POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Informatica

Tesi di Laurea Magistrale

Confronto prestazionale dei codificatori video VVC e AV1 usando un sistema HPC

Relatore
Prof. Enrico Masala

Candidato
Federico Baldassi

Anno Accademico 2024-2025

Sommario

Lo scopo di questa tesi è la valutazione delle prestazioni dei più moderni e sofisticati codificatori video (AV1 e VVC) per una realtà, come quella odierna ad elevato grado di richiesta di qualità e quantità, che si trova a dover fronteggiare le problematiche che ne derivano. Per raggiungere il mio scopo, ho analizzato il comportamento dei suddetti codificatori (comprimendo sequenze video di diversa tipologia) variandone di volta in volta la configurazione e valutandone il comportamento.
In generale, eseguire questo tipo di codifiche significa dover comprimere la stessa sequenza video più volte per ognuno dei formati di compressione che si vuole esaminare e questo richiede un'elevata capacità di calcolo. Per sopperire a questo problema ho utilizzato l'HPC (High Performance Computing) del Politecnico di Torino, ovvero un cluster di nodi di elaborazione in grado di eseguire un elevato numero di elaborazioni video in parallelo, riducendo così il tempo necessario per l'esecuzione di questi esperimenti. Questo tipo di elaboratori fornisce grande capacità di calcolo, ma, allo stesso tempo, richiede anche un attento lavoro di sviluppo di script di supporto da parte di chi li utilizza, che deve esserne in grado di sfruttarne appieno l'architettura nel modo più efficiente possibile. Al fine di eseguire in maniera automatizzata gli esperimenti della tesi, ho sviluppato degli script in grado di allocare le risorse necessarie nell'HPC, avviare le codifiche video sfruttando i codificatori scelti e successivamente raccoglierne i risultati, utilizzando metriche oggettive come VMAF e PSNR per la valutazione della qualità dei video compressi.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

Negli ultimi anni la fruizione di contenuti multimediali in alta definizione (video, film, serie tv, ecc...) ha avuto una crescita esponenziale. La rapida diffusione di dispositivi in grado di riprodurre video in risoluzione full-HD, UHD e 4K ha generato un notevole incremento della domanda di questo tipo di contenuti, con conseguente aumento dello spazio richiesto per la loro memorizzazione nei server.
Oltre a questo aspetto, è necessario tenere in considerazione l'aumento del carico richiesto alle reti di comunicazione, che spesso risultano sottodimensionate e non in grado di tenere il passo con la crescente qualità del servizio richiesta dagli utenti.
In questo scenario, lo sviluppo di nuovi formati di compressione video risulta vitale, al fine di ridurre la quantità di spazio occupato in memoria e banda richiesta, senza però sacrificare la qualità video dei contenuti.
Secondo i dati riportati nel Cisco Annual Internet Report (2018–2023), il traffico Internet globale è in continua crescita e lo streaming video ne rappresenta la componente predominante. Le stime di Cisco indicano che i contenuti video arriveranno a costituire oltre l'80% di tutto il traffico internet globale.
Alla luce di questi fatti, nasce la necessità di adottare codificatori video sempre più sofisticati ed efficienti, in grado di ridurre la quantità di bit da memorizzare nei server e da trasmettere su Internet, senza sacrificare però la qualità video finale.

1.1 Scopo e struttura della tesi

In questo contesto, la presente tesi si propone di analizzare le prestazioni di due dei più moderni codec presenti sul mercato (VVC ed AV1), valutandone l'efficienza di compressione con diverse configurazioni di bit-rate e risoluzione, in modo da individuarne, per quanto possibile, i punti di forza e gli aspetti migliorabili. Le prestazioni di AV1 e VVC sono state confrontate anche con quelle del più vecchio HEVC, utilizzato come punto di riferimento per la valutazione dei miglioramenti introdotti dai codificatori più moderni.
La tesi è strutturata nel seguente modo:

- un primo capitolo di breve introduzione alle tematiche trattate nella tesi;
- nel secondo capitolo vengono presentati i fondamenti della codifica delle immagini e video, con una breve spiegazione dei codec analizzati nella tesi;

- nel terzo capitolo vengono introdotti i principali metodi di valutazione oggettiva della qualità delle immagini e video;
- un quarto capitolo di spiegazione dell'utilizzo di un sistema HPC per l'esecuzione delle codifiche video, con descrizione degli script utilizzati e configurazione degli esperimenti;
- nel quinto capitolo viene fatta un'analisi dei risultati ottenuti, con un confronto prestazionale dei vari codec;
- infine, il sesto capitolo conclude il documento, riassumendo i risultati emersi nel capitolo 5, evidenziando i punti di forza dei vari codec analizzati e individuando le principali criticità migliorabili.

Capitolo 2

Codifica video

2.1 Introduzione alla codifica video

Quando si parla di codifica video si intende il processo di compressione e trasformazione di un video digitale in un formato specifico, in modo che risulti più leggero e quindi più adatto ad essere memorizzato e trasmesso su reti Internet. Infatti, l'obiettivo principale della codifica video è quello di ridurre la dimensione del file video originale, ma senza rinunciare a mantenere un'ottima qualità del video.
Il parametro principale per misurare lo spazio occupato da un video o la quantità di banda Internet richiesta per la sua trasmissione è chiamato bit-rate e la sua unità di misura è il bit per secondo.
Per fare un esempio, il bit-rate di una sequenza video UHD non compressa si può calcolare in questo modo:

Bit rate = Risoluzione x Frame rate x Profondità di colore x Canali = 5,97 Gbit/s

Parametri:

- Risoluzione UDH: 3840 × 2160 pixel
- Frame rate: 30 fps
- Profondità di colore: 8 bit
- Canali: 3 canali

Il risultato di 5,97 Gbit/s è eccessivamente elevato per poter essere trasmesso via Internet, per cui è necessario ridurlo, applicando la codifica video alla sequenza non compressa.
In generale, quando si parla di streaming video, una sequenza viene prima codificata, alla sorgente, per poi venire trasmessa utilizzando la rete internet e successivamente decodificata, alla destinazione, per poi essere visualizzata dall'utente finale.

2.2 Principi fondamentali della codifica video

I codificatori video, ovvero i software che si occupano della compressione delle sequenze video, possono essere differenziati in due macrocategorie: lossless e lossy.
Nel caso dei codificatori lossless, la compressione del video originale avviene senza perdita

di informazione, ovvero viene ridotta la dimensione del file senza però ridurre in alcun modo la qualità. Questa tipologia di tecniche permette di ottenere una riduzione della dimensione rispetto al file originale al massimo del 30-50%, il che tuttavia è troppo poco se, ad esempio, si vuole trasmettere su internet sequenze video full-HD con bit-rate dell'ordine di 1-2 Gbit/s. Per ottenere livelli di compressione sufficientemente elevati è necessario eliminare parte dell'informazione contenuta nella sequenza video, riducendone la qualità.
È qui che entrano in gioco i codificatori lossy che, per poter ottenere delle riduzioni sostanziali nei bit-rate, eliminano in maniera intelligente l'informazione del video, ovvero quella meno rilevante, in modo che la qualità finale percepita sia il più possibile vicina a quella originale. I codificatori lossy moderni permettono di passare da bit-rate di 1-2 Gbit/s a qualche decina di Mb/s, mantenendo un qualità video altissima, in alcuni casi praticamente indistinguibile dall'originale.

2.2.1 Codifica delle immagini

Per poter capire il funzionamento generale di un codificatore video, è necessario prima studiare come funziona la codifica delle immagini, in quanto i video sono sostanzialmente successioni di immagini statiche (o frame) riprodotti in sequenza al fine di creare l'illusione del movimento. Per prima cosa, per ottenere un buon livello di compressione dei singoli frame del video, vengono applicate le tecniche di compressione lossy delle immagini, che si suddividono in 4 step:

1. **Passaggio da rappresentazione RGB a Y'CbCr e sottocampionamento della crominanza**
   Di solito le immagini digitali sono in formato RGB, in cui ogni pixel è descritto da tre componenti che indicano l'intensità di ogni colore primario (Red, Green, Blue).
   Il modello Y'CbCr invece separa la luminosità (luminanza Y') dalle informazioni relative al colore (crominanza CbCr), questo perchè l'occhio umano è più sensibile alle variazioni di luminosità che a quelle di crominanza. La separazione in queste componenti permette di facilitare la riduzione dei dati della crominanza, senza compromettere la qualità percepita.
   Il processo di riduzione della risoluzione delle componenti relative al colore è chiamato "sottocampionamento della crominanza".
   Il sottocampionamento viene rappresentato mediante uno schema a tre cifre, come ad esempio il 4:2:2, in cui la prima cifra è il numero di campioni di luminanza per una determinata area (tipicamente 4 per i blocchi di 4 pixel in orizzontale), la seconda cifra invece è il numero di campioni di crominanza Cb e Cr nella prima riga del blocco, mentre la terza cifra è il numero di campioni di crominanza nella seconda riga.

