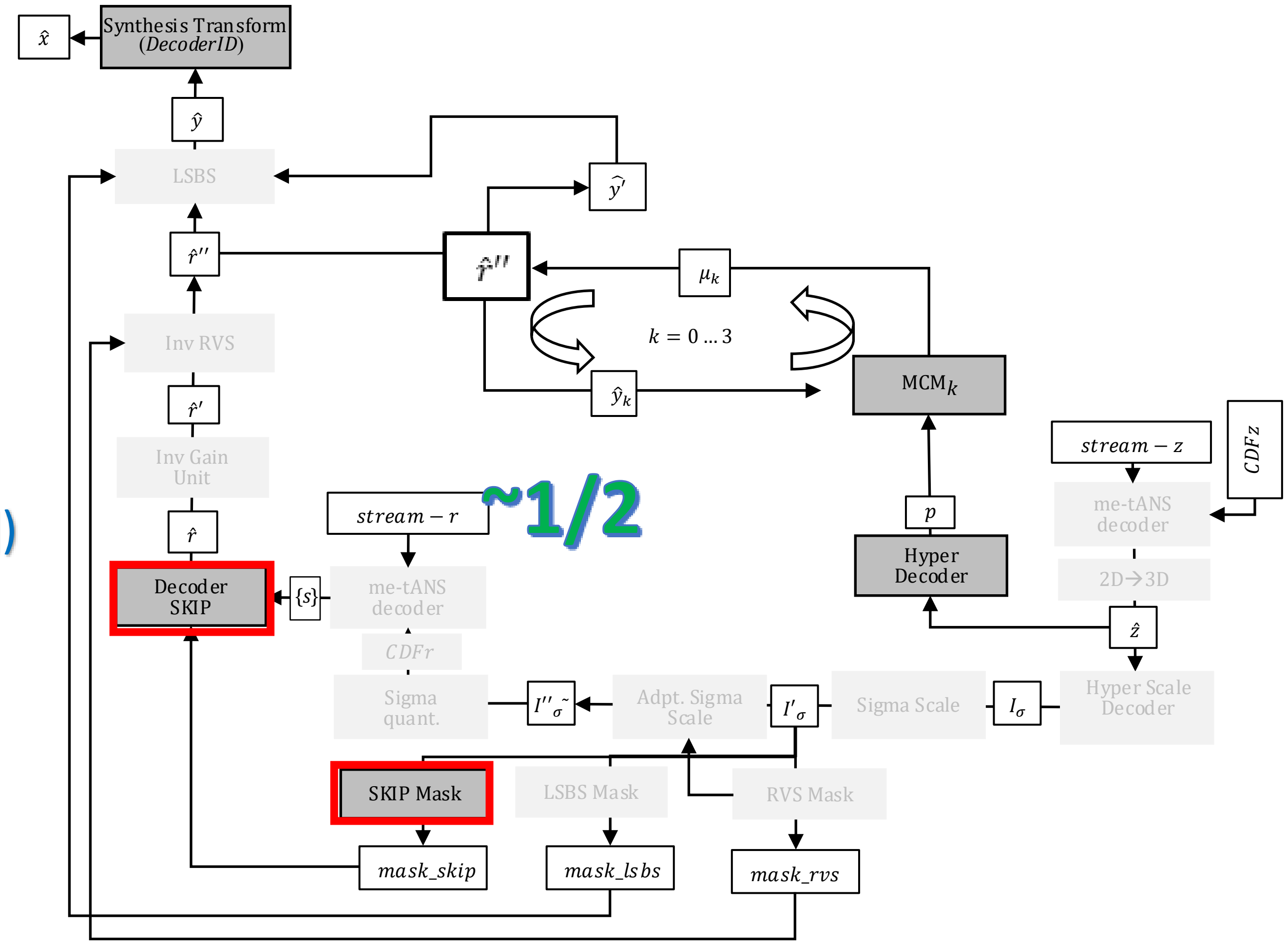
Help arithmetic coder



$$\hat{r} = ((I_\sigma < Th)\&\text{cube\_flag[i>>3,j>>3]})$$
$$?\, s : 0$$
$$Th \ =382$$

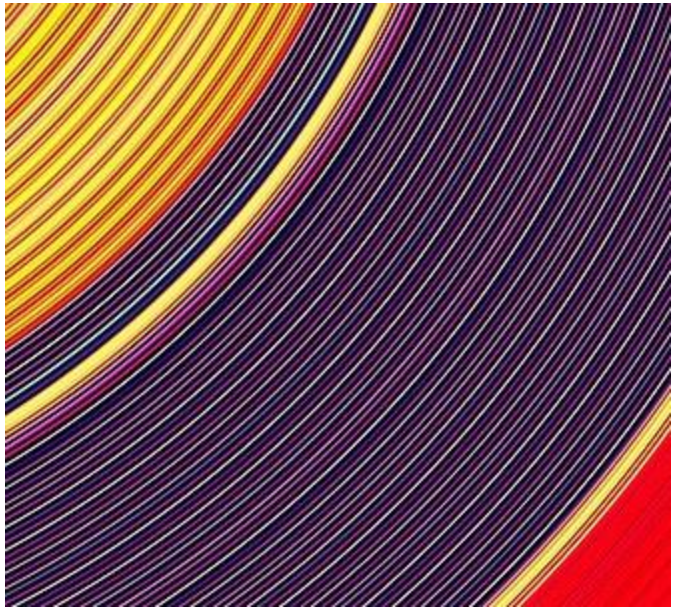
cube_flag is signaled

(the only encoder decision)

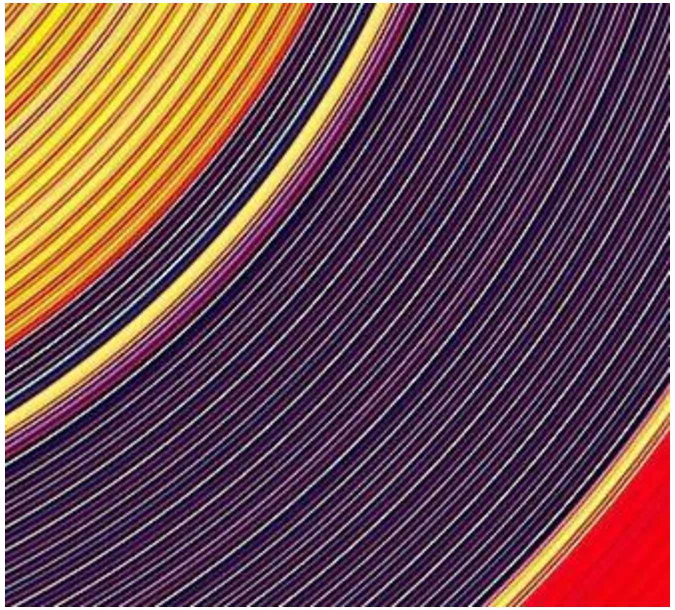


How cube_flag helps?

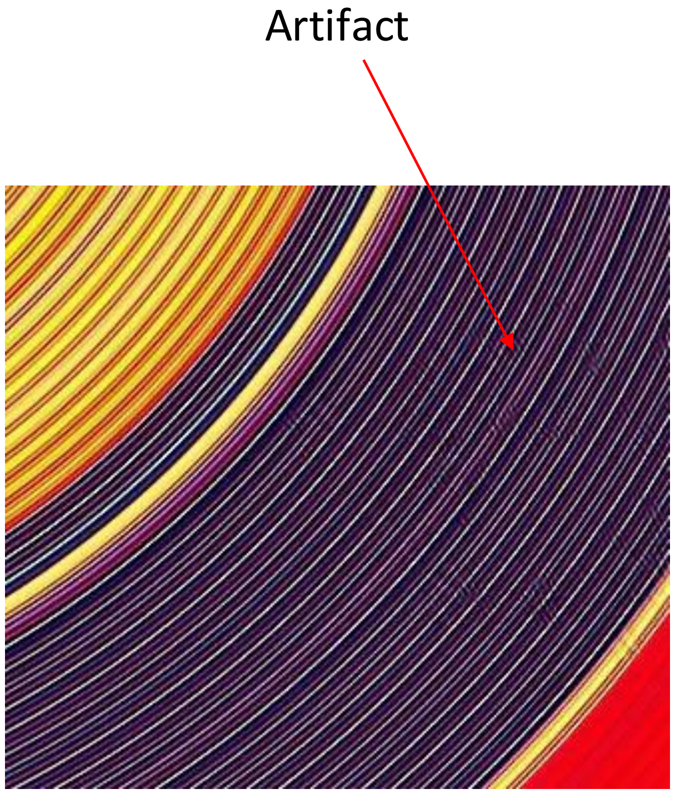
synthetic test set, 14016



original



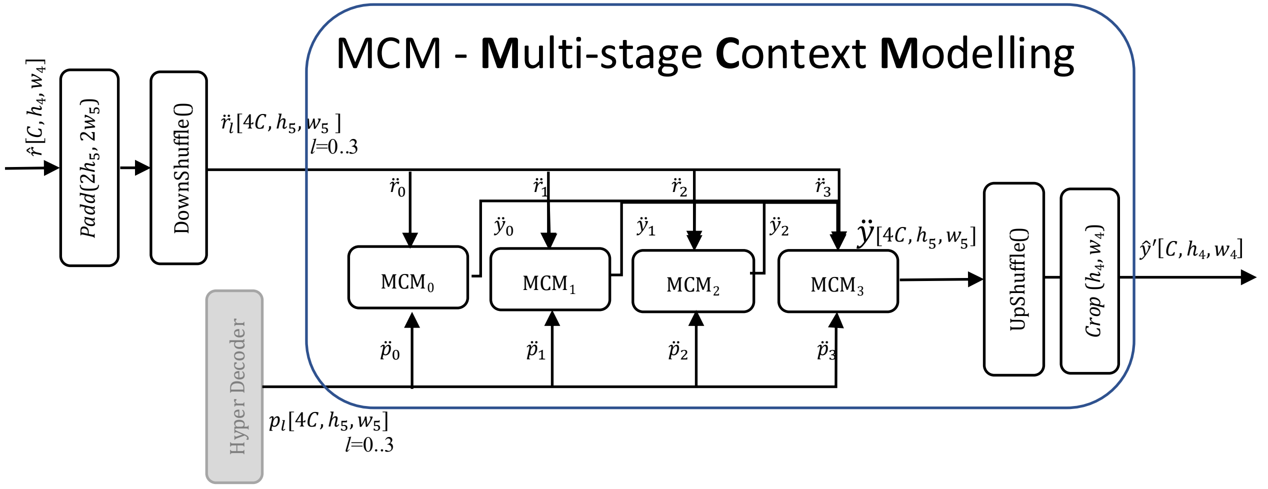
with cube_flags enable (***default***)
DecoderID=0, 2nd the highest rate



with cube_flags enable (*disabled*)
DecoderID=0, 2nd the highest rate

Latent domain prediction with context modeling

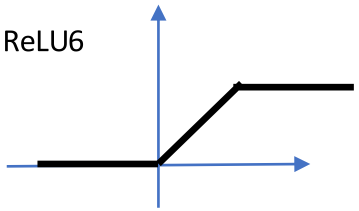
Latent domain prediction



Hyper decoder

$\hat{z}$[C, h_6, w_6]
CONV (1×1, C, C)
$CONV^{-1}$(4×4, C , C , 2↑)
Crop(h_5,w_5)
ReLU6 ()
CONV (3×3, C, C)
ReLU6 ()
CONV (3×3, C, 4·C)
p[4·C, h_5, w_5]