2. **Trasformata DCT**
   La Trasformata Discreta del Coseno è un'operazione matematica molto utilizzata nella compressione dei segnali, in particolare nelle immagini e nei video. Essa permette di ottenere la rappresentazione nel dominio della frequenza di un segnale digitale rappresentato nel dominio dello spazio o del tempo. La DCT permette di separare le informazioni essenziali da quelle meno percettibili.
   Nella compressione delle immagini e video, la DCT viene applicata a piccoli blocchi dell'immagine (8x8 pixel), trasformando i valori di luminanza e crominanza in componenti in frequenza. Successivamente, le componenti ad alta frequenza, ovvero quelle che

rappresentano i dettagli fini delle immagini, possono essere eliminate o ridotte senza influenzare eccessivamente la qualità percepita.

3. **Quantizzazione**
   Il passo successivo alla DCT è la quantizzazione, che consiste nel processo matematico di riduzione o eliminazione dell'informazione delle componenti ad alta frequenza. Per ottenere ciò ogni coefficiente ottenuto dalla DCT viene moltiplicato per un valore specifico (compreso tra 0 e 1) preso da una matrice di quantizzazione e poi arrotondato. Per ottenere una maggiore riduzione delle componenti ad alta frequenza rispetto a quelle a bassa frequenza, è necessario che i coefficienti di quantizzazione associati alle componenti ad alta frequenza abbiano un valore inferiore rispetto agli altri e, in alcuni casi, pari a zero, per eliminare una determinata componente.
   Successivamente i coefficienti vengono tutti moltiplicati per un fattore chiamato parametro di quantizzazione Qp, necessario per regolare la compressione.

4. **Codifica dei dati**
   Dopo aver eliminato le informazioni meno importanti (utilizzando la trasformata DCT e la quantizzazione), viene applicata una serie di tecniche per ridurre la qualità di bit per codificare i dati. Queste tecniche sono la zig-zag re-ordering, la run-length encoding e la codifica entropica.
   Nel zig-zag re-ordering, i coefficienti della DCT vengono organizzati in un percorso che favorisce la disposizione sequenziale degli zeri, facilitando la compressione.
   Successivamente, la Run-Length Encoding (RLE) codifica le sequenze di zeri consecutivi in coppie (valore, conteggio delle ripetizioni), riducendo il numero di simboli da memorizzare.
   Infine, la codifica entropica, come Huffman o CABAC (Context-Adaptive Binary Arithmetic Coding), assegna codici più corti ai valori più frequenti e codici più lunghi a quelli meno comuni, ottimizzando ulteriormente la compressione dei dati. Queste tecniche lavorano insieme per ridurre il bit-rate mantenendo una buona qualità video.

2.3 Tecniche utilizzate nella codifica video

Nei video, oltre ad applicare tutte le tecniche di codifica delle immagini viste precedentemente, è anche possibile sfruttare le somiglianze tra i frame consecutivi, anche chiamata "ridondanza temporale", in modo tale da ridurre la quantità di dati necessaria a rappresentare una sequenza video. Le ridondanze temporali vengono sfruttate nella tecnica chiamata DPCM (Differential Pulse Code Modulation), in cui si effettua una predizione dei valori dei pixel del frame attuale basandosi su informazioni dei frame precedenti. In particolare, nella DPCM, non viene codificato direttamente il valore assoluto di un pixel, ma viene memorizzata la differenza tra il valore reale e quello predetto. Questa differenza è chiamata errore di predizione e si calcola come:

$$e(n) = x(n) - \hat{x}(n)$$

dove:

- x(n) è il valore effettivo
- $\hat{x}$(n) è il valore predetto

- e(n) è il valore differenziale memorizzato

Memorizzare le informazioni sotto forma di errori di predizione permette di avere dei valori più piccoli e più vicini tra loro rispetto alla memorizzazione dei valori assoluti, la cui distribuzione sarebbe più ampia e più difficile da comprimere. Avere valori più piccoli e concentrati permette infatti di utilizzare delle tecniche più efficienti di compressione dei dati, come ad esempio tecniche di codifica che assegnano meno bit agli errori a maggiore frequenza.
Inoltre, nel caso dei video, spesso accade che porzioni intere di frame rimangano uguali o simili tra i diversi frame, magari solo spostate di posizione. Questo accade soprattutto nei video di eventi sportivi, dove, ad esempio, un pallone cambia di posizione tra un frame e l'altro, mantenendo la sua forma e aspetto invariati.
Sfruttando queste caratteristiche dei frame consecutivi è possibile ridurre la quantità di informazioni da memorizzare, codificando una sola volta i pixel che descrivono la forma e l'aspetto del pallone e specificando, per i frame successivi, di quanto questi pixel si siano spostati di posizione. Lo spostamento delle porzioni di pixel viene descritto mediante i motion vector (o vettori di moto).
La tecnica che sfrutta i motion vector viene chiamata MC-DPCM (Motion Compensated DPCM). Nella MC-DPCM, per individuare la porzione di immagine simile nel frame precedente, l'immagine viene suddivisa in macroblocchi (MB) di pixel e viene effettuata la ricerca della posizione precedente del macroblocco, in modo da poterne descrivere il moto mediante i motion vector. Infine, se il macroblocco trovato nel frame precedente presenta delle discrepanze con quello attuale, viene codificata e memorizzata questa differenza, chiamata "prediction residual". Se, oltre ad avere a disposizione i frame passati, si ha anche a disposizione un certo numero di frame futuri, è possibile effettuare delle predizioni bi-direzionali (frame passati e futuri) in modo tale da ridurre ulteriormente gli errori di predizione.
Nella codifica video i frame vengono classificati in diversi tipi, in base al ruolo specifico che gli viene assegnato:

- I-frame (Intra-coded frame): è un frame completo e codificato indipendentemente dagli altri. Viene compresso solo con compressione intra-frame (come se fosse un'immagine JPEG).
- P-frame (Predicted frame): è un frame predetto da un frame precedente. Utilizza la compressione inter-frame con compensazione del moto.
- B-frame (Bidirectionally predicted frame): è un frame predetto da due frame di riferimento, uno precedente ed uno successivo. Usa infatti la compensazione del moto bidirezionale.

L'uso combinato di I-frame, P-frame e B-frame consente di ridurre drasticamente il bit-rate di una sequenza video. Nella compressione video, i frame I, P e B spesso vengono trasmessi in un ordine diverso rispetto a quello in cui vengono elaborati, questo per eseguire nel modo più efficiente possibile la compressione. Questo è necessario perchè i B-frame, che sfruttano sia i frame precedenti che quelli successivi, devono essere codificati solo dopo che i loro frame di riferimento sono disponibili.
Il codificatore analizza i frame, calcola la sequenza più efficiente, applica la compressione e trasmette i frame. Il decodificatore, invece, riceve i frame nell'ordine in cui sono stati codificati (coding order), li decodifica e li riordina in modo tale che possano essere visualizzati.
Questo processo permette di ottenere video di alta qualità con minore occupazione di banda o memoria, cosa fondamentale ad esempio in applicazioni quali lo streaming video.

2.4 Caratteristiche degli standard di codifica video: HEVC, AV1 e VVC

In questo parte del capitolo verranno analizzate le caratteristiche più importanti di tre standard di codifica video: HEVC (High Efficiency Video Coding), AV1 (AOMedia Video 1) e VVC (Versatile Video Coding).