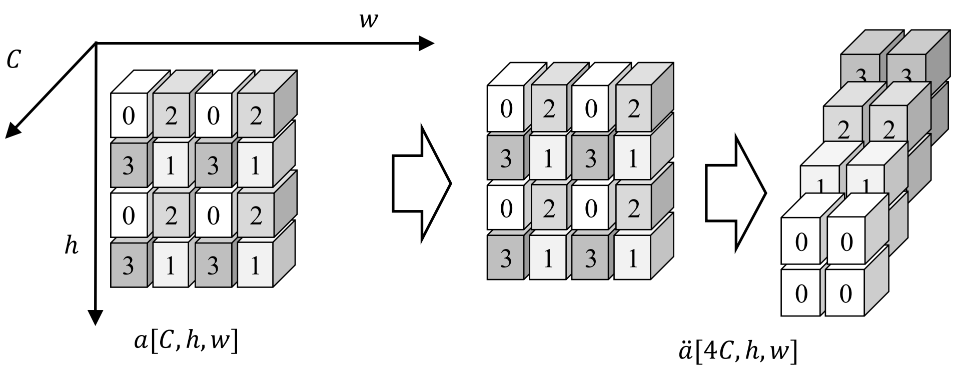
ReLU6



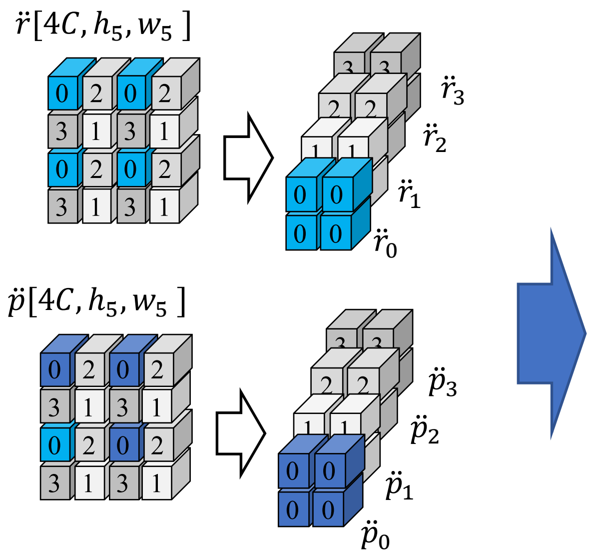
Sequential operations

~5% BD-rate gain

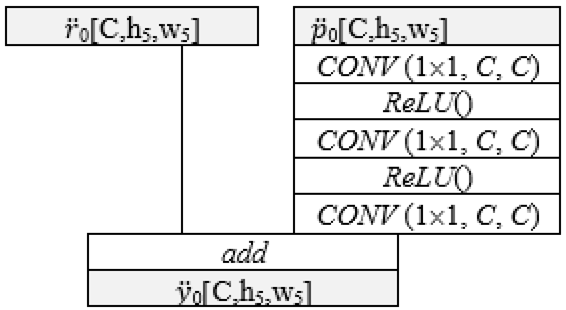
“3D chess-board” split of tensor in MCM



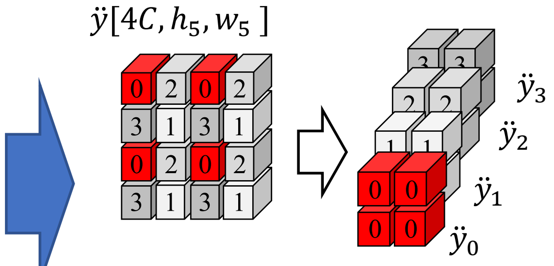
MCM process stage = 0



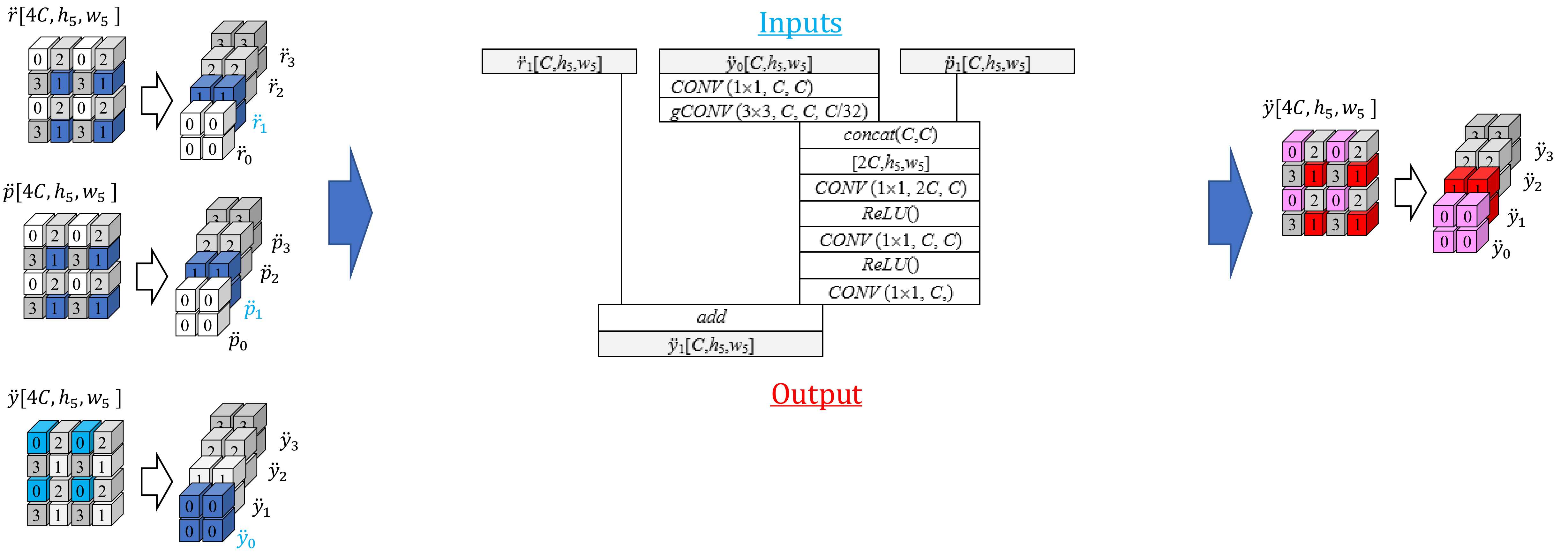
Inputs



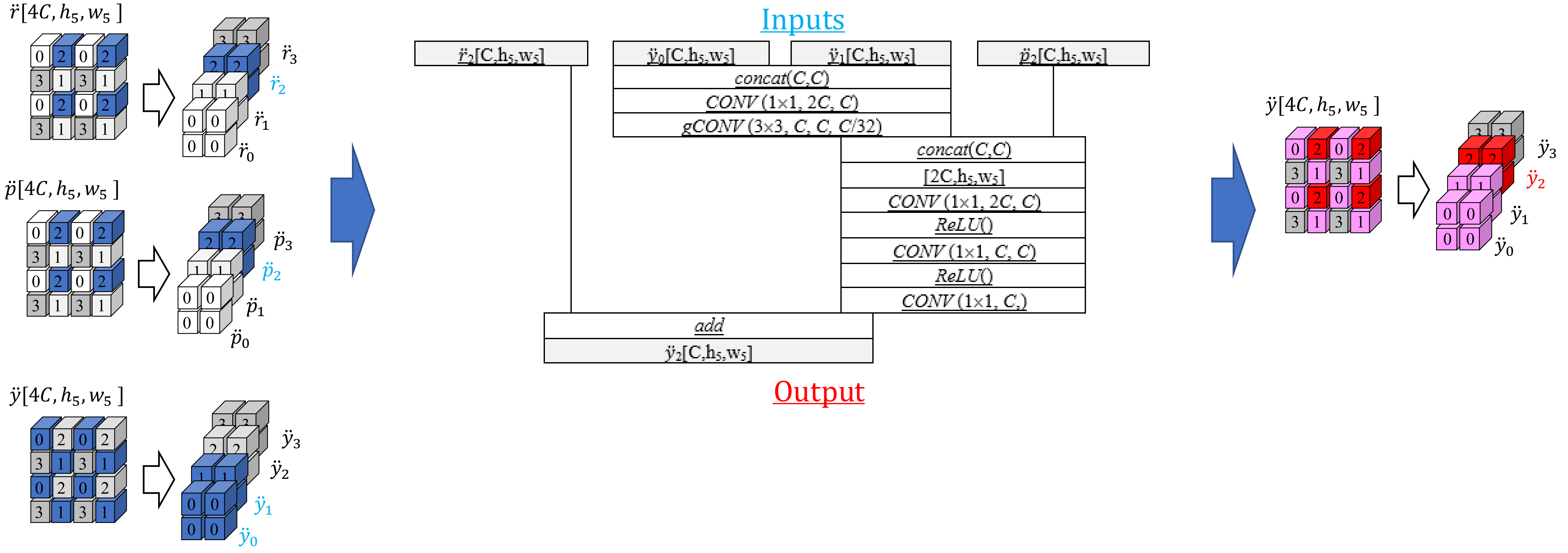
Output



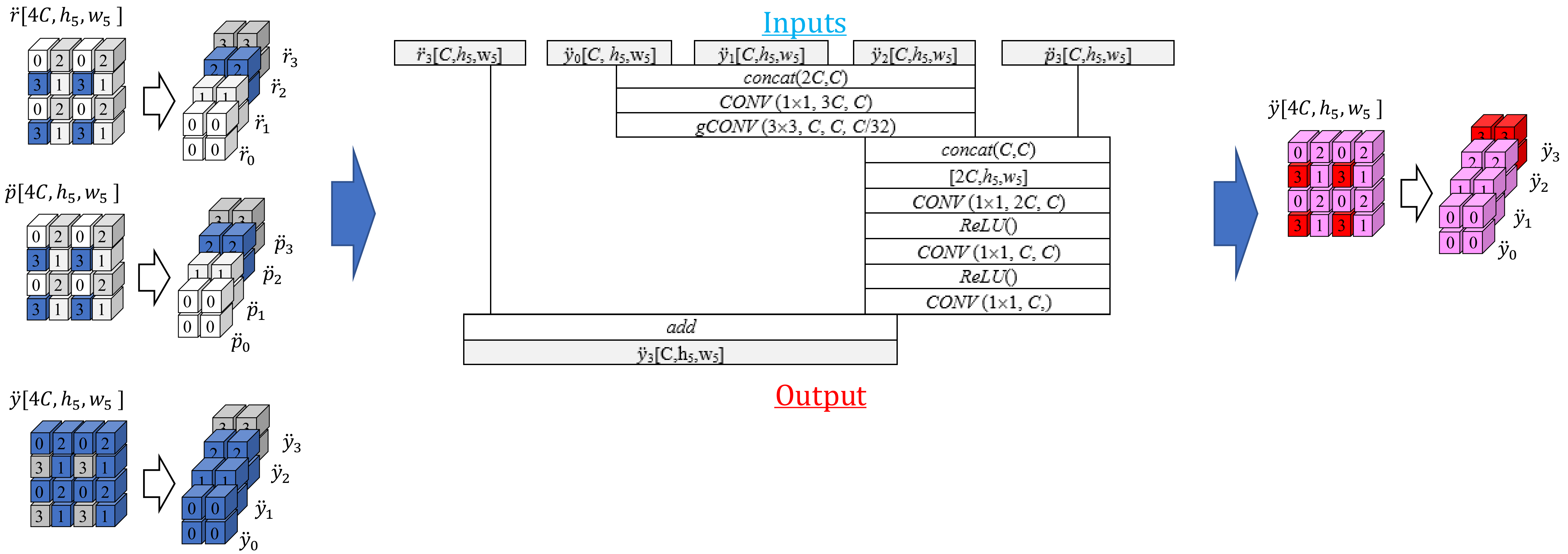
MCM process stage = 1



MCM process stage = 2



MCM process stage = 3



Analysis and synthesis transforms

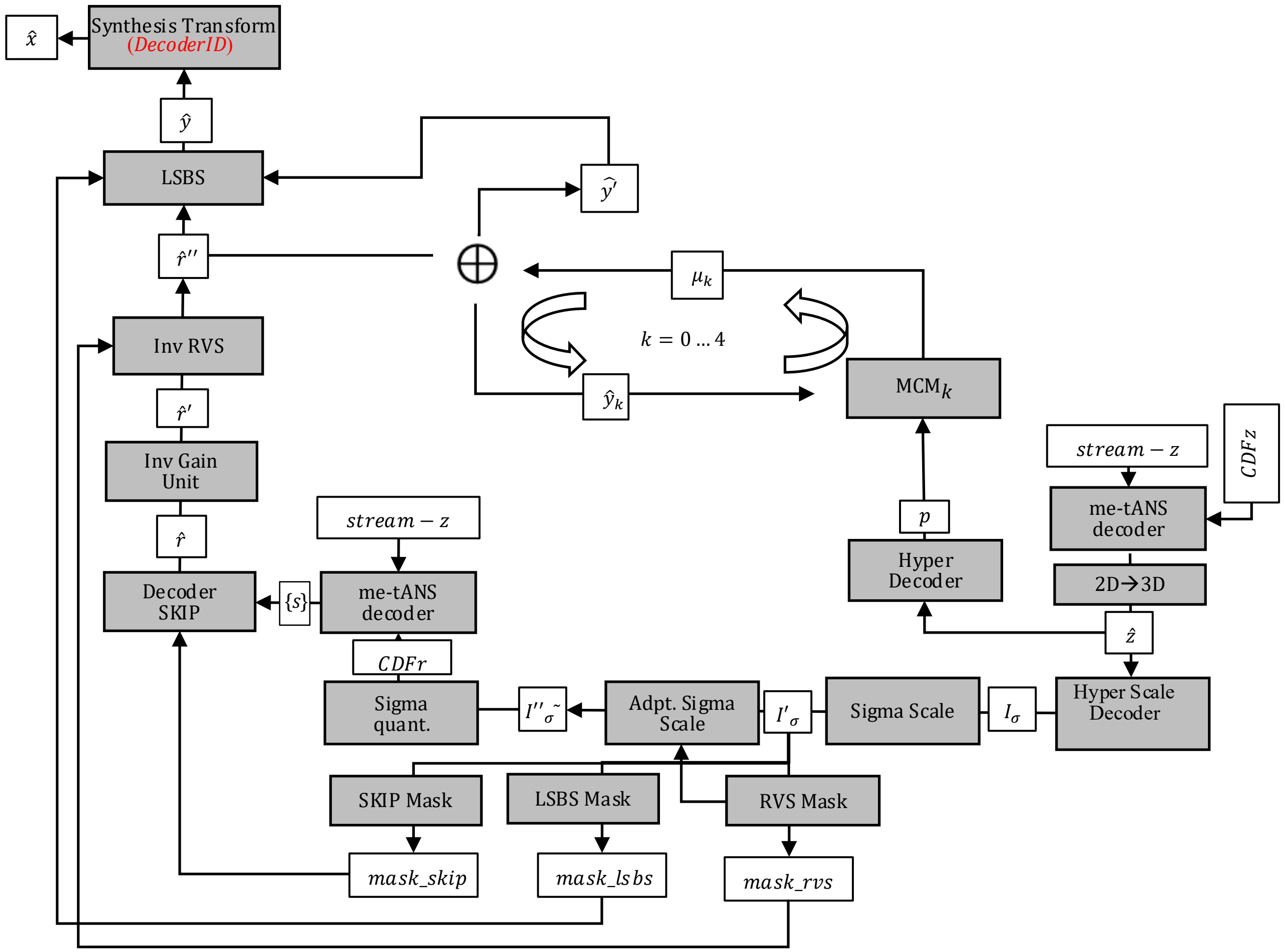
Different synthesis transforms

DecoderID = 0, 1 or 2
signaled in picture header

DecoderID = 0
CPU friendly operations

DecoderID = 1
Proven for mobile

DecoderID = 2
Attention mechanism
(transformers)



Two analysis tr. nets

$x_Y[1,H,W]$	$x_U[1, H/c_{ver}, W/c_{hor}]$	$x_V[1, H/s_{ver}, W/c_{hor}]$
Padd $(2h_1, 2w_1)$	Padd $(2h_1/c_{ver}, 2w_1/c_{hor})$	Padd $(2h_1/c_{ver}, 2w_1/c_{hor})$
UnShuffle (2,2)	Shuffle (2,2)	Shuffle (2,2)
$x_Y[4, h_1, w_1]$	$x_U[2/c_{ver}+2/c_{hor}, h_1, w_1]$	$x_V[2/c_{ver}+2/c_{hor}, h_1, w_1]$

Concat

$x_{YUV}[4+4/c_{ver}+4/c_{hor}, h_1, w_1]$

Encoder #0 (33kMAC/pxl)		Encoder#1 (163kMAC/pxl)	
Primary component	Secondary component	Primary component	Secondary component
$x_Y[1,H,W]$	$x_{YUV}[4+4/c_{ver}+4/c_{hor}, h_1, w_1]$	$x_Y[1,H,W]$	$x_{YUV}[4+4/c_{ver}+4/c_{hor}, h_1, w_1]$
Padd $(2h_1, 2w_1)$		Padd $(2h_1, 2w_1)$	
CONV (3×3, 1, 128, 2↓)		CONV (3×3, 1, 128, 2↓)	
$[128, h_1, w_1]$		$[128, h_1, w_1]$	
ResAU (128)	Padd $(2h_2, 2w_2)$	ResAU (128,4)	Padd $(2h_2, 2w_2)$
Padd $(2h_2, 2w_2)$	CONV (3×3, 12, 128, 2↓)	Padd $(2h_2, 2w_2)$	CONV (3×3, 12, 128, 2↓)
	$[128, h_2, w_2]$	TAM(0, 128)	$[128, h_2, w_2]$
CONV (3×3, 128, 128, 2↓)	ResAU (128, 4)	CONV (3×3, 128, 128, 2↓)	ResAU (128,4)
$[128, h_2, w_2]$	Padd $(2h_3, 2w_3)$	$[128, h_2, w_2]$	Padd $(2h_3, 2w_3)$
ResAU (128,4)		ResAU (128,4)	TAM(1, 128)
Padd $(2h_3, 2w_3)$	CONV (3×3, 128, 128, 2↓)	Padd $(2h_3, 2w_3)$	CONV (3×3, 128, 128, 2↓)
	$[128, h_3, w_3]$	CAB(128)	$[C_{sh}, h_3, w_3]$
CONV (3×3, 128, 128, 2↓)	ResAU (128, 4)	CONV (3×3, 128, 128, 2↓)	ResAU (128,4)
$[128, h_3, w_3]$	Padd $(2h_4, 2w_4)$	$[128, h_3, w_3]$	Padd $(2h_4, 2w_4)$
ResAU (128,4,4)		ResAU (128,4)	CAB(128)
Padd $(2h_4, 2w_4)$	CONV (3×3, 128, 128, 2↓)	Padd $(2h_4, 2w_4)$	CONV (3×3, 128, 128, 2↓)
CONV (3×3, 128, 160, 2↓)	$[128, h_4, w_4]$	CONV (3×3, 128, 160, 2↓)	$[128, h_4, w_4]$
$[160, h_4, w_4]$	ResAU (128, 4)	$[160, h_4, w_4]$	ResAU (128,4)
CONV (1×1, 160, 160)	CONV (1×1, 128, 96)	CONV (1×1, 160, 160)	CONV (1×1, 128, 96)
$y_Y[160, h_4, w_4]$	$y_{UV}[96, h_4, w_4]$	$y_Y[160, h_4, w_4]$	$y_{UV}[96, h_4, w_4]$

Shape of latent space tensors (Cp = 160, Cs = 96 and h_4, w_4) important to be compliant with decoder spec

Three synthesis transfrom nets

decoderID =0		
Primary component	Secondary component	
$\hat{y}_Y$[160, h_4,w_4]	$\hat{y}_{UV}$[96, h_4,w_4]	$\hat{y}_Y$[160, h_4,w_4]
LRB(160)		
CONV(2×2, 160, 64·2·2)		
PixelShuffle (2,2)		
Crop(h_3,w_3)		
[64, h_3,w_3]	*LCB*(*160* , *96*)	
ResAU(64)	[3·96/2 , h_4,w_4]	
CONV(2×2, 64, 32·2·2)	*CONV*(2×2, , 3·96/2 , 32·2·2)	
Shuffle (2,2)	*Shuffle* (2,2)	
Crop(h_2,w_2)	*Crop*(h_3,w_3)	
[32, h_2,w_2]	[32, h_3,w_3]	
ResAU(32, 2)	*ResAU*(32, 2)	
CONV(3×3, 32, 32)	*CONV*(3×3, 32, 32)	
ResAU(32, 2)	*ResAU*(32, 2)	
CONV(1×1, 32, 1·4·4)	*CONV*(1×1, 32, 2·8·8/c_{ver} /c_{hor})	
Shuffle (4,4)	*Shuffle* (8/c_{ver},8/c_{hor})	
Crop(*H*,*W*)	*Crop*(*H*/c_{ver}, *W*/c_{hor})	
$\hat{x}_Y$ [1, H, W]	$\hat{x}_{UV}$ [2, H/c_{ver}, W/c_{hor}]	

Shape of latent tensors
(Cp = 160, Cs = 96 and h_4, w_4)
important to be compatible
with residual coding and
prediction

decoderID =1		
Primary component	Secondary component	
$\hat{y}_Y$[160, h_4,w_4]	$\hat{y}_{UV}$[96, h_4,w_4]	$\hat{y}_Y$[160, h_4,w_4]
LRB(160)		
CONV^{-1}(4×4, 160, 64, 2↑)		
Crop(h_3,w_3)		
[64, h_3,w_3]	*LCB*(*160* , *96*)	
ResAU(64, 4)	[3·96/2 , h_4,w_4]	
CONV^{-1}(4×4, 64, 64, 2↑)	*CONV*$^{-1}$(4×4, 3·96/2 , 64, 2↑)	
Crop(h_2,w_2)	*Crop*(h_3,w_3)	
[64, h_2,w_2]	[64, h_3,w_3]	
ResAU(64, 4)	*ResAU*(64, 4)	
CONV(3×3, 64, 64)	*CONV*(3×3, 64, 64)	
ResAU(64, 4)	*ResAU*(64, 4)	
CONV(1×1, 64, 1·4·4)	*CONV*(1×1, 64, 2·8·8/c_{ver}/c_{hor})	
Shuffle (4,4)	*Shuffle* (8/c_{ver},8/c_{hor})	
Crop(*H*,*W*)	*Crop*(*H*/c_{ver}, *W*/c_{hor})	
$\hat{x}_Y$ [1, H, W]	$\hat{x}_{UV}$ [2, H/c_{ver}, W/c_{hor}]	