2.4.1 HEVC (High Efficiency Video Coding)

HEVC, noto anche come H.265, è il successore rispetto a AVC/H.264, rispetto al quale presenta un notevole incremento di efficienza nella compressione (fino al 50% di riduzione nel bit-rate), a parità di qualità video.
Un'importante innovazione introdotta con HEVC è l'uso delle Coding Tree Units (CTU) che sostituiscono i macroblocchi tradizionali di AVC. Queste nuove strutture dati consentono una maggiore flessibilità nella suddivisione dei blocchi dell'immagine, permettendo al codificatore di adattarsi meglio alle caratteristiche visive del contenuto.
Con HEVC è stato anche introdotto un supporto diretto all'esecuzione parallela della compressione su più thread, cosa che in AVC non esisteva.
Un'altra importante novità è l'introduzione del supporto per risoluzioni fino a 8K HDR a 120 fps, cosa che rende questo standard ideale per trasmissione televisiva in alta definizione e lo streaming video-on-demand.
Uno dei principali fattori che ha limitato l'adozione di questo standard sono stati i notevoli costi di licenza per il suo utilizzo, cosa che ha spinto molte aziende a cercare alternative migliori e più accessibili come AV1.

2.4.2 AV1 (AOMedia Video 1)

AV1 è un codec sviluppato dall'Alliance for Open Media (AOMedia) che, a differenza di HEVC, è progettato per essere open-source e royalty-free. Rispetto a HEVC, AV1 offre un miglioramento in termini di efficienza di compressione pari a circa 30% in termini di riduzione di bit-rate, a parità di qualità video percepita. Questo lo rende una soluzione ideale per applicazioni di streaming video online come Netflix, Youtube e simili.
Due delle implementazioni principali di AV1 sono SVT-AV1 e AOM-AV1. SVT-AV1 (Scalable Video Technology for AV1) è ottimizzato per garantire prestazioni elevate su CPU multi-core di Intel, mantenendo comunque un'elevata qualità video. AOM-AV1, invece, è l'implementazione ufficiale di AOMedia che si focalizza di più sull'efficienza della compressione a scapito di tempi di codifica, che diventano più lunghi.

2.4.3 VVC (Versatile Video Coding)

VVC, anche chiamato H.266, è il successore di HEVC, rispetto al quale introduce un miglioramento nell'efficienza di compressione di circa 30-50% a seconda dei casi. Questo codec è stato sviluppato per gestire le nuove esigenze del settore, come i video 8K, HDR, VR e streaming adattivo.
Un'implementazione chiave di VVC è VVenC, ovvero un codec open-source sviluppato dal Fraunhofer HHI. Questo codec è progettato per garantire una compressione video altamente efficiente, che offra diverse ottimizzazioni e che si adatti in maniera efficace ai diversi scenari

di utilizzo, come ad esempio lo streaming video, l'archiviazione o le applicazioni immersive (visori VR, 8K, video a 360°, ecc...).
VVC, per ottenere una codifica più efficiente rispetto a HEVC, introduce diversi miglioramenti nelle tecniche di compressione dell'immagine, tra cui notevoli miglioramenti nell'intra-prediction, inter-prediction e nei filtri post-processing.
Il principale svantaggio di VVC e VVenC è la notevole complessità di computazione introdotte dai sofisticati miglioramenti apportati alle tecniche di compressione. Questa maggiore complessità richiede infatti harware potenti e tempi di codifica molto più lunghi rispetto a HEVC.
Il fatto di essere uno standard molto giovane implica che anche le sue implementazioni lo siano, per cui è possibile che ci sia ancora molto spazio per l'ottimizzazione. Un maggiore sviluppo del codice e la sua ottimizzazione sicuramente porterà, nel corso degli anni, ad avere delle implementazioni sempre più efficienti e veloci, come è avvenuto per gli altri codec del passato.

Capitolo 3

Metodi per la valutazione della qualità video

3.1 Introduzione alla valutazione della qualità video

Per poter valutare la qualità video esistono sostanzialmente due categorie di tecniche: quelle oggettive e quelle soggettive.
In questa tesi ho preso in considerazione solamente le tecniche oggettive, ovvero le metodologie basate su algoritmi.

3.2 Metodi oggettivi per l'analisi di qualità dei video

Questa famiglia di metodi è suddivisa in:

- Full reference (FR): necessitano dei dati originali non compressi.
- Reduced Reference (RR): viene richiesta solo una versione a bassa qualità del video originale o alcune caratteristiche estratte dal video originale. Questi metodi sono utili in scenari di trasmissione in cui i dati multimediali originali e non corrotti non sono dispobili al decoder.
- No reference (NR): non è necessario avere i dati multimediali originali non compressi. Sono molto meno affidabili rispetto a FR e RR.

In questa tesi le tecniche di valutazione oggettiva della qualità prese in considerazione sono solamente Full Reference, ovvero il PSNR e il VMAF.

3.3 PSNR

Il Peak Signal-to-Noise ratio è una delle principali misure per l'analisi della qualità di una immagine compressa rispetto all'originale. È basato sull'MSE (Mean Squared Error) e si calcola con la seguente formula:

$$PSNR_{db} = 10 \log_{10} \frac{(2^n - 1)^2}{MSE} \tag{3.1}$$

In questa formula, il $(2^n - 1)^2$ rappresenta il massimo valore assumibile dalla luminanza, rappresentata su n bit.
Il PSNR di una sequenza video si ottiene calcolando la media aritmetica dei valori di PSNR di ogni singolo frame della sequenza. In generale più è alto il valore del PSNR, maggiore è la somiglianza della sequenza compressa rispetto a quella originale.
In generale valori tra i 40 e i 50 dB indicano un'ottima qualità di compressione.

3.4 VMAF

Il Video Multimethod Assessment Fusion è una metrica avanzata per la valutazione della qualità video sviluppata da Netflix e rilasciata come open-source nel 2016, in collaborazione con università e laboratori di ricerca.
Il suo obiettivo è valutare la qualità percepita di un video in modo oggettivo ed affidabile.
VMAF utilizza un approccio basato su machine learning per combinare più metriche di qualità video tradizionali:

- VIF: valuta la perdita di fedeltà dell'informazione;
- DLM: misura la perdita di dettagli e le alterazioni che distraggono l'attenzione dello spettatore;
- MCPD: misura la differenza temporale tra fotogrammi sulla componente di luminanza.

Le caratteristiche sopra descritte vengono combinate utilizzando una regressione basata su SVM (Support Vector Machine) per fornire un unico punteggio di output compreso tra 0 e 100 per ogni fotogramma video, dove 100 indica una qualità identica al video di riferimento.
Questi punteggi vengono poi aggregati temporalmente sull'intera sequenza video mediante la media aritmetica, al fine di ottenere un punteggio complessivo di opinione media differenziale (DMOS).
In generale un VMAF superiore a 90 indica una qualità praticamente indistinguibile dall'originale.

3.5 Curve rate-quality

Le curve rate-quality sono un utile strumento per visualizzare graficamente il compromesso tra bitrate (rate) e qualità in un sistema di compressione video.
Queste curve descrivono come la quantità di dati utilizzata per codificare una sequenza video (bit-rate) influenzi la qualità percepita della sequenza video risultato della compressione.
Generalmente sull'asse delle ascisse (x) troviamo il bit-rate (espresso in bit/s) mentre sulle ordinate (y) c'è la metrica che misura la qualità (VMAF, PSNR, ecc...).
Le curve rappresentate in questo modo dovrebbero avere un andamento crescente, perchè ad un aumento del bit-rate impostato per la codifica dovrebbe corrispondere un aumento della qualità della sequenza video risultante.

Capitolo 4

Utilizzo dell'HPC per la codifica video e valutazione oggettiva della qualità

4.1 Cos'è un High Performance Computing (HPC)

In generale, con High Performance Computing si intende l'esecuzione di calcoli complessi realizzata in parallelo su più server ad alte prestazioni.
Questi gruppi di server collegati tra di loro tramite una rete vengono chiamati cluster. Ogni server di questi cluster viene chiamato nodo di elaborazione.
Un'altra importante componente di questi sistemi è lo scheduler, che si occupa di monitorare le risorse disponibili ed assegnare in maniera efficiente il carico di lavoro ai vari nodi di elaborazione.

4.2 Il cluster LEGION del Politecnico di Torino

In questa tesi ho utilizzato il cluster LEGION del Politecnico di Torino, ovvero un cluster InfiniBand con le seguenti caratteristiche:

Architecture	Linux Infiniband-EDR MIMD Distributed Shared-Memory Cluster
Node Interconnect	Infiniband EDR 100 Gb/s
Service Network	Gigabit Ethernet 1 Gb/s
CPU Model	2x Intel Xeon Scalable Processors Gold 6130 2.10 GHz 16 cores
GPU Model	4x nVidia Tesla V100 SXM2 - 32 GB - 5120 cuda cores (on 6 nodes)
Sustained performance (Rmax)	21.094 TFLOPS (last update: july 2019) [14 nodes]
Peak performance (Rpeak)	110.865 TFLOPS (last update: july 2020) [57 nodes]
Computing Cores	1824
Number of Nodes	57
Total RAM Memory	21.888 TB DDR4 REGISTERED ECC
OS	Centos 7.6 - OpenHPC 1.3.8.1
Scheduler	SLURM 18.08.8

Tabella 4.1: Caratteristiche tecniche del cluster legion del Politecnico di Torino

Per poter eseguire le codifiche nell'HPC è necessario sottomettere dei job allo scheduler. Ogni job può eseguire una o più codifiche in base a come si decide di suddividere il carico di lavoro.
In generale, è importante scegliere accuratamente quante codifiche assegnare ad ogni job, in modo da massimizzare l'esecuzione parallela del lavoro.
Nel caso dell'HPC del Politecnico, tutti i jobs devono essere sottomessi allo scheduler utilizzando il comando *sbatch*, al quale viene passato un file di configurazione contenenente le direttive necessarie alla configurazione del job. Un'esempio di script *sbatch* è il seguente:

```
1  #!/bin/bash
2  #SBATCH --job-name=vvc_hevc
3  #SBATCH --mail-type=ALL
4  #SBATCH --mail-user=federico.baldassi@studenti.polito.it
5  #SBATCH --partition=global
6  #SBATCH --time=60:00:00
7  #SBATCH --nodes=1
8  #SBATCH --ntasks-per-node=1
9  #SBATCH --cpus-per-task=4
10 #SBATCH --output=job_%j.log
11 #SBATCH --mem=10G
12 #SBATCH --workdir=/home/fbaldassi
13
14 file_path=$1
15 config_path=$2
16 bit_rate_param=$3
17 module load singularity/3.2.1
18 singularity exec ubuntu_vvc.sif \
19 bash -c "cd scripts && sh encoding.sh \
20 ../${file_path} ../${config_path} ${bit_rate_param}"
```

La prima parte contiene del direttive per lo scheduler, mentre i comandi non preceduti da *#SBATCH* costituiscono effettivamente il codice che il job deve eseguire.

4.3 Singularity container

Singularity è un software gratuito e open source in grado di realizzare la virtualizzazione a livello di sistema operativo, mediante la creazione di container.
Una delle caratteristiche principali di Singularity è la facilità nella portabilità dei container da un sistema all'altro.
È possibile, ad esempio, creare il proprio container sul proprio personal computer, installarci tutti i pacchetti software necessari a eseguire i propri calcoli scientifici, esportare il container in un HPC ed eseguire i propri esperimenti senza difficoltà.
Per creare un container Singularity è sufficiente creare un file di configurazione contenente i comandi di installazione dei software che si intende utilizzare, eseguire il comando di "building" del container e poi esportare il file generato sull'HPC.
Per la creazione del container ho utilizzato Singularity versione 4.2.0-rc.1+114-g1464c9a23, con il comando "sudo singularity build ubuntu_vvc.sif ubuntu_vvc.def".
Questo comando genera un container .sif a partire da un file .def contenente la configurazione del container ed i comandi necessari ad installare al suo interno i pacchetti software che si

intende utilizzare. Il file di building che ho utilizzato per generare il container Singularity è riportato in Appendice A.

4.4 FFmpeg e librerie utilizzate per la codifica

FFmpeg è un software open-source costituito da un insieme di tool e librerie utili all'elaborazione di flussi multimediali, file video e audio.
La versione di FFmpeg che ho utilizzato è la N-118350-ge20ee9f9ae.
Per le codifiche ho utilizzato ffmpeg, componente di FFmpeg, ovvero un software a linea di comando in grado di convertire le sequenze video da un formato ad un altro.
Di seguito, nella tabella 4.2, sono riportate le librerie utilizzate da ffmpeg per le codifiche, con relativa versione.

Codec	**Libreria**	**Versione**
HEVC	libx265	3.5+1-f0c1022b6
AOM-AV1	libaom-av1	3.8.2
SVT-AV1	libsvtav1	2.3.0-100-g783c3f1f
VVenC	libvvenc	1.12.1

Tabella 4.2: Specifiche dei vari codec utilizzati

Per ognuno di questi codec esistono diversi preset di codifica che è possibile specificare in fase di configurazione: un preset più lento ad esempio impiegherà più tempo a comprimere la sequenza rispetto ad un preset più veloce, ma la sequenza compressa risultante avrà una qualità migliore.
I preset disponibili per i diversi codificatori sono i seguenti:

- **HEVC**: ultrafast, superfast, veryfast, faster, fast, medium, slow, slower, veryslow, placebo;
- **AOM-AV1**: 0-8 (0 minima velocità - 8 velocità massima);
- **SVT-AV1**: 0-13 (0 minima velocità - 13 velocità massima)
- **VVenC**: faster, fast, medium, slow, slower

4.5 Script di supporto all'esecuzione delle codifiche e all'analisi oggettiva dei risultati

Al fine di eseguire nel modo più automatizzato possibile gli esperimenti di questa tesi, ho sviluppato alcuni script di supporto (il cui codice completo è riportato in Appendice A). Gli script per la codifica e raccolta dei risultati sono fondamentalmente tre:

- encoding.sh
- encoding_bit_rate.sh

- vmaf_bit_rate.sh

Encoding.sh si occupa di richiamare gli altri script di codifica e di valutazione della qualità. Encoding_bit_rate.sh invece è lo script che effettivamente richiama il comando ffmpeg di codifica e decodifica delle sequenze. Al suo interno infatti vengono importati tutti i parametri necessari da passare al comando ffmpeg, compresi i parametri specifici dei codificatori e i parametri delle varie sequenze (risoluzione, frame-rate, ecc...).
Vmaf_bit_rate.sh invece è lo script che esegue il calcolo del VMAF sulle sequenze compresse, calcola i bit-rate e memorizza i risultati nel csv. Il bit-rate viene calcolato dividendo la dimensione (in megabyte) del file compresso per la sua durata (in secondi). Per l'analisi della qualità video ho utilizzato VMAF versione 3.0.0, che supporta sia il calcolo del VMAF che del PSNR. Per fare ciò mi è stato sufficiente richiamare l'eseguibile VMAF specificando le metriche VMAF e PSNR, oltre agli altri parametri di configurazione.
I vari script importano i parametri per l'esecuzione delle codifiche da un unico file .cfg.
Un esempio di file di configurazione utilizzato in questa tesi è questo:

```
1  # config.cfg
2
3  ENCODE="on"
4  DECODE="on"
5  EVALUATE="on"
6  CODECS="AV1-SVT"
7  BIT_RATES="10000"
8  PIX_FMT_FOR_ENC_DEC="yuv420p"
9  PIX_FMT_FOR_VMAF="420"
10 BIT_DEPTH_FOR_VMAF="8"
11 HEVC_PRESET="slow"
12 VVC_PRESET="slow"
13 AV1_PRESET="2"
14 AV1_SVT_PRESET="2"
15 VVC_ENCODING_MODE="ABR"
16 FFMPEG_VVC_PARAMETERS="-qpa 1"
17 VVENC_PARAMETERS=""
18 FFMPEG_HEVC_PARAMETERS=""
19 HEVC_PARAMETERS=""
```

I file di configurazione contengono i parametri fondamentali per l'esecuzione dei comandi *ffmpeg* e *vmaf*.
I tre parametri *ENCODE*, *DECODE*, *EVALUATE* servono ad indicare se siano da eseguire la codifica e/o la decodifica e/o la valutazione della qualità della sequenza compressa.
Impostando i vari parametri su "off" si disattiverà la relativa funzionalità.
Sono poi presenti le configurazioni dei parametri relativi al sottocampionamento della crominanza (sia per ffmpeg che per vmaf), la profondità di colore, i preset di codifica per i vari codec e i parametri per specificare delle opzioni specifiche per i vari codificatori.
Altri parametri come ad esempio il frame-rate o la risoluzione della sequenza vengono direttamente estratti dal file originale di input (.y4m).