Shape of output tensors
(c_{ver} , c_{ver} , H and W as they are in Picture Header)

Three synthesis transfrom nets

Shape of latent tensors
(C_p = 160, C_s = 96 and h_4, w_4)
important to be compatible
with residual coding and
prediction

Sigmoid (exp)

LayerNorm (sqrt),
ELU (exp)

decoderID =2	
Primary component	Secondary component
$\hat{y}_Y$[160, h_4,w_4]	$\hat{y}_{UV}$[96, h_4,w_4] | $\hat{y}_Y$[160, h_4,w_4]
RB (160)	
CONV^{-1}(3×3, 160, 128, 2↑)	
Crop(h_3,w_3)	LCB(160 , 96)
[128, h_3,w_3]	[3·96/2 , h_4,w_4]
ResAU(128, 4)	CONV^{-1}(3×3, 3·96/2 , 64, 2↑)
CONV^{-1}(3×3, 128 , 128, 2↑)	CAB(64)
CAB(128)	Crop(h_3,w_3)
Crop(h_2,w_2)	[64 , h_3,w_3]
[128, h_2,w_2]	ResAU(64, 4)
ResAU(128, 4)	CONV(1×1, 64 , 2·2·64)
CONV(1×1, 128 , 2·2·?)	Shuffle (2,2)
Shuffle (2,2)	TAM(1, 64)
TAM(0, 128)	Crop(h_2,w_2)
Crop(h_1,w_1)	ResAU(64, 4)
ResAU(128, 4)	CONV^{-1}(3×3, 64, 2·2·2/c_{ver} /c_{hor} , 2↑)
CONV^{-1}(3×3, 128 1, 2↑)	Shuffle (2/c_{ver},2/c_{hor})
Crop(H,W)	Crop(H/c_{ver}, W/c_{hor})
$\hat{x}_Y$ [1, H, W]	$\hat{x}_{UV}$ [2, H/c_{ver}, W/c_{hor}]

Shape of output tensors
(c_{ver} , c_{ver} , H and W as they are in Picture Header)

Attention modules

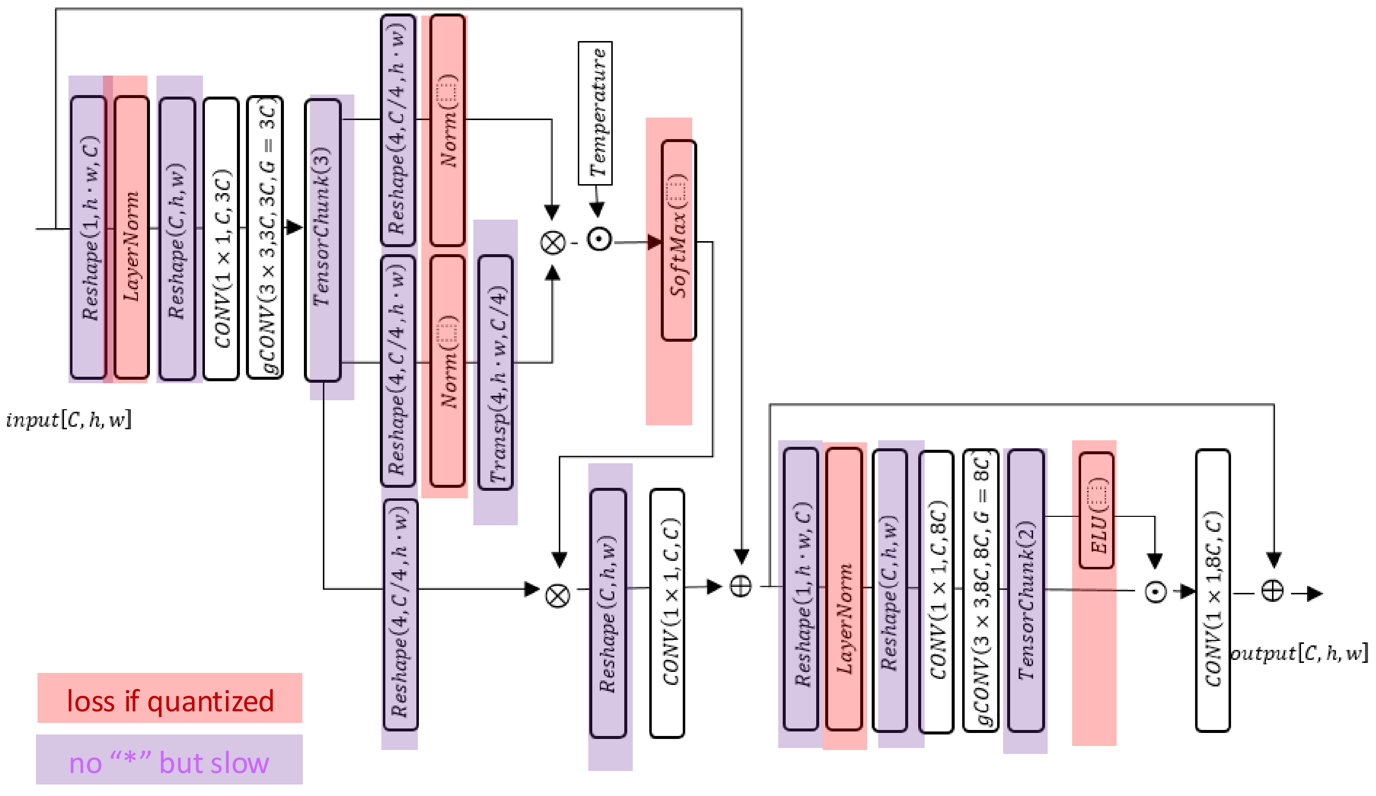


Figure 3.5.39-1 Transformer-based attention block.

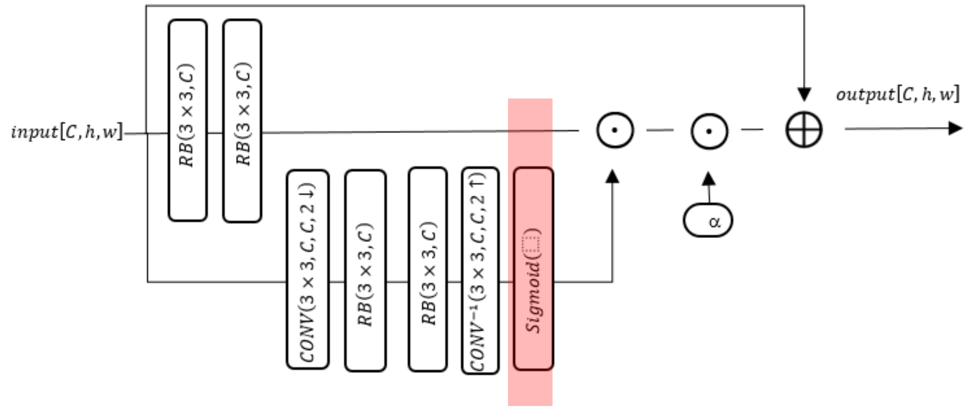


Figure 3.5.30-1 - Convolution-base attention block

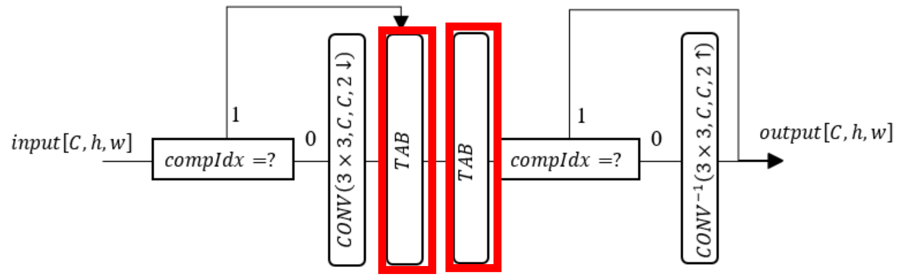


Figure 3.5.40-1 Transformer-based attention module

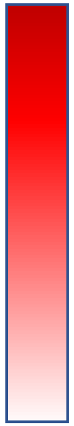
Bits allocation by JPEG AI and VVC

Original image
1336x872

High

<u>distortion</u>

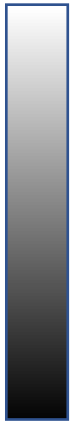
Low

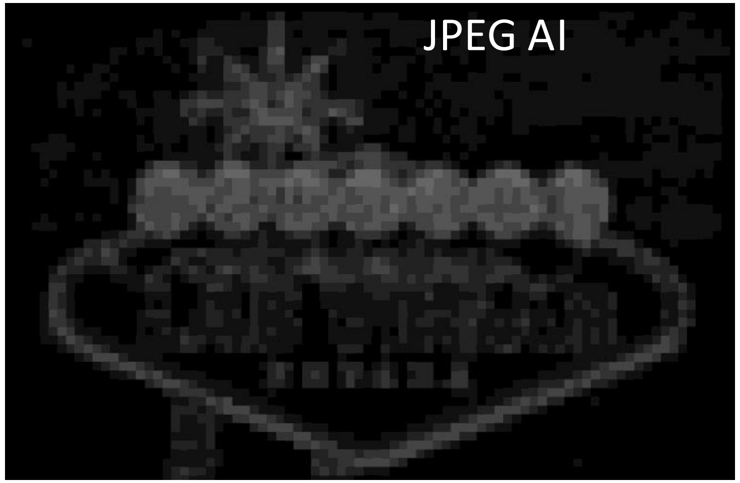


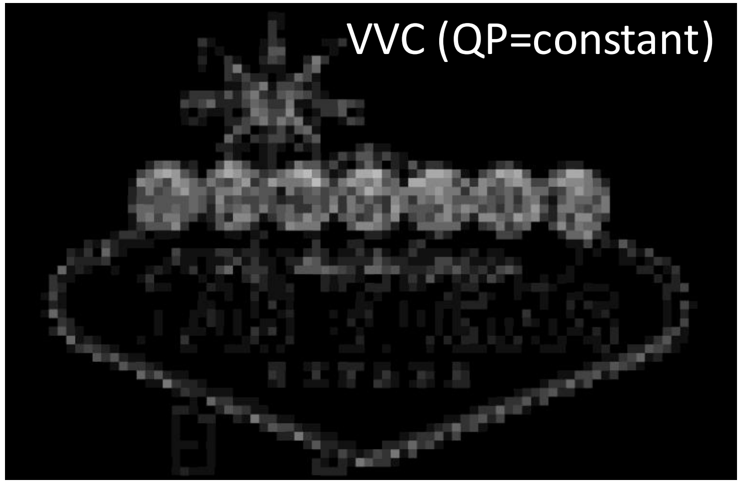
High

<u>bits</u>

Low

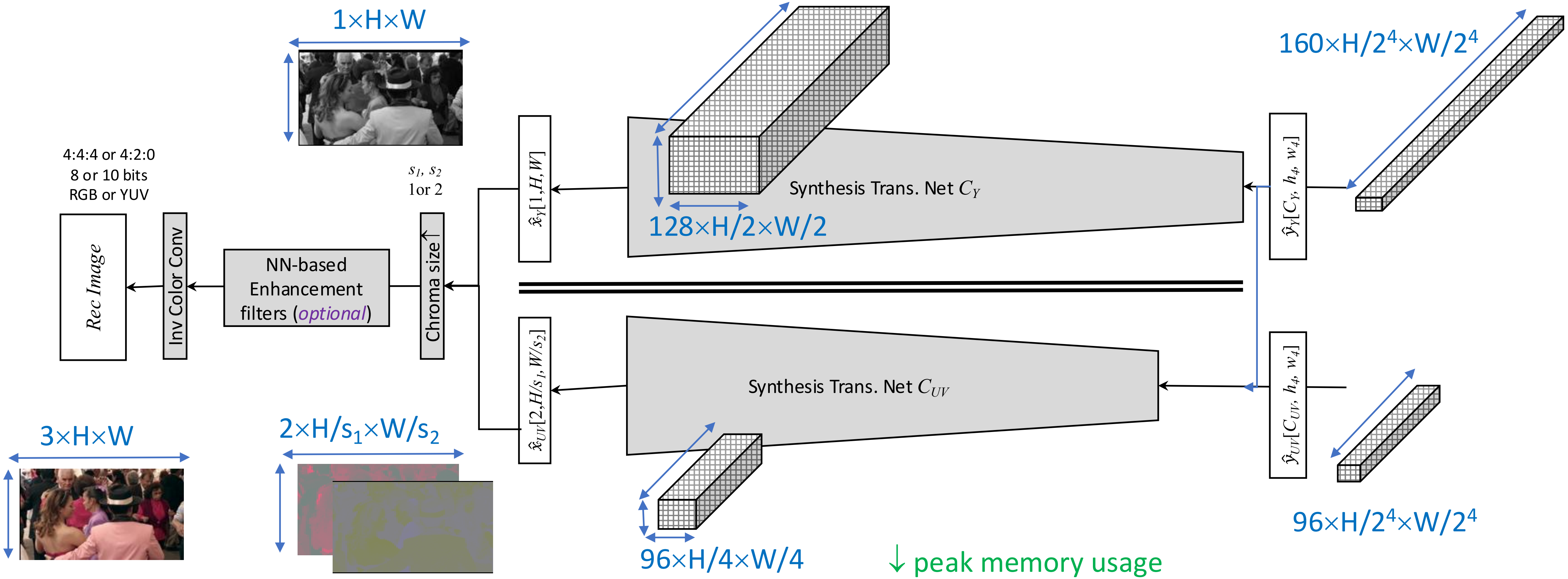






Design aspects motivated by implementation cost

Why colors are separated?

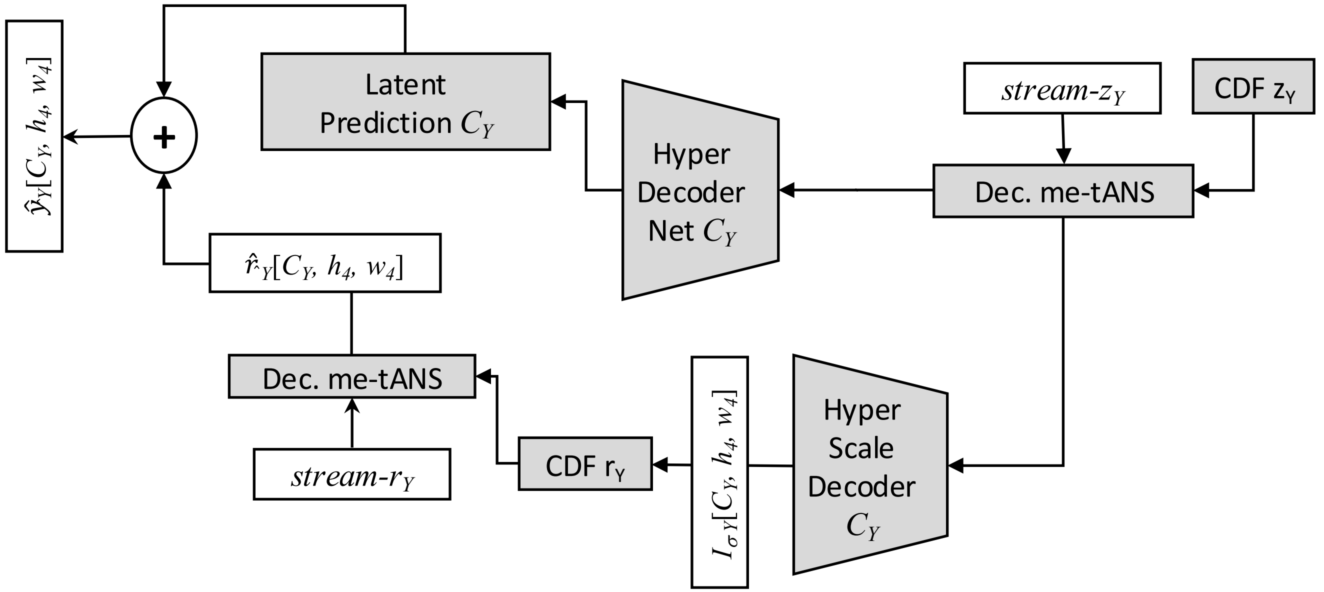


Peak memory usage for 3 colors joint coding for Autoencoder with 256 channels is 256 × W/2 × H/2 → twice higher

Why prediction / residual coding?