4.6 Configurazioni delle codifiche

Per l'esecuzione delle codifiche ho utilizzato la modalità a bit-rate variabile (VBR), in cui il bit-rate varia dinamicamente in base alla complessità del contenuto da comprimere.
In questa modalità, le scene più complesse, ad esempio quelle dove è presente molto movimento o un'alta complessità dei dettagli, vengono codificate ad un bit-rate più elevato per poterne preservare la qualità, a differenza di scene più statiche o meno dettagliate, che possono ricevere un bit-rate inferiore e risparmiare spazio.
La modalità VBR permette di ottenere un migliore rapporto qualità/dimensione del file rispetto alla modalità a bit rate costante, ma richiede uno sforzo maggiore in termini di tempo di esecuzione della codifica.
Ognuno dei codificatori è stato testato utilizzando i preset in tabella 4.3.

Codificatore	Preset
HEVC	Slow (2), medium (5), ultrafast (8)
AOM-AV1	2, 4, 8
AOM-SVT	2, 4, 13
VVenC	slow (2), medium (3), faster (5)

Tabella 4.3: Preset utilizzati per le codifiche

4.7 Sequenze video testate

Nome sequenza	**Risoluzione**	**Frame rate**	**Durata**
CSGO_30fps_30sec_v2	1920x1080	30 fps	30 s
Hearthstone_30fps_30sec_v2	1920x1080	30 fps	30 s
bigbuck_bunny_8bit	4000x2550	60 fps	10 s
cutting_orange_tuil	1920x1080 3840x2160	59,94 fps	10 s
Daydreamer_SDR_8s	1920x1080 3840x2160	60 fps	8 s
Sparks_cut_13	1920x1080 3840x2160	59,94 fps	8 s
Sparks_cut_15	1920x1080 3840x2160	59,94 fps	8 s
vegetables_tuil	1920x1080 3840x2160	59,94 fps	10 s
water_netflix	1920x1080 3840x2160	59,94	10 s

Tabella 4.4: Specifiche tecniche delle sequenze analizzate

Le sequenze video utilizzate per gli esperimenti di questa tesi si possono suddividere in 2 categorie: sintetiche e non sintetiche.
Le sequenze video sintetiche sono state generate artificialmente dalla registrazione di gameplay di videogiochi (CSGO ed Heartstone) o estratte da sequenze video animate (big buck

bunny), mentre quelle non sintetiche (o naturali) derivano da riprese reali, catturate da fotocamere o dispositivi di acquisizione video.
Tutte le sequenze analizzate presentano una profondità di colore di **8 bit** e schema di sottocampionamento della crominanza pari a **4:2:0**.

Figura 4.1: CSGO_30fps_30sec_v2

Figura 4.2: Hearthstone_30fps_30sec_v2

Figura 4.3: bigbuck_bunny_8bit

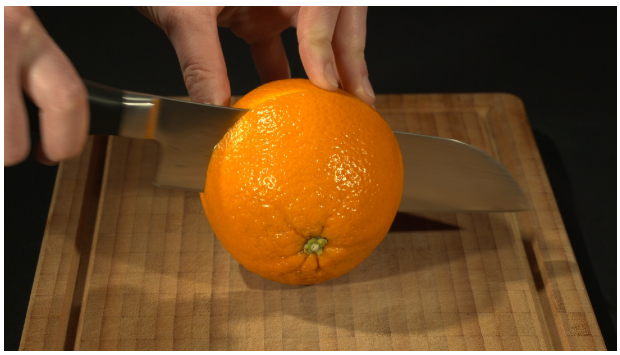

Figura 4.4: cutting_orange_tuil

Figura 4.5: Daydreamer_SDR_8s

Figura 4.6: Sparks_cut_13

Figura 4.7: Sparks_cut_15

Figura 4.8: vegetables_tuil

Figura 4.9: water_netflix

Capitolo 5

Risultati ottenuti

5.1 Risultati ottenuti

In questo capitolo ho riportato i risultati ottenuti dall'esecuzione delle codifiche. Tutti i risultati sperimentali e gli script di supporto utilizzati per le analisi sono disponibili nel seguente repository GitHub: https://github.com/fede1997483/Encoder-Performance-Tools.
Per ogni sequenza, ci sono i grafici relativi alla misura del PSNR, del VMAF e del tempo medio di codifica (calcolato sul tempo che la CPU ha trascorso in user mode).
Infatti, oltre ai valori di VMAF e PSNR, anche il tempo di codifica è un parametro rilevante per valutare le prestazioni di un codec video.
Le prime due sequenze, ovvero *CSGO_30fps_30sec_v2* e *Hearthstone_30fps_30sec_v2* sono state codificate solamente in risoluzione FULL-HD.
Per quanto riguarda invece tutte le altre sequenze, sono state invece compresse sia in FULL-HD che in 4K, per evidenziare come variano i bit-rate di codifica all'aumentare della risoluzione. Le sequenze video non compresse FULL-HD sono state ottenute riducendo la risoluzione delle sequenze originali 4K, con il seguente comando ffmpeg:

ffmpeg
-i file_input.y4m
vf scale=1920x1080:flags=lanczos:param0=3
sws_flags lanczos+accurate_rnd+full_chroma_int
pix_fmt yuv420p
file_output.y4m

In questo modo viene effettuato un ridimensionamento ad alta qualità delle sequenze video 4K.
Per quanto riguarda i grafici, invece, è importante evidenziare che per le sequenze compresse sia in FULL-HD che in 4K sono stati riportati i grafici con i risultati per entrambe le risoluzioni.