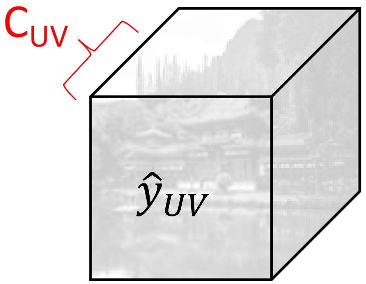
CDF modeled by Gaussian distribution
(μ, σ) - 2 parameters
CDF tables are 2D

For residual μ = 0
CDF tables are 1D

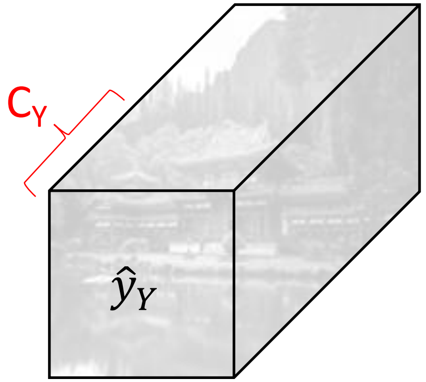


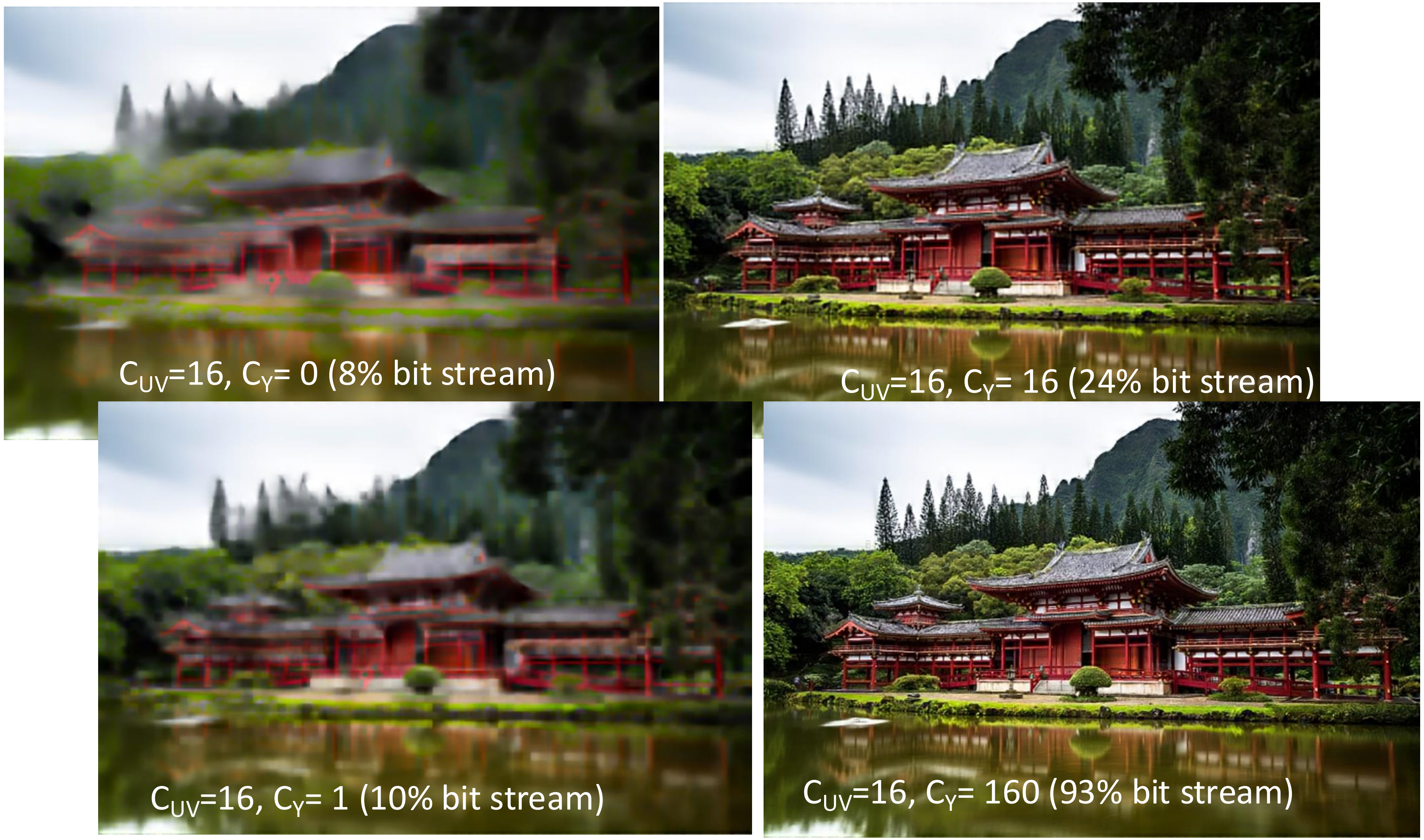
JPEG AI: progressive decoding

Chroma residual
(total C_{UV}= 96)



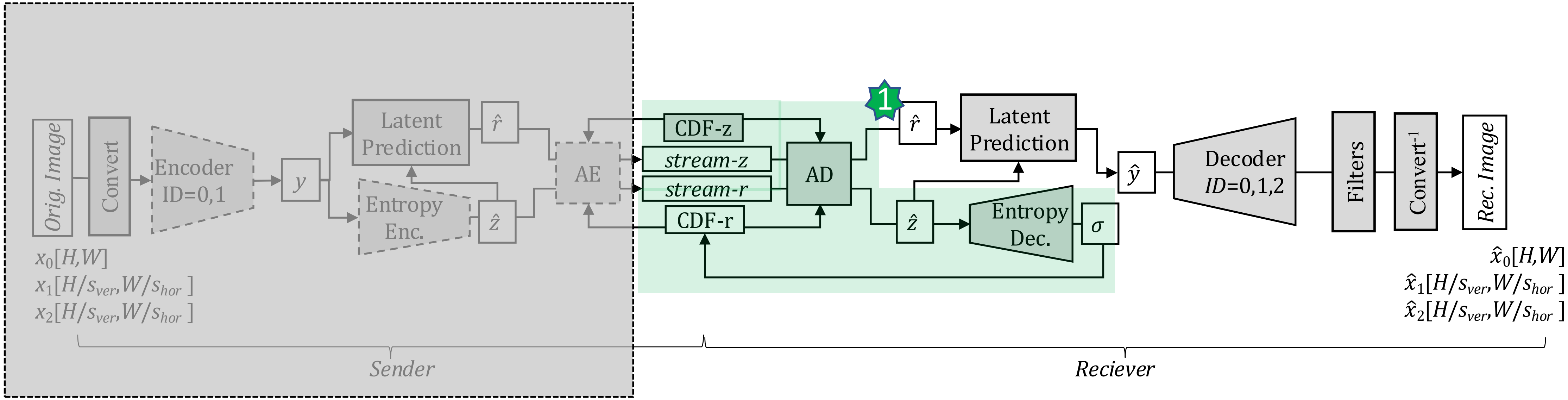
Luma residual
(total C_Y= 160)





Conformance

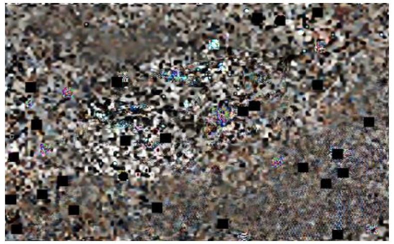
JPEG AI STRONG conformance



In case of violation:

Encoded decoded on same device

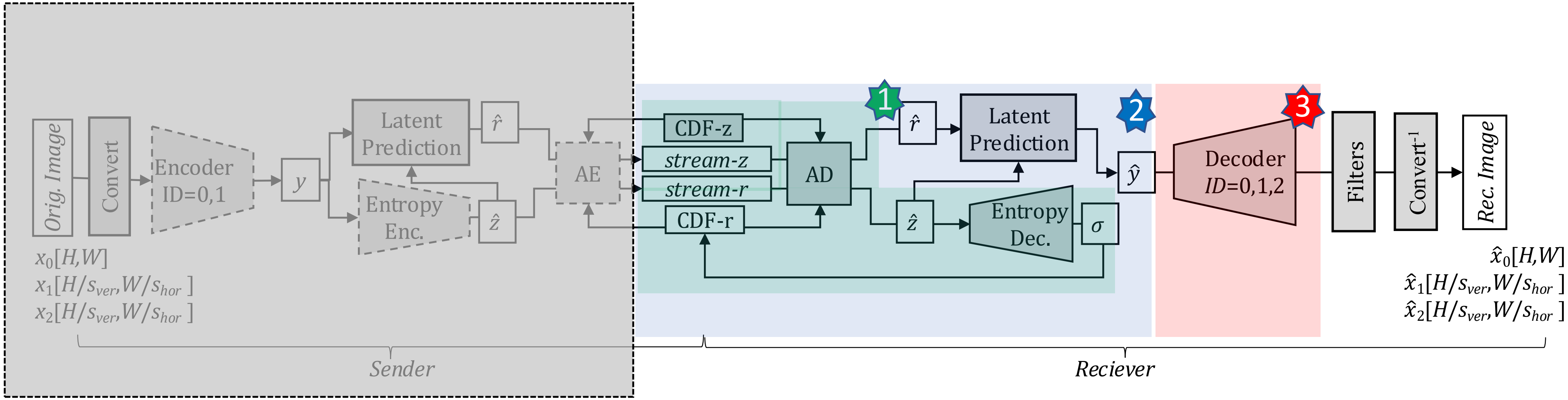
Encoded decoded on different devices



Bit.-exact behaviors required!!!

- Only integer
- Overflow aware NN quant
- Only controllable operations

JPEG AI <u>weak</u> conformance

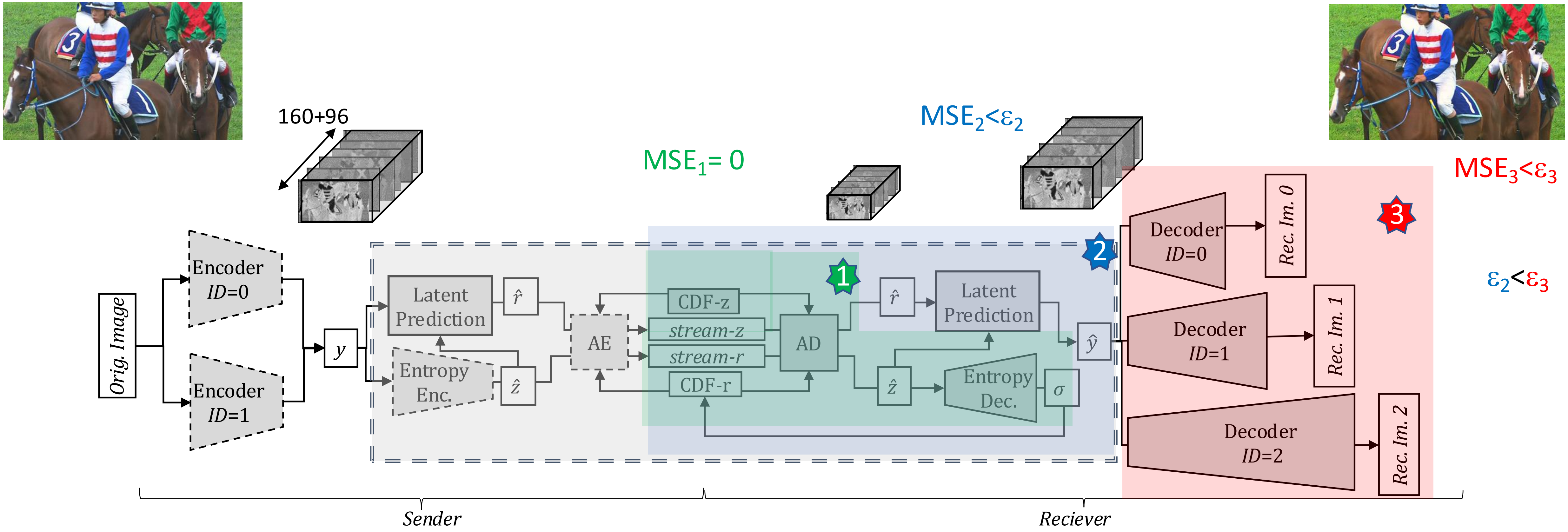


Strong: Bit.-exact behaviors required!!!
- Only integer
- Overflow aware NN quant
- Only controllable operations

‚**Weak**‘ conformance point after merging prediction and residual in order all ‚profiles‘, (including future extension) recieve the same ***tensor representaiton for image***

‚**Weaker**‘ conformance point after synthesis transfrom allows custom quantizer for NN

JPEG AI weak conformance



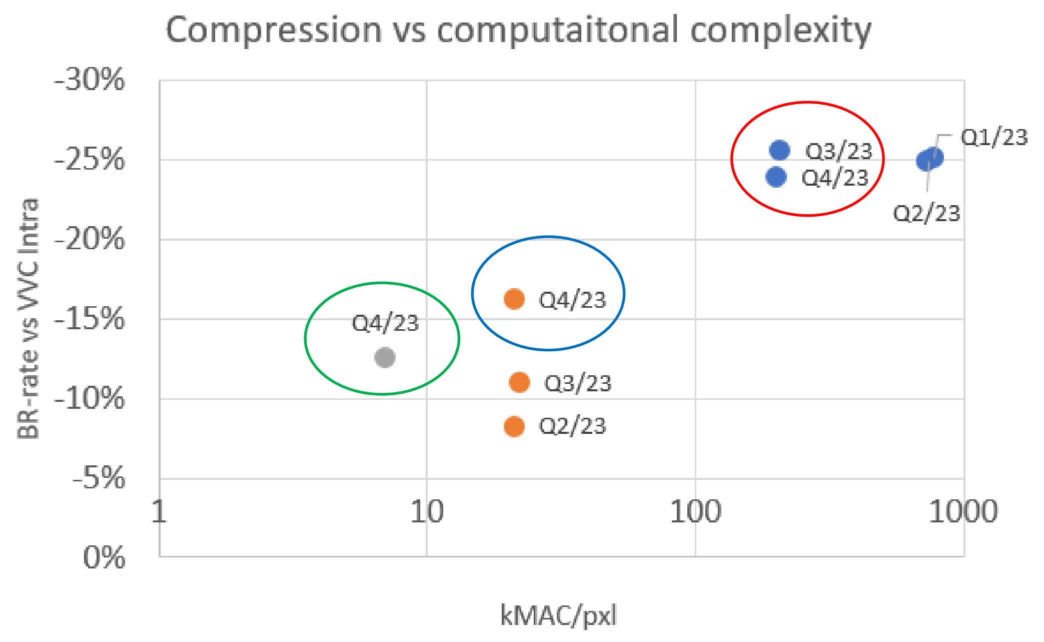
Strong conformance point: Bit.-exact!!!