5.1.1 CSGO_30fps_30sec_v2

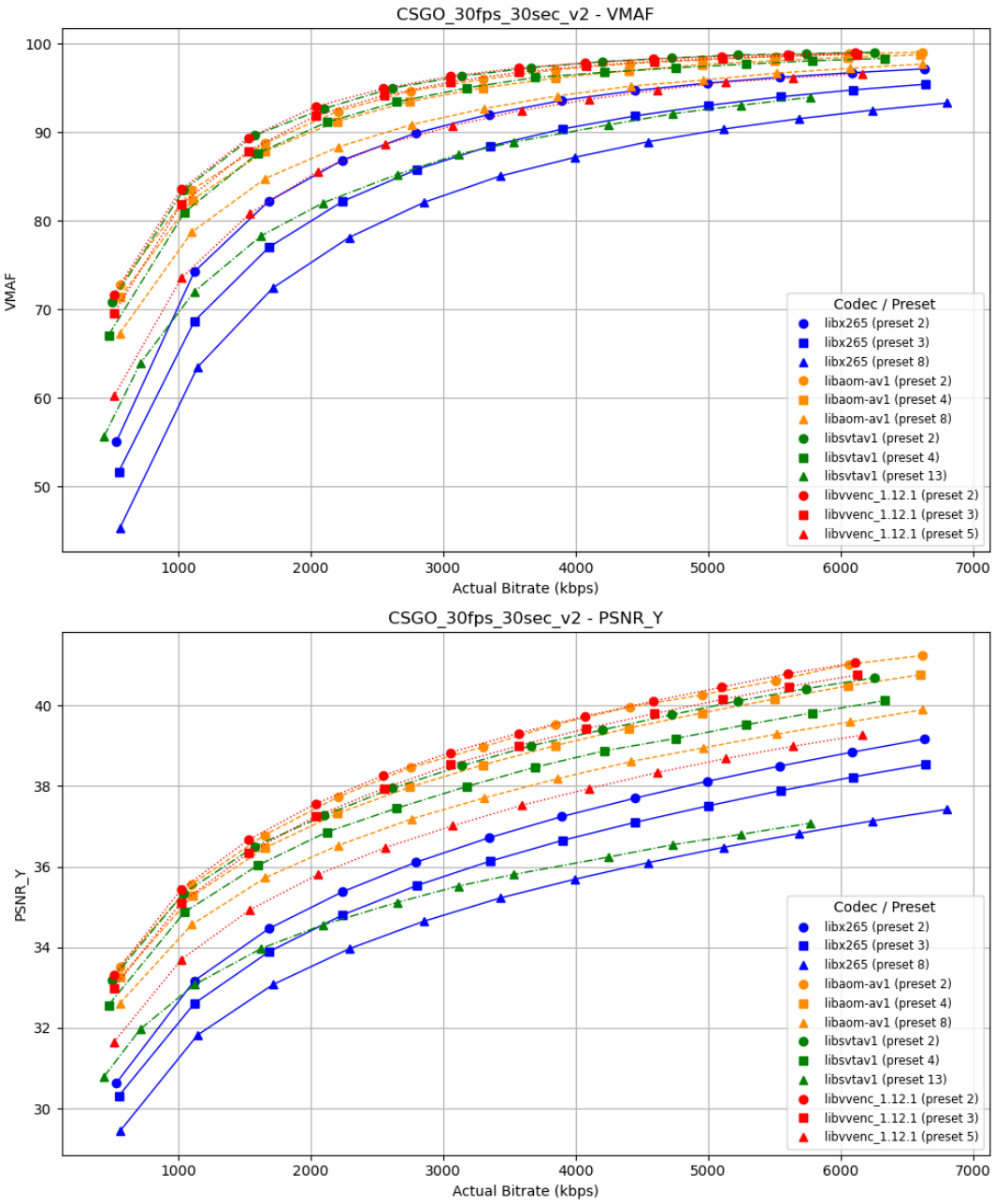

Figura 5.1: Grafici VMAF e PSNR per CSGO_30fps_30sec_v2

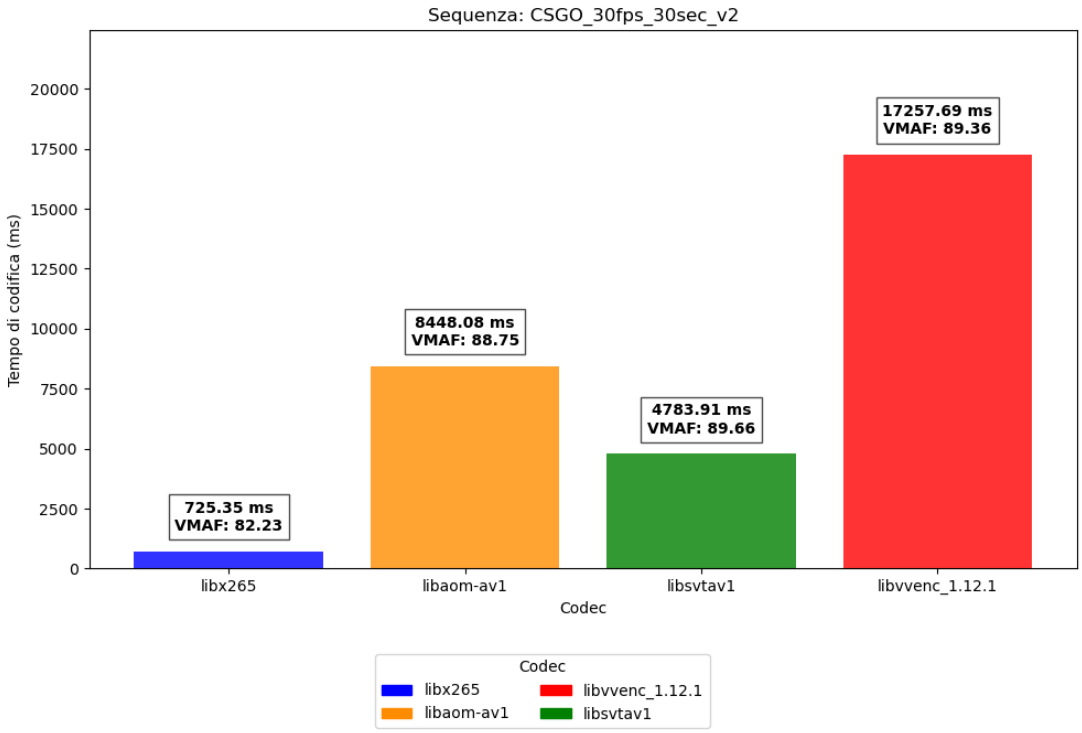

Figura 5.2: Grafico tempo di codifica per CSGO_30fps_30sec_v2 (1500 KB/s)