‚**Weak**‘ conformance point

‚**Weaker**‘ conformance point

Demo and performance

JPEG AI evolution



JPEG AI vs VVC Intra (VTM-11 SW)

	BD-rate	kMAC /pxl	× DecT GPU	× DecT CPU	× EncT GPU
SIMPLE@main	**-12.0%**	8	0.4	1	0.0004
BASE@main	**-16.7%**	23	0.4	2	0.0005
HIGH@main	**-24.0%**	214	0.6	28	0.0010

Enc0Dec0 Target CPU decoding -> SIMPLE@main
Ecn0Dec1 Target Smartphone NPU decoding -> BASE@main
Enc1Dec2 Target GPU decoding -> HIGH@main

Performance test results

5 points BD-rate (0.12, 0.25, 0.5, 0.75, 1.0)

		BD rate vs VVC-012-025-050-075-100							Dec. complexity			Enc.
Test	AVG	MS-SSIM	VIF	FSIM	NLPD	IW-SSIM	VMAF	psnrHVS	kMAC /pxl	Time GPU, x	Time CPU, x	Time GPU, x
VTM-Intra (VVC Ref SW)	**0.0%**	0%	0%	0%	0%	0%	0%	0%	1	1	1	1
JPEG AI -Enc0Dec0	**-12.0%**	-31%	7%	-15%	-12%	-26%	-7%	1%	8	0.36	1	0.0005
JPEG AI-Enc0Dec0-tools-on	-16.2%	-31%	7%	-22%	-13%	-27%	-30%	3%	14	0.41	2	0.0011
JPEG AI-Enc0Dec1	**-16.7%**	-33%	1%	-20%	-16%	-29%	-15%	-4%	23	0.38	2	0.0005
JPEG AI-Enc0Dec1-tools-on	-20.2%	-33%	1%	-27%	-17%	-29%	-35%	-2%	28	0.41	3	0.0011
JPEG AI-Enc1Dec2-tools-off	**-24.0%**	-38%	-9%	-29%	-23%	-34%	-24%	-11%	214	0.61	28	0.0012
JPEG AI-Enc1Dec2-tools-on	-27.0%	-38%	-8%	-35%	-24%	-34%	-42%	-9%	215	0.64	29	0.0018

Reference SW

c++

Python →

Codec	8K Encoding, s
JPEG	5
HEVC / H.265	2689
VVC / H.266	18725
JPEG AI	**5 (Enc0) or 20 (Enc1)**

Tests by Alexander Karabutov – JPEG AI project SW coordinator

also an author of JPEG AI Smartphone encoder & decoder prototype

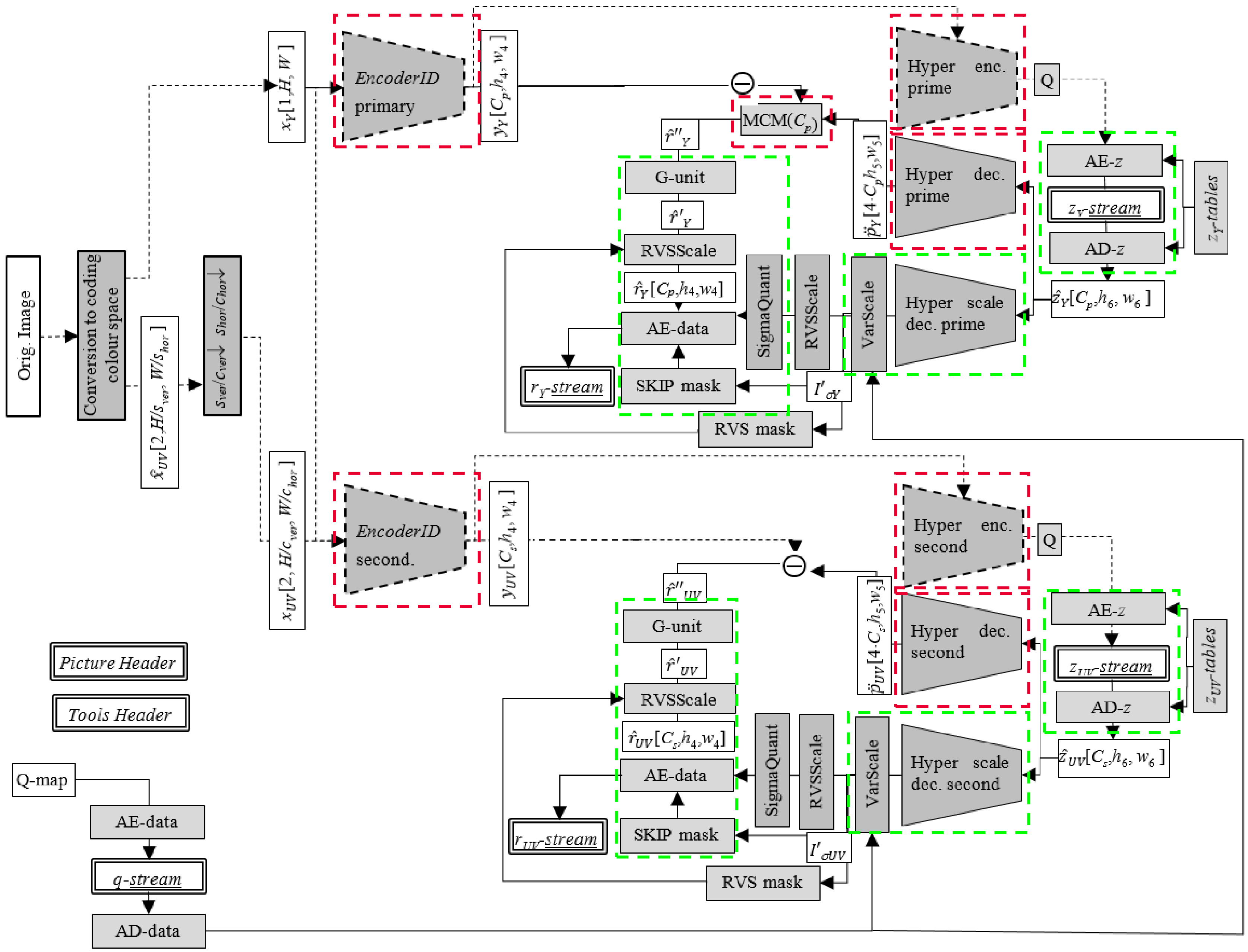
World's first smartphone implementation of JPEG AI's decoder

<u>Main targets:</u>

1. Demonstrate to the world that JPEG AI can fly on smartphone right now even without dedicated chip
2. Identify JPEG AI design issues preventing deployment on mobile platform as early as possible
3. Verify device interoperability of JPEG AI standard

- <u>Device</u>: Huawei Mate50 Pro with Qualcomm Snapdragon 8+ Gen1
- <u>Technology stack</u>: C++, Java, SNPE AI acceleration framework, Bolt CPU inference framework
- <u>Features</u>: JPEG AI base profile decoding, high resolution support (4K), tiling

Encoder structure



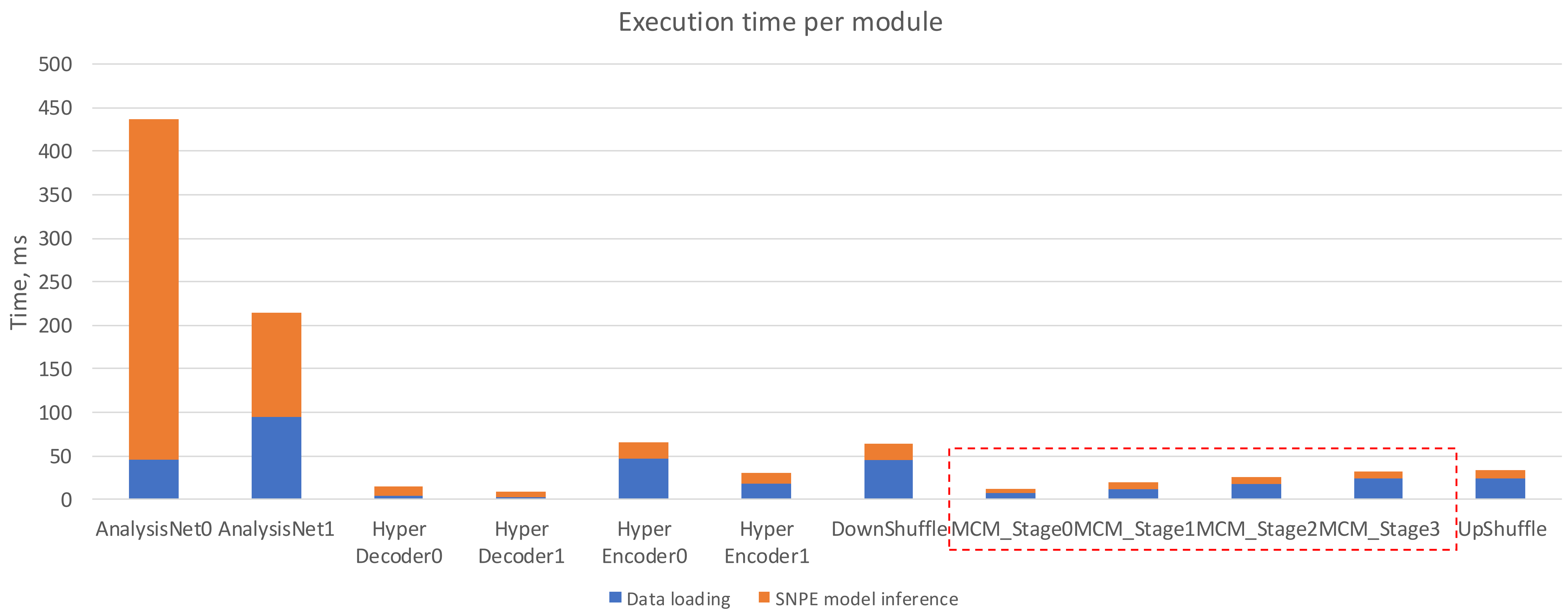
Operation	Duration, ms
Colour convertion	40
Chroma subsampling	42
Luma pipeline	**804**
Chroma pipeline	**313**
Entropy parts:	**75**
HSDs	55
Gain units (siagma scale)	2
Skip masks generation	2
Arithmetic (only Luma)	10
Arithmetic (only Chroma)	6

Executed on NPU with loading data from CPU

Executed on CPU

*Time was measured on Mate50

Analysis of execution time of modules on NPU



Each of the iterations executes on NPU, but data between them are stored on CPU because of the rounding operation

The total time of data loading is **340 ms**, the models' execution time is **614 ms**.

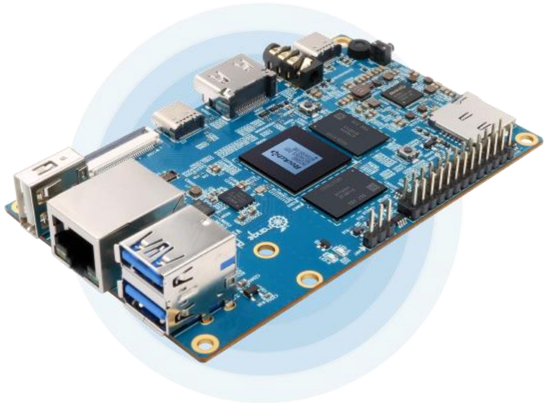
*Time was measured on Mate50

Results on Orange Pi 5

CPU: 8-core 64 core architecture, 4*Cortex-A76 + 4*Cortex-A55

GPU: ARM Mali-G610

NPU: 6Tops AI computing power



Streams are encoded on Nvidia GPU and decoded on Orange Pi 5

		5 points BD-rate (0.12, 0.25, 0.5, 0.75, 1.0)								10%				
		BD rate vs VVC-012-025-050-075-100												
Test	AVG	msssim Torch	vif	fsim	nlpd	iw-ssim	vmaf	psnrHVS	Monotonicity	Max Bit Dev.	MAX kMAC/pxl	AVG kMAC/pxl	Time GPU, x	Time CPU, x
RCv5.0s4-bopEnc-sopDec-tools-off-CPU	-12.0%	-31.4%	2.9%	-14.8%	-12.6%	-26.8%	-2.1%	1.0%	TRUE	207%	7	7	0.1	1.6
BOP-SOP-tools-off	-12.0%	-31.4%	2.9%	-14.8%	-12.6%	-26.8%	-2.1%	1.0%	TRUE	207%	7	7	#VALUE!	3.2
RCv5.0s4-bopEnc-sopDec-tools-on-CPU	-17.2%	-31.5%	4.1%	-24.5%	-15.1%	-27.6%	-26.0%	0.3%	TRUE	230%	17	12	0.2	9.2
BOP-SOP-tools-on	-17.3%	-31.6%	3.8%	-24.6%	-15.3%	-27.7%	-26.2%	0.1%	TRUE	230%	17	12	#VALUE!	7.7
RCv5.0s4-BOP-tools-off-CPU	-15.9%	-33.3%	-2.0%	-19.6%	-15.7%	-29.1%	-8.2%	-3.3%	TRUE	207%	21	21	0.15	3.1
BOP-tools-off	-15.9%	-33.3%	-2.0%	-19.6%	-15.7%	-29.1%	-8.2%	-3.3%	TRUE	207%	21	21	#VALUE!	5.7
RCv5.0s4-BOP-tools-on-CPU	-20.7%	-33.3%	-0.8%	-28.6%	-18.1%	-29.7%	-30.8%	-3.7%	TRUE	230%	31	24	0.20	10.5
BOP-tools-on	-20.9%	-33.4%	-1.0%	-28.7%	-18.2%	-29.8%	-30.9%	-3.9%	TRUE	230%	31	24	#VALUE!	9.2
RCv5.0s4-HOP-tools-off-CPU	-23.5%	-37.8%	-11.6%	-29.7%	-22.3%	-34.0%	-18.4%	-10.7%	TRUE	210%	206	200	0.35	43.6
HOP-tools-off	-23.5%	-37.8%	-11.6%	-29.7%	-22.3%	-34.0%	-18.4%	-10.7%	TRUE	210%	206	200	#VALUE!	61.8
RCv5.0s4-HOP-tools-on-CPU	-27.9%	-37.4%	-10.6%	-37.6%	-24.3%	-34.3%	-40.2%	-10.5%	TRUE	233%	215	201	0.46	48.2
HOP-tools-on	-27.7%	-37.5%	-10.5%	-37.4%	-24.4%	-34.3%	-38.9%	-11.0%	TRUE	233%	215	201	#VALUE!	65.6

Storage scenario (visually lossless)

HEIF compressed image. Total size of the image is 661 KB

$\times 2 \ldots 3 \downarrow$

JPEG-AI image. Total size of the image is 272 KB.

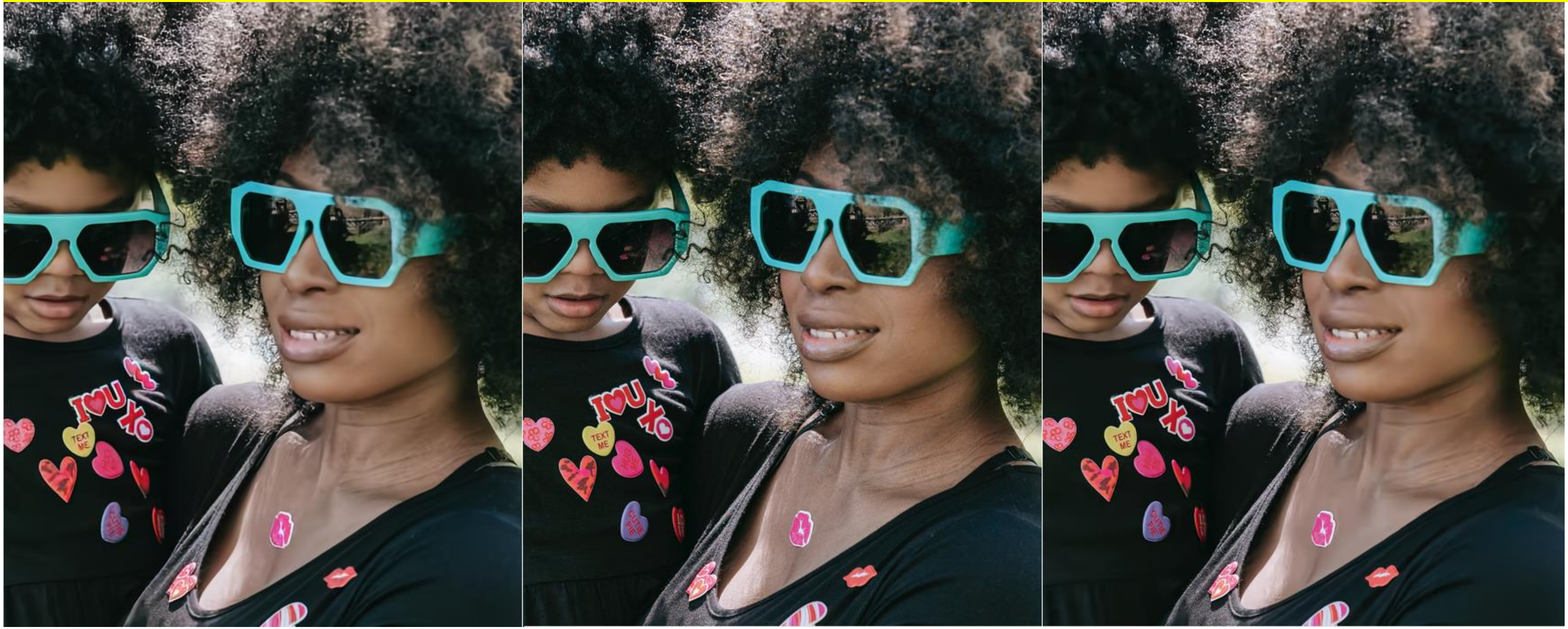
Image sharing though messengers

WeChat transferred image with scaling and reencoding. Total size of the image is 213 KB

JPEG-AI image without rescaling. Total size is 272 KB.

Visual quality analysis

JPEG AI main@high 0.25bpp | ORIG | VTM Intra (VVC Ref SW) 0.5 bpp



Visual quality analysis

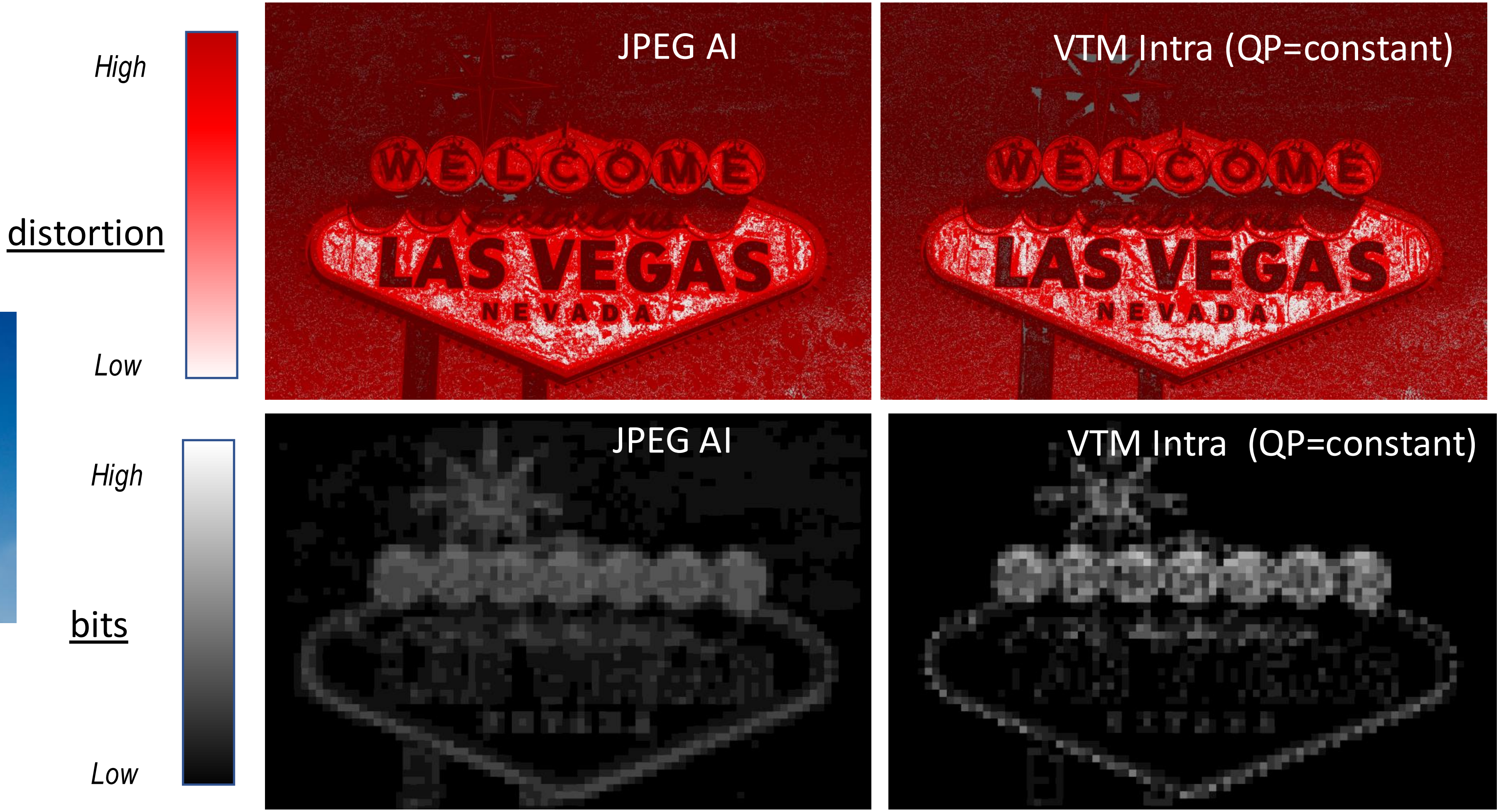
JPEG AI main@high 0.5bpp | ORIG | VTM Intra (VVC Ref SW) 0.5 bpp


0.25 bpp
MS-SSIM = 0.9766
PSNR-Y = 33.6
VMAF = 81.6


0.26 bpp
MS-SSIM = 0.9890
PSNR-Y = 34.2
VMAF = 86.3

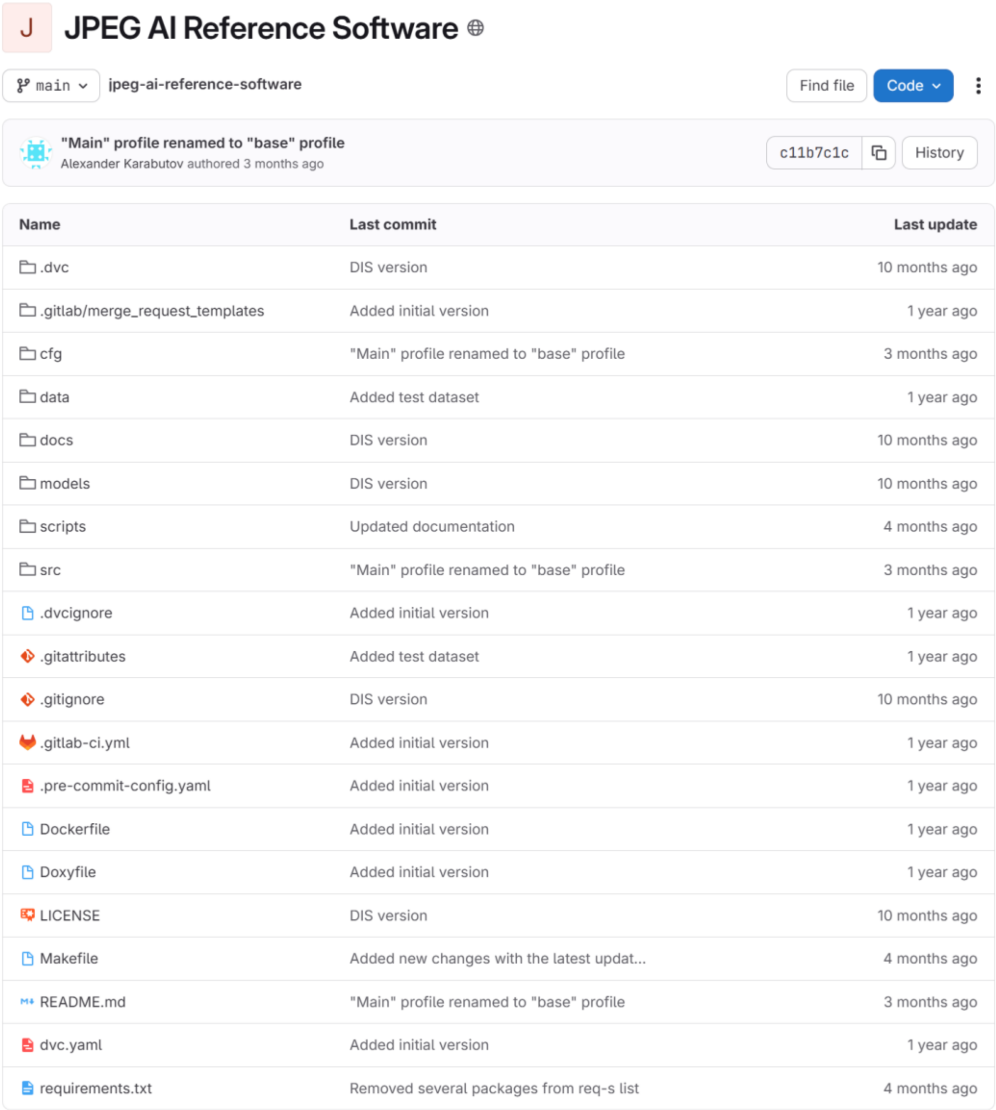
Bits allocation by JPEG AI and VTM (VVC)

Original image
1336x872



JPEG AI verification model and reference software

JPEG AI Reference SW



J JPEG AI Reference Software

main jpeg-ai-reference-software Find file Code

"Main" profile renamed to "base" profile
Alexander Karabutov authored 3 months ago c11b7c1c History

Name	Last commit	Last update
.dvc	DIS version	10 months ago
.gitlab/merge_request_templates	Added initial version	1 year ago
cfg	"Main" profile renamed to "base" profile	3 months ago
data	Added test dataset	1 year ago
docs	DIS version	10 months ago
models	DIS version	10 months ago
scripts	Updated documentation	4 months ago
src	"Main" profile renamed to "base" profile	3 months ago
.dvcignore	Added initial version	1 year ago
.gitattributes	Added test dataset	1 year ago
.gitignore	DIS version	10 months ago
.gitlab-ci.yml	Added initial version	1 year ago
.pre-commit-config.yaml	Added initial version	1 year ago
Dockerfile	Added initial version	1 year ago
Doxyfile	Added initial version	1 year ago
LICENSE	DIS version	10 months ago
Makefile	Added new changes with the latest updat...	4 months ago
README.md	"Main" profile renamed to "base" profile	3 months ago
dvc.yaml	Added initial version	1 year ago
requirements.txt	Removed several packages from req-s list	4 months ago

Link: https://gitlab.com/wg1/jpeg-ai/jpeg-ai-reference-software

SW package contains:

1) Test dataset
2) Encoder/decoder modules
3) SW for evaluation of a dataset
4) Training code
5) Pretrained models and the best checkpoints from the trainings
6) Docker file for creating an image

JPEG AI Dataset

JPEG AI Reference Software

main · jpeg-ai-reference-software · Find file · Code

"Main" profile renamed to "base" profile
Alexander Karabutov authored 3 months ago · c11b7c1c · History

Name	Last commit	Last update
.dvc	DIS version	10 months ago
.gitlab/merge_request_templates	Added initial version	1 year ago
cfg	"Main" profile renamed to "base" profile	3 months ago
data	Added test dataset	1 year ago
docs	DIS version	10 months ago
models	DIS version	10 months ago
scripts	Updated documentation	4 months ago
src	"Main" profile renamed to "base" profile	3 months ago
.dvcignore	Added initial version	1 year ago
.gitattributes	Added test dataset	1 year ago
.gitignore	DIS version	10 months ago
.gitlab-ci.yml	Added initial version	1 year ago
.pre-commit-config.yaml	Added initial version	1 year ago
Dockerfile	Added initial version	1 year ago
Doxyfile	Added initial version	1 year ago
LICENSE	DIS version	10 months ago
Makefile	Added new changes with the latest updat...	4 months ago
README.md	"Main" profile renamed to "base" profile	3 months ago
dvc.yaml	Added initial version	1 year ago
requirements.txt	Removed several packages from req-s list	4 months ago

data

Find file · Code

Added test dataset
Alexander Karabutov authored Jul 30, 2024 · 348e4d86 · History

Name	Last commit	Last update
..		
calibration_set	Added initial version	1 year ago
test	Added test dataset	1 year ago
test_10bit	Added initial version	1 year ago

test

Find file · Code

Added test dataset
Alexander Karabutov authored Jul 30, 2024 · 348e4d86 · History

Name	Last commit	Last update
..		
00001_TE_2096x1400_8bit_sRGB.png LFS	Added test dataset	1 year ago
00001_TE_2096x1400_8bit_sRGB.png.dvc	Added initial version	1 year ago
00002_TE_2144x1424_8bit_sRGB.png LFS	Added test dataset	1 year ago
00002_TE_2144x1424_8bit_sRGB.png.dvc	Added initial version	1 year ago
00003_TE_1944x1296_8bit_sRGB.png LFS	Added test dataset	1 year ago
00003_TE_1944x1296_8bit_sRGB.png.dvc	Added initial version	1 year ago
00004_TE_1808x1352_8bit_sRGB.png LFS	Added test dataset	1 year ago
00004_TE_1808x1352_8bit_sRGB.png.dvc	Added initial version	1 year ago

JPEG AI sources structure

J **JPEG AI Reference Software**

main · jpeg-ai-reference-software · Find file · Code

"Main" profile renamed to "base" profile
Alexander Karabutov authored 3 months ago · c11b7c1c · History

Name	Last commit	Last update
.dvc	DIS version	10 months ago
.gitlab/merge_request_templates	Added initial version	1 year ago
cfg	"Main" profile renamed to "base" profile	3 months ago
data	Added test dataset	1 year ago
docs	DIS version	10 months ago
models	DIS version	10 months ago
scripts	Updated documentation	4 months ago
src	"Main" profile renamed to "base" profile	3 months ago
.dvcignore	Added initial version	1 year ago
.gitattributes	Added test dataset	1 year ago
.gitignore	DIS version	10 months ago
.gitlab-ci.yml	Added initial version	1 year ago
.pre-commit-config.yaml	Added initial version	1 year ago
Dockerfile	Added initial version	1 year ago
Doxyfile	Added initial version	1 year ago
LICENSE	DIS version	10 months ago
Makefile	Added new changes with the latest updat...	4 months ago
README.md	"Main" profile renamed to "base" profile	3 months ago
dvc.yaml	Added initial version	1 year ago
requirements.txt	Removed several packages from req-s list	4 months ago

src · Find file · Code

"Main" profile renamed to "base" profile
Alexander Karabutov authored 3 months ago · c11b7c1c · History

Name	Last commit	Last update
..		
codec	Added new changes with the latest updates	4 months ago
dump	Added new changes with the latest updates	4 months ago
models_export/scripts	DIS version	10 months ago
quant/scripts	DIS version	10 months ago
reco	"Main" profile renamed to "base" profile	3 months ago
train	"Main" profile renamed to "base" profile	3 months ago

“src” is the directory with source code
codec is a directory with general mode

Set-up an environment

1. Conda environment
   a) Install Conda (https://www.anaconda.com/) or Miniconda (https://docs.anaconda.com/miniconda/)
   b) Run “make configure” command.

   It will create “jpeg-ai-vm” conda environment and install all necessary packages.

2. Docker container:
   a) Download and run:
      Run “make run_docker” command.

   a) Build an image:
      Run “make build_docker && make run_docker” command.

   In both cases the docker image will include the same conda environment as the item 1 provides.

Run “make build_test_libs” command afterwards.

The encoder and the decoder usage

1. **Common step**: activate the conda environment by the command:

```
conda activate jpeg_ai_vm
```

2. The command line to run the encoder:

```
python -m src.reco.coders.encoder input.png output.jai
```

2. The command line to run the decoder:

```
python -m src.reco.coders.decoder input.jai output.png
```

The evaluation script usage

1. **Common step**: activate the conda environment by the command:

```
conda activate jpeg_ai_vm
```

2. The command line to run the evaluation script:

`make test` or `python -m src.reco.scripts.eval [--in_dir <INDIR>] [--out_dir <OUTDIR>]`

where `<INDIR>` is an input directory with the dataset ("data/test" is by default), `<OUTDIR>` in an output directory ("results/test" is by default)

Training process

Training stages:

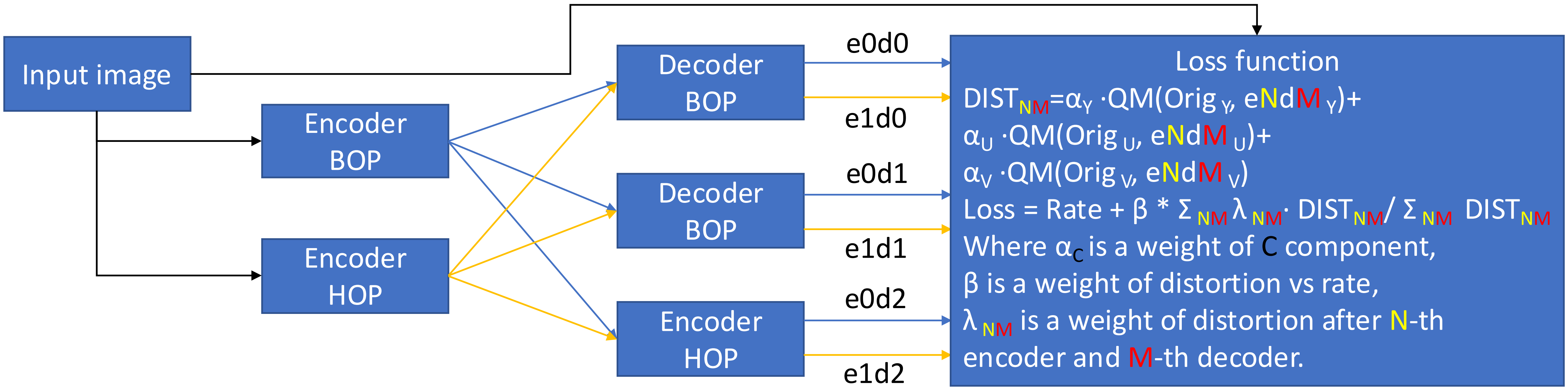
MSE training
Loss function: MSE
Epochs: 64
Variable rate: off

Mixed training
Loss function: 0.5* (MSE + MSSSIM)
Epochs: 36
Variable rate: off

Decoder finetune
Loss function: 0.5* (MSE + MSSSIM)
Epochs: 20
Variable rate: off

Variable rate finetuning
Loss function: 0.5* (MSE + MSSSIM)
Epochs: 20
Variable rate: on

The loss function



Training

1. Download training and validation datasets:

```
make download_train_ds
```

2. Run training of all models on 8 GPUs from the scratch:

```
make train
```

or

```
python -m scripts.acc_train_scripts.acc_train_local \
       --data_dir <TRAIN_DS_PATH> \
       --lst <TRAIN_LST_PATH> \
       --val_data_dir <VAL_DS_PATH> \
       --val_lst <VAL_LST_PATH> \
       --train_url <OUTPUT_PATH>
```

where <TRAIN_DS_PATH> is a path to the directory with the training dataset, <TRAIN_LST_PATH> is a path to the file with a list of files to be processed from the training dataset, <VAL_DS_PATH> is a path to the directory with the validation dataset, <VAL_LST_PATH> is a path to the file with a list of files to be processed from the training dataset, <OUTPUT_PATH> is a path to the output directory.