5.1.2 Hearthstone_30fps_30sec_v2

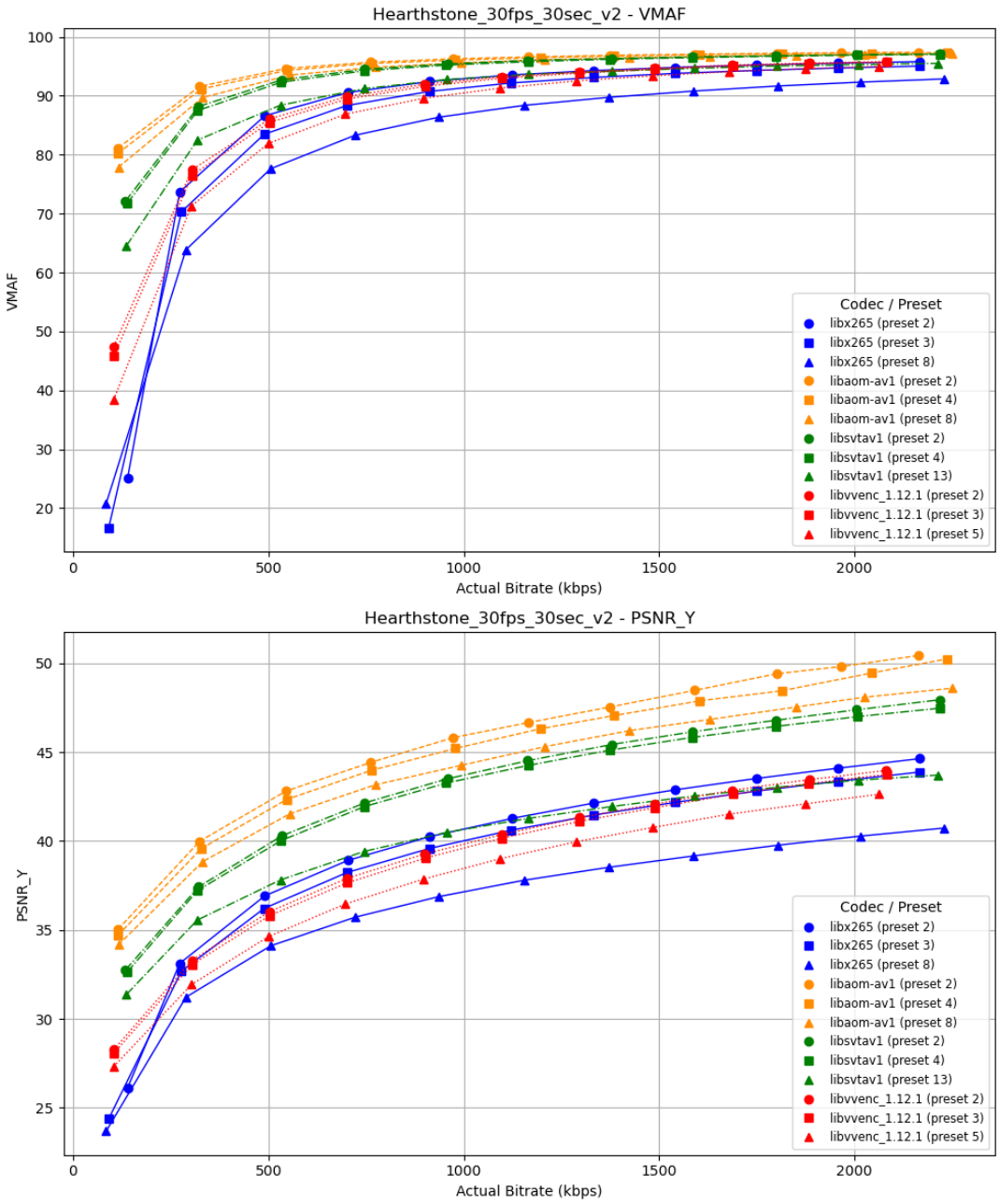

Figura 5.3: Grafici VMAF e PSNR per Hearthstone_30fps_30sec_v2

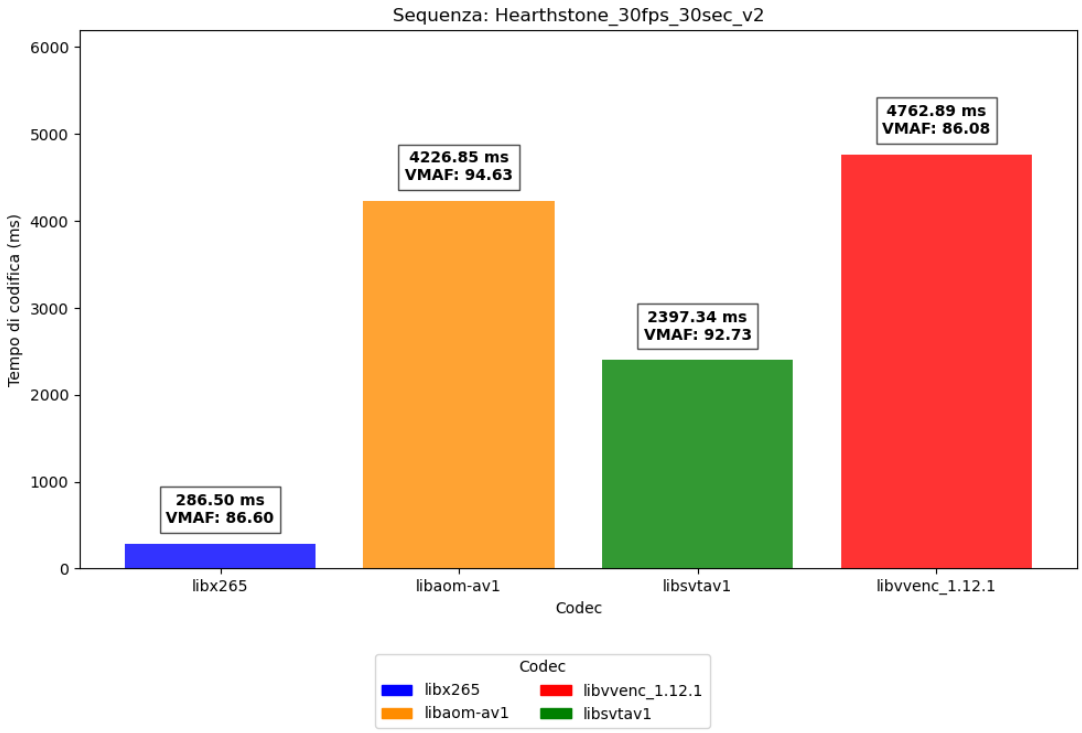

Figura 5.4: Grafico tempo di codifica per Hearthstone_30fps_30sec_v2 (500 KB/s)

5.1.3 bigbuck_bunny_8bit

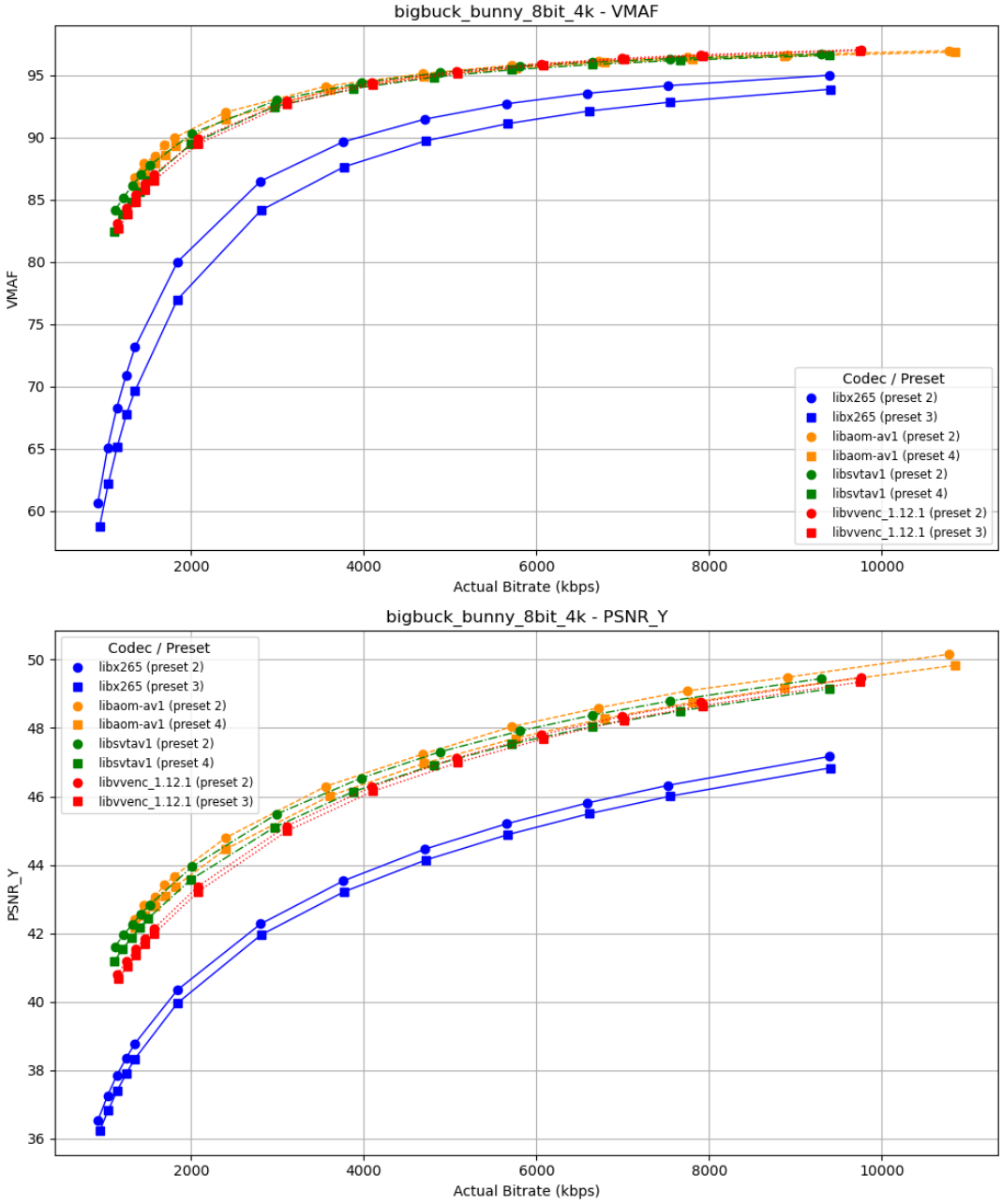

Figura 5.5: Grafici VMAF e PSNR per bigbuck_bunny_8bit_4k

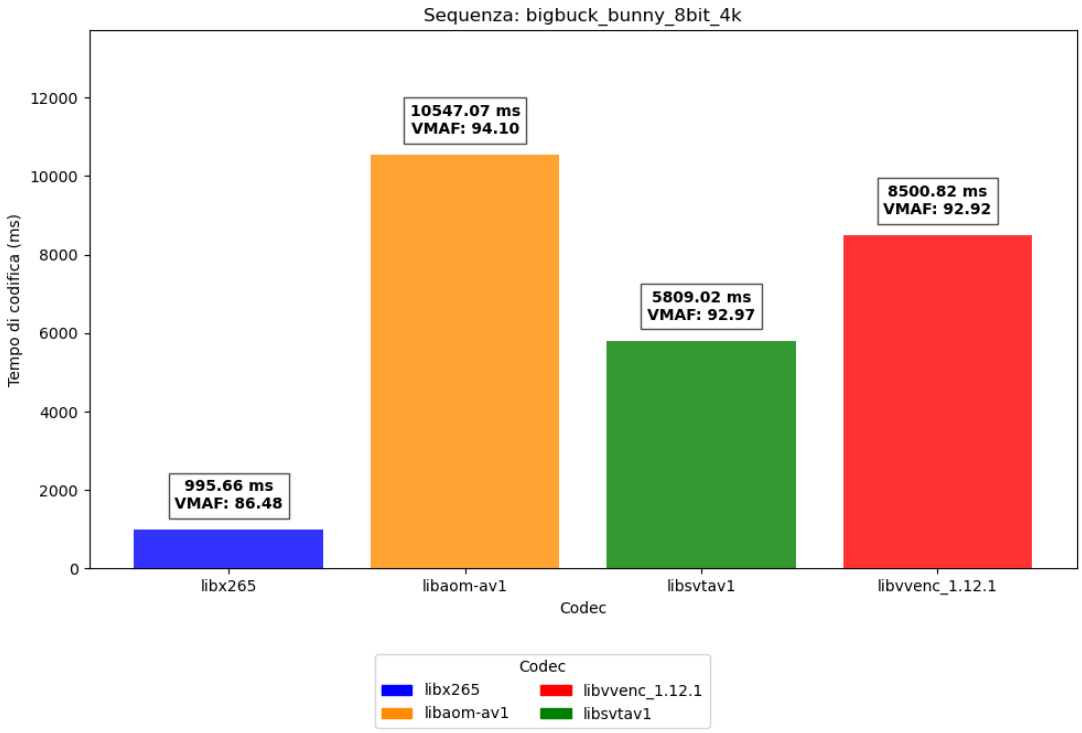

Figura 5.6: Grafico tempo di codifica per bigbuck_bunny_8bit_4k (3000 KB/s)