3. Post-processing of retrained models:
    1. copy all models to "models/VM_common/train_stages/train_stages/MSE_VariableRate_12/<BETA>", where <BETA> is the model's training beta.
    2. Run command "`./scripts/models_processing/all.sh`"

Post-processing of retrained models

1. Splitting training models on encoder/decoder/common parts.
2. Reduction of Z distributions to 64 unique distributions for all models
3. Weights reordering for supporting progressive decoding
4. Integerization of weights

		5 points BD-rate (0.12, 0.25, 0.5, 0.75, 1.0)								319%
		BD rate vs BOP-SOP								Max Bit Dev.
Test	AVG	msssim Torch	vif	fsim	nlpd	iw-ssim	vmaf	psnrHVS	Monotonicity	
Step1-BOP-SOP	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**TRUE**	**319%**
Step2-BOP-SOP	**0.3%**	**0.4%**	**0.3%**	**0.4%**	**0.3%**	**0.4%**	**0.3%**	**0.3%**	**TRUE**	**319%**
Step3-BOP-SOP	**0.3%**	**0.4%**	**0.3%**	**0.4%**	**0.3%**	**0.4%**	**0.4%**	**0.3%**	**TRUE**	**319%**
Step4-BOP-SOP	**0.4%**	**0.4%**	**0.4%**	**0.5%**	**0.4%**	**0.5%**	**0.4%**	**0.4%**	**TRUE**	**319%**

Resuming training from the pretrained weights

Command is:

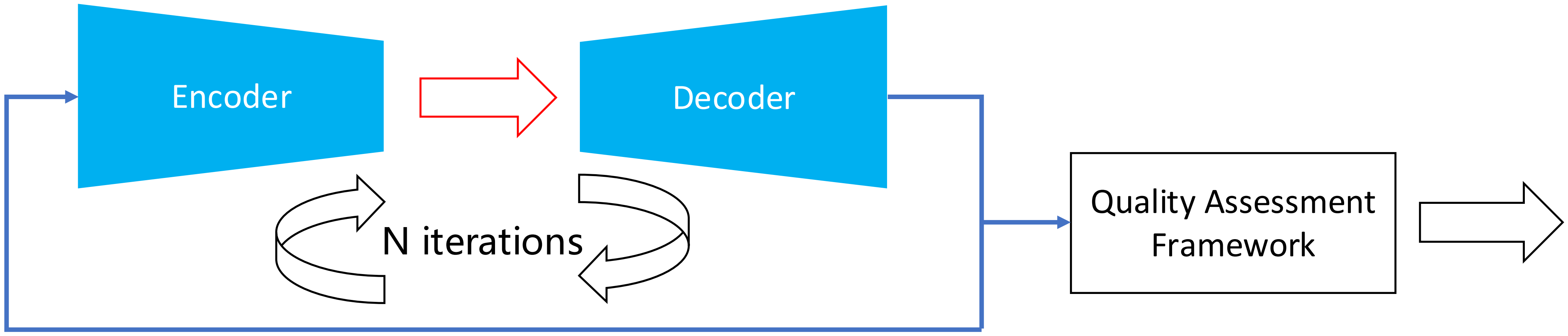
```
make train
```

or

```
python -m scripts.acc_train_scripts.acc_train_local \
       --data_dir <TRAIN_DS_PATH> \
       --lst <TRAIN_LST_PATH> \
       --val_data_dir <VAL_DS_PATH> \
       --val_lst <VAL_LST_PATH> \
       --train_url <OUTPUT_PATH> \
       --copy_to_train_url_dir <DIR_PRETRAINED_WEIGHTS> \
       --resume_from_stage <STAGE_NAME> \
       --frozen_part <FPART1>[ <FPART2> […]]
```

where <DIR_PRETRAINED_WEIGHTS> is a path to the directory with pretraing models (as instance, “models/VM_common/train_stages”), <STAGE_NAME> is the name of the stage which models will be used on resuming. An example is “MSE_VariableRate_12”, <FPART..> defines a list of freeze part of the codec and can be: “entropy” is for entropy part, “synthesis” is for the synthesis parts on decoder-side, “gain_unit” is for variable-rate functionality, “analysis” is for analysis part on encoder-side.

Multi-encoding and quality degradation

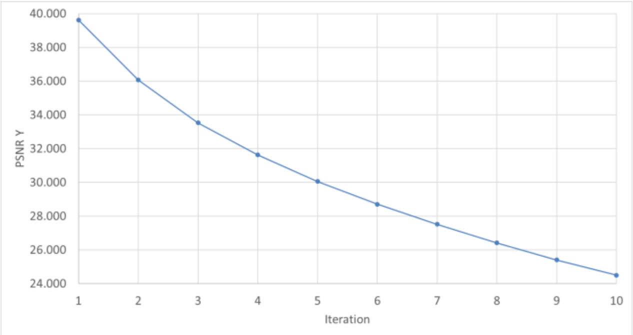


- Image after multi-encoding (Simple profile @ 0.75 bpp)

Image: 16002_TE_640x443_8bit_sRGB

After 1st iteration

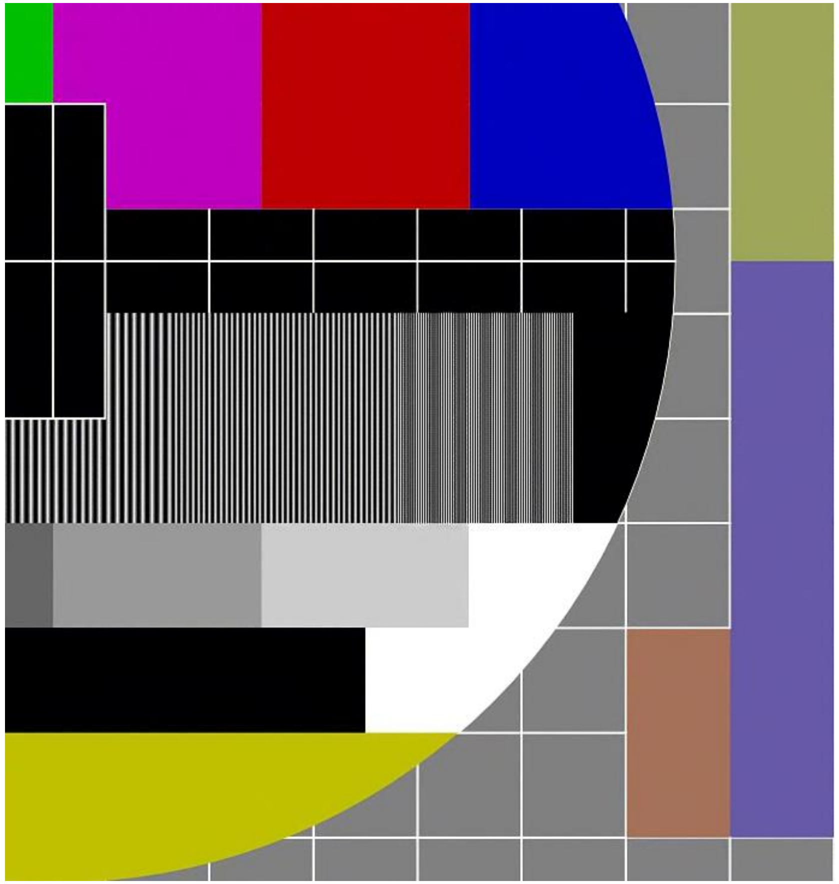
After 10th iteration (15 dB loss)



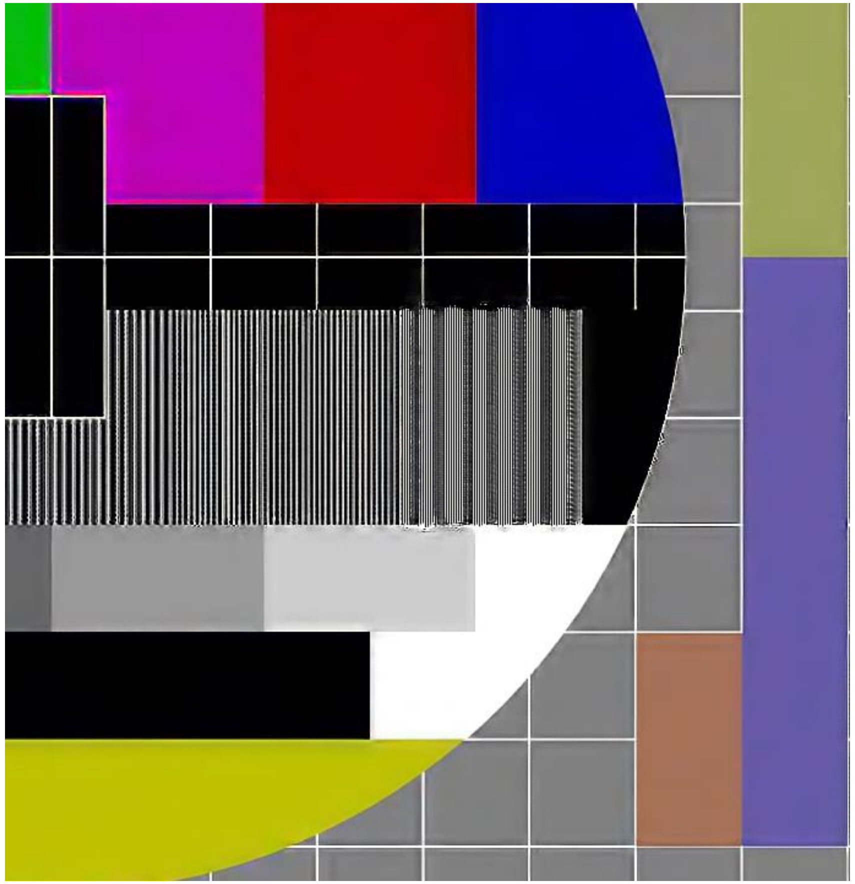
- Image after multi-encoding (Simple profile @ 0.75 bpp)

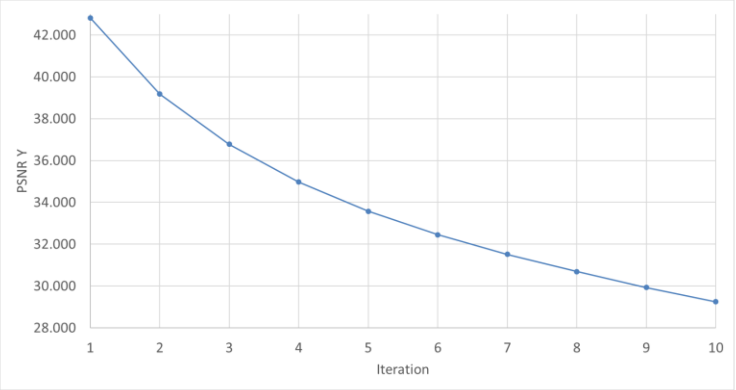
Image: 17006_TE_1920x1200_8bit_sRGB

After 1st iteration

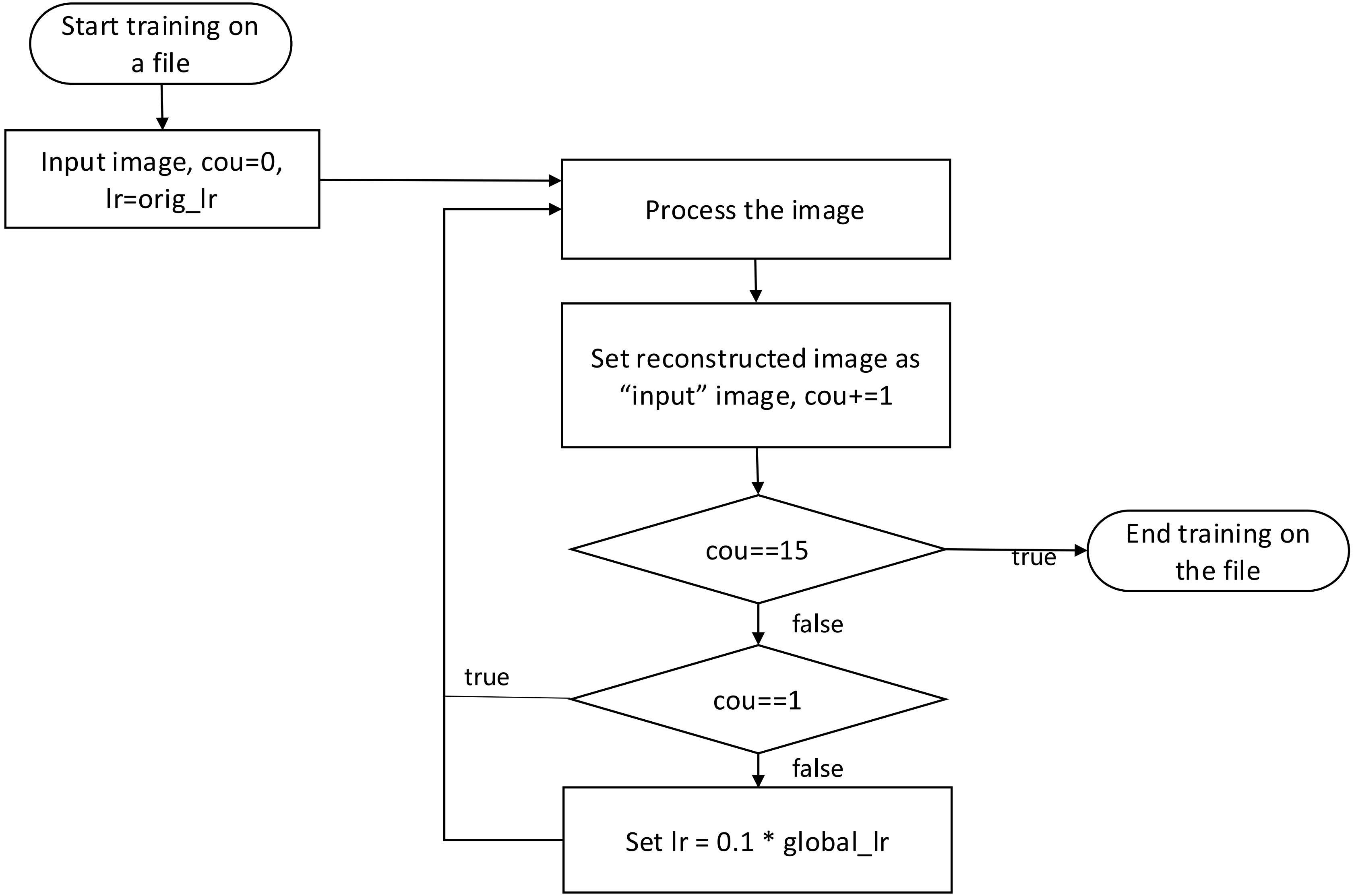


After 10th iteration (14 dB loss)





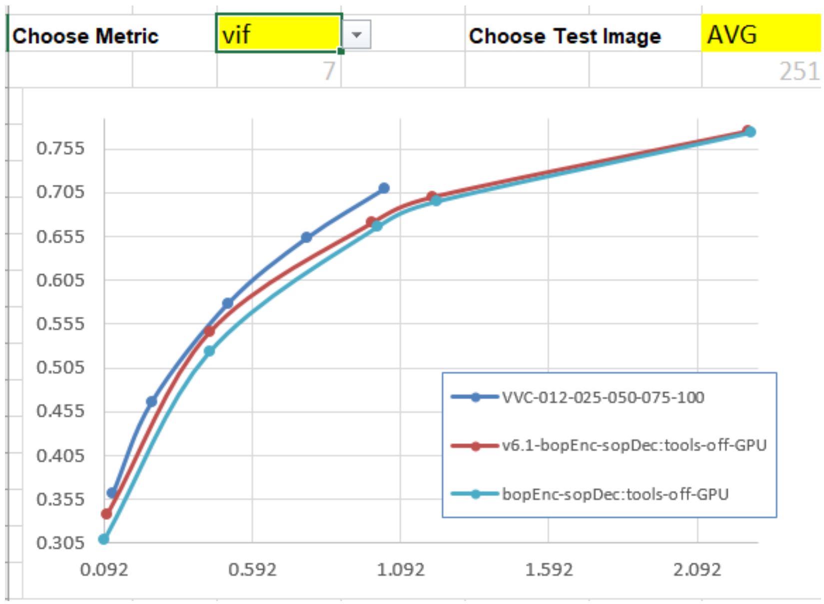
Training of encoder with multi-reencoding

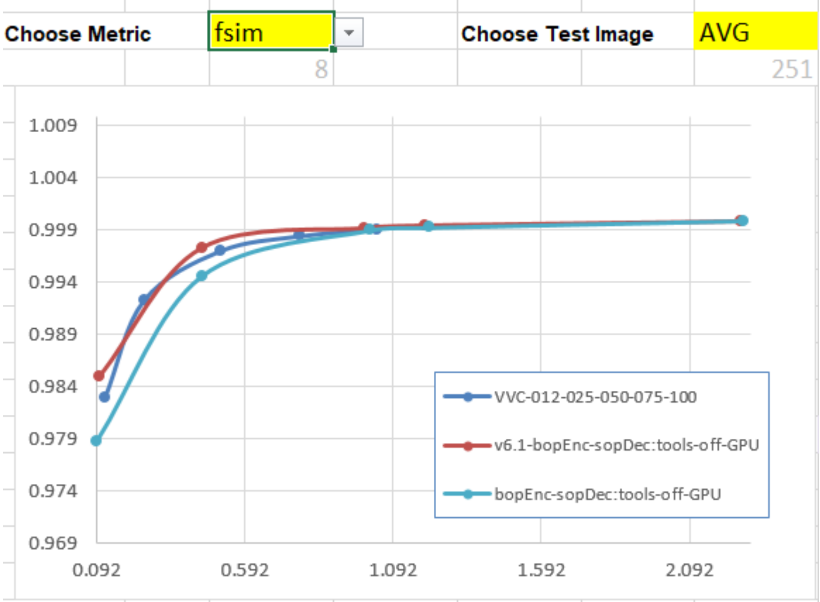


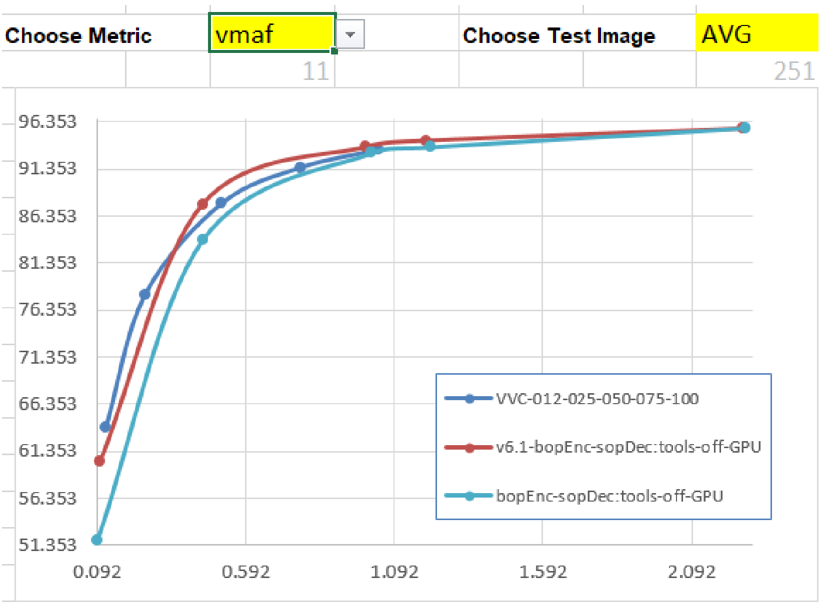
Performance on 50 images dataset (CTTC condition)

Microsoft Excel
ro-Enabled Worksl

		5 points BD-rate (0.12, 0.25, 0.5, 0.75, 1.0)								10%
		BD rate vs VVC-012-025-050-075-100								
Test	AVG	msssim Torch	vif	fsim	nlpd	iw-ssim	vmaf	psnrHVS	Monotonicity	Max Bit Dev.
v6.1-bopEnc-sopDec:tools-off-GPU	**-12.0%**	**-31.1%**	**6.7%**	**-15.1%**	**-12.1%**	**-26.5%**	**-7.3%**	**1.4%**	**TRUE**	**327%**
v6.1-bop:tools-off-GPU	**-16.7%**	**-33.5%**	**0.8%**	**-20.2%**	**-16.2%**	**-29.2%**	**-15.0%**	**-3.5%**	**TRUE**	**327%**
bop:tools-off-GPU	**8.4%**	**-17.3%**	**15.2%**	**50.7%**	**-4.1%**	**-14.3%**	**18.8%**	**10.1%**	**TRUE**	**321%**
bopEnc-sopDec:tools-off-GPU	**12.3%**	**-15.6%**	**20.4%**	**54.5%**	**-1.0%**	**-12.0%**	**25.7%**	**13.8%**	**TRUE**	**321%**







*Degradation of the performance comes from low rates

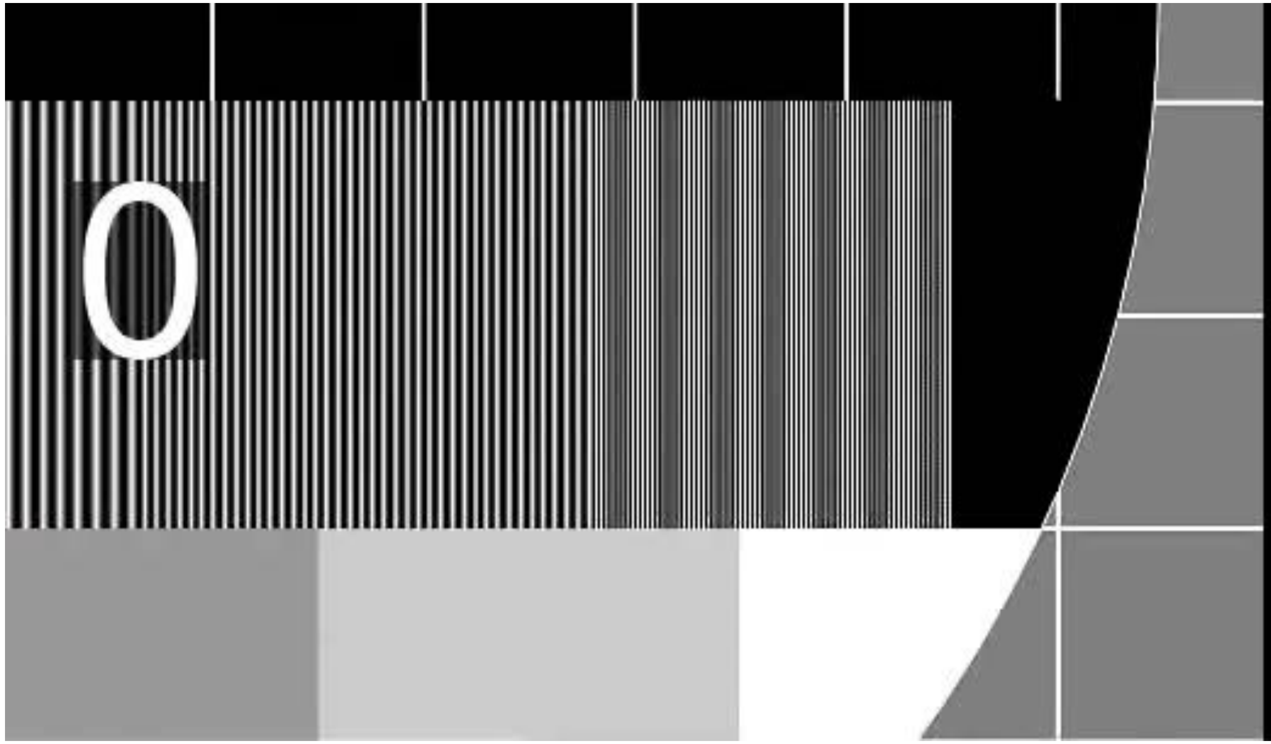
Comparison of DIS and retrained model 2 for 16002_TE_640x443_8bit_sRGB.png

DIS model

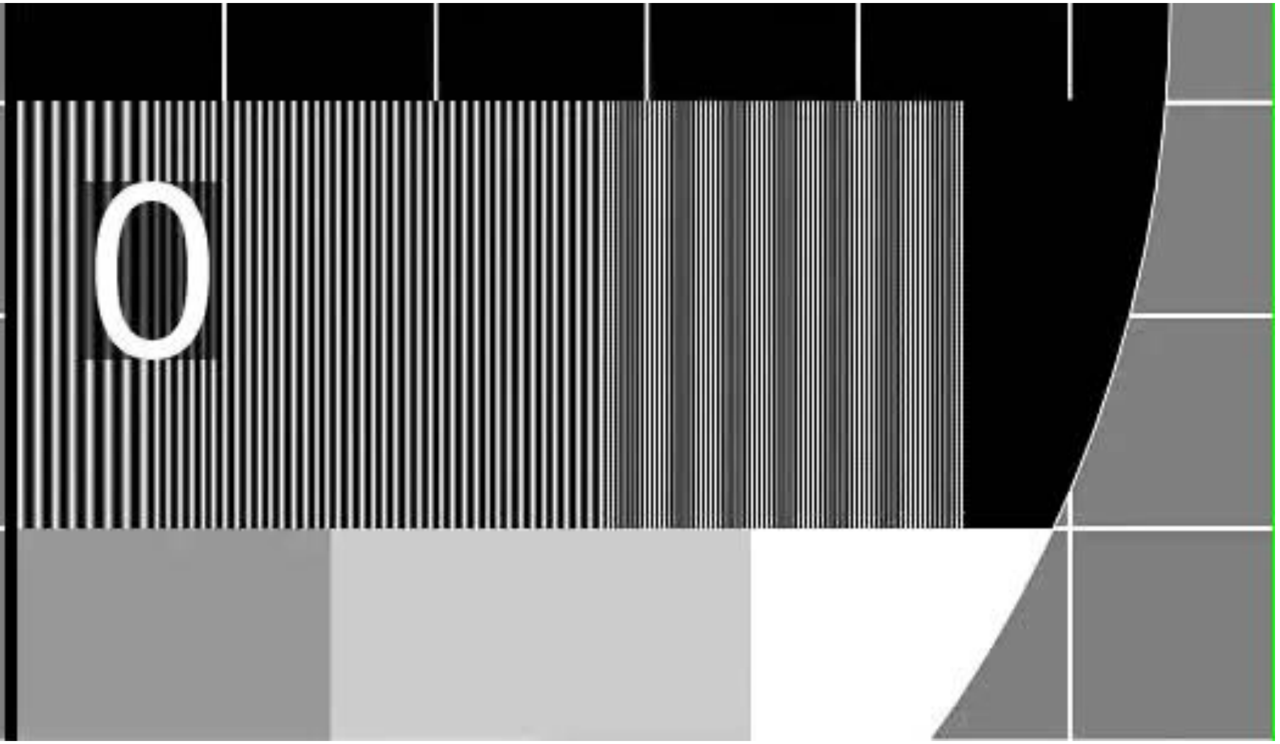
Retrained model

Comparison of DIS and retrained model 2 for 17006_TE_1920x1200_8bit_sRGB.png (cropped region)

DIS model



Retrained model



JPEG AI and intra coding in video

Comparison with VTM Intra

y:31.4
u:38.8
v:39.4

y:29.7
u:39.8
v:40.3

VTM 7 KB

Encoding time: 0:00:**12.9**57

QP = 36

JPEG AI VM6.1 7KB

Encoding time: 0:00:00.081

025 (CTTC)

Comparison with VTM Intra

VTM 7 KB

Encoding time: 0:00:**12.9**57

JPEG AI VM6.1 7KB

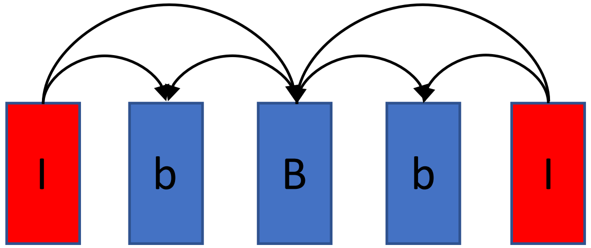
Encoding time: 0:00:00.081

Comparison with VVenC

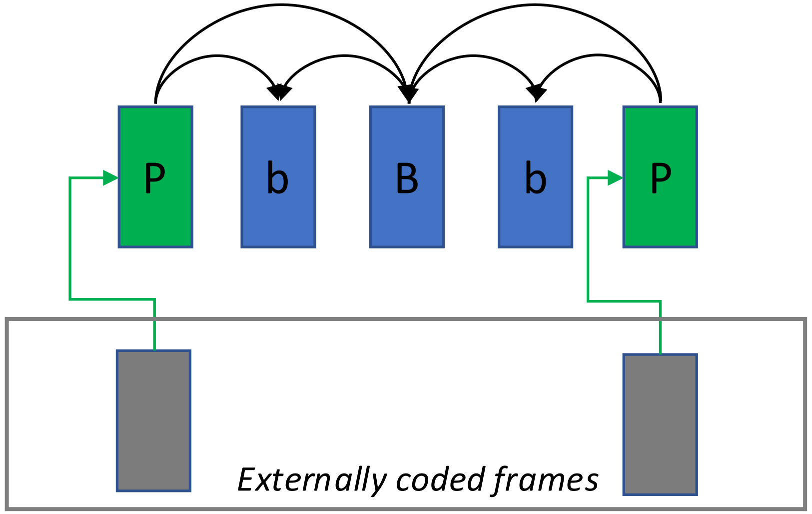
VVenC 7 KB JPEG AI 7KB
(perceptual optimization enabled)

Easy way to incorporate to video coding

HEVC v1 coder



HEVC v3 (scalable and multi-view)



HEVC v1 coder

HEVC v3 + E2E AI coded Intra frame

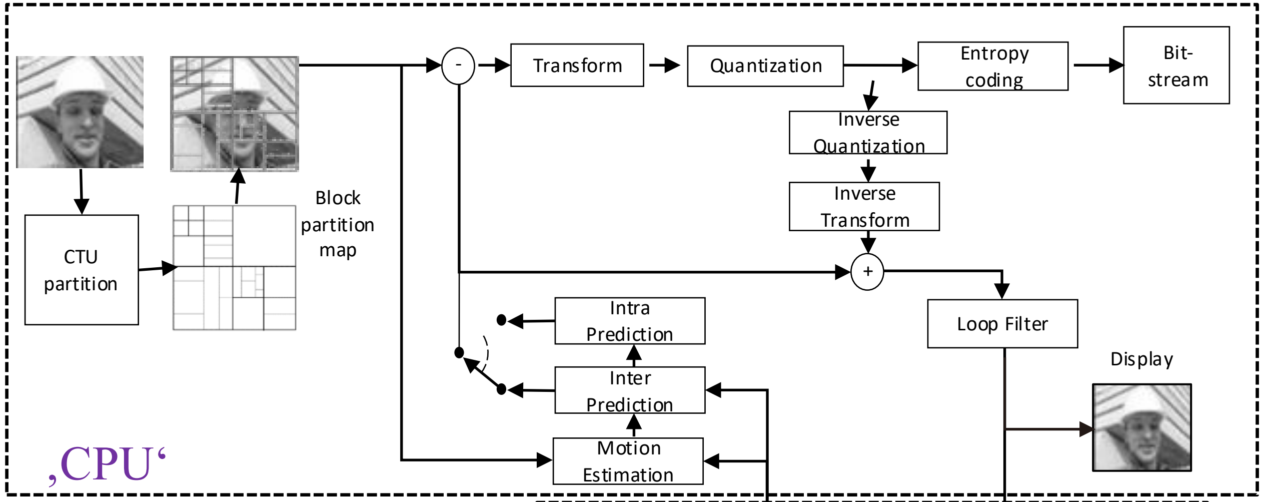
AI coded refPic can be added to any traditional codec

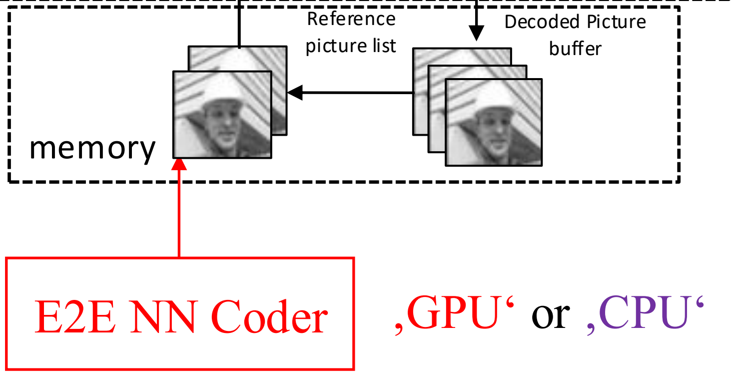
JVET-AL0196: VTM + AI coded RefPic

Evaluation under JVET CTC							
	Y-PSNR	U-PSNR	V-PSNR	EncT (CPU)	DecT (CPU)	EncT (GPU)	DecT (GPU)
All Intra cfg (AI)							
Class A1	0.8%	-7.4%	-10.7%	163%	16366%		
Class A2	-6.0%	-15.5%	-7.5%	121%	13046%		
Class B	-3.4%	-18.0%	-12.4%	84%	13220%		
Class C	-4.4%	-12.5%	-11.0%	58%	10095%		
Class E	-7.9%	-21.8%	-24.3%	86%	14140%		
Overall	**-4.1%**	**-15.2%**	**-13.0%**	**92%**	**13019%**	<100%	50..100%
Class D	-5.8%	-16.6%	-14.1%	63%	11921%		

Random Access cfg (RA)							
Class A1	-0.3%	-2.6%	-2.0%	98%	729%		
Class A2	-2.0%	-6.9%	-2.7%	94%	620%		
Class B	-1.3%	-8.8%	-5.0%	84%	596%		
Class C	-1.0%	-4.3%	-3.0%	99%	767%		
Overall	**-1.1%**	**-6.0%**	**-3.4%**	**92%**	**669%**	<100%	50..100%
Class D	-1.4%	-6.5%	-5.9%	98%	845%		

E2E AI coded RefPic is
DCVC-FM I-frame 525 kMac/pxl





Dr. Fabian Brand