5.1.4 cutting_orange_tuil

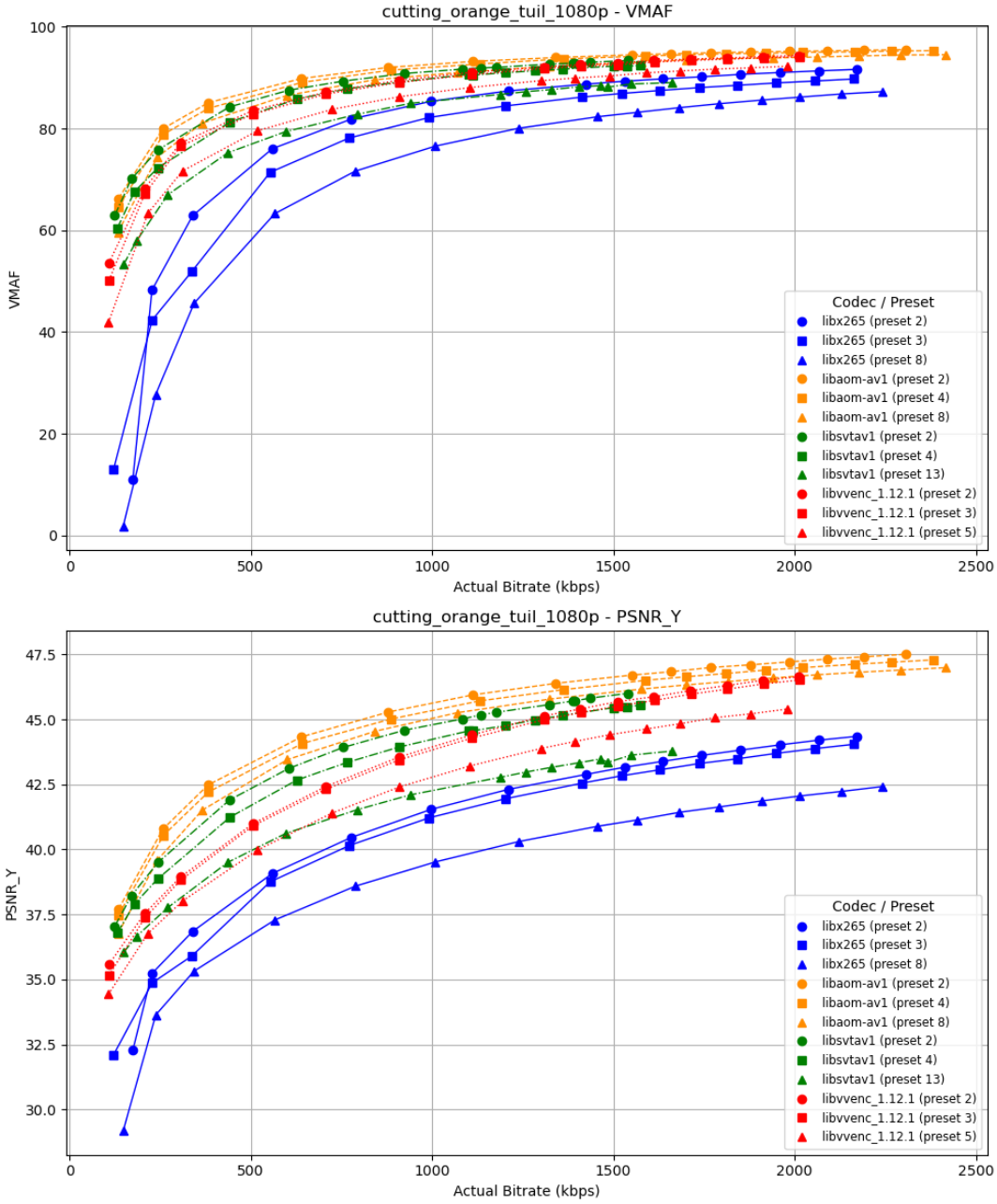

Figura 5.7: Grafici VMAF e PSNR per cutting_orange_tuil_1080p

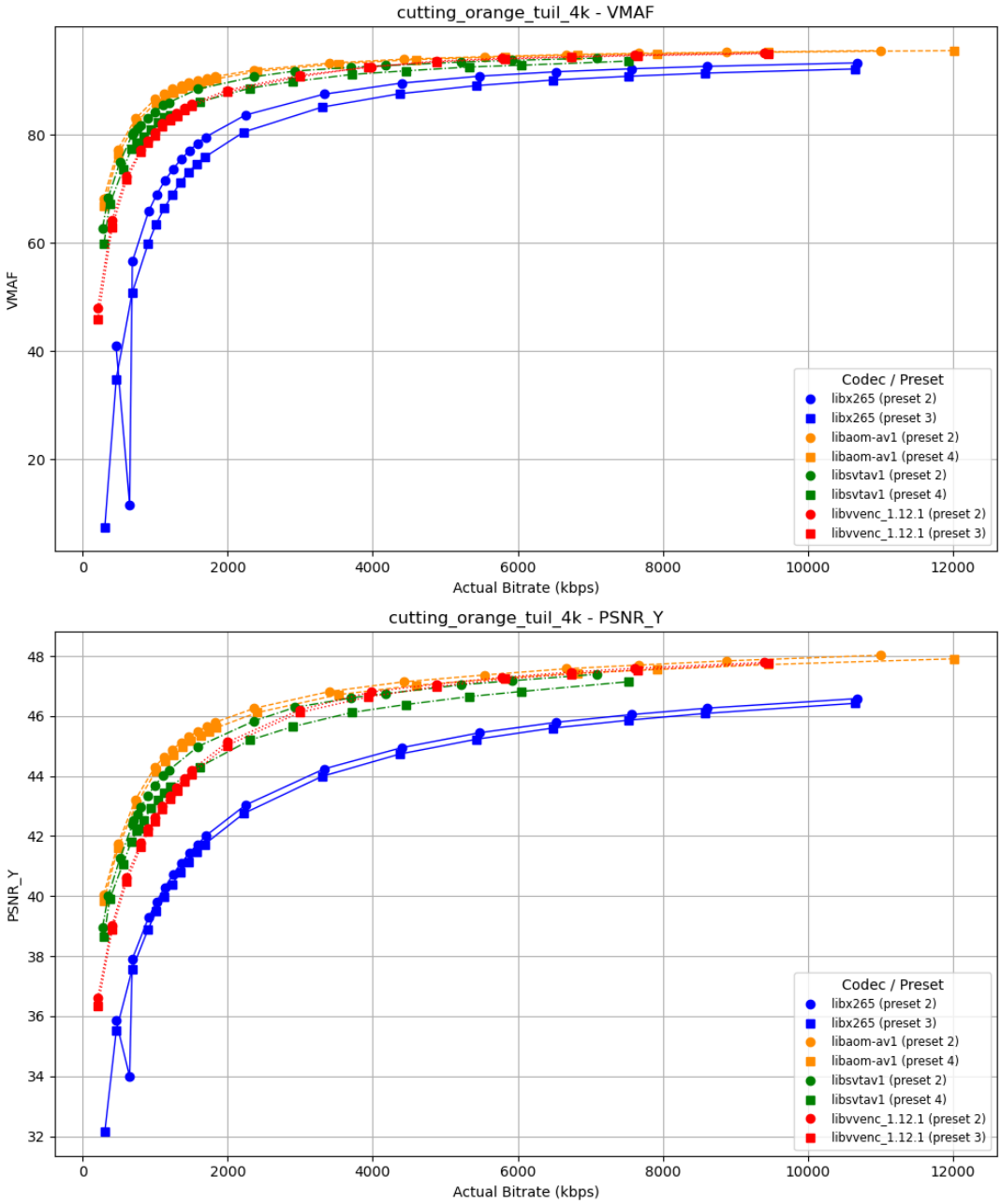

Figura 5.8: Grafici VMAF e PSNR per cutting_orange_tuil_4k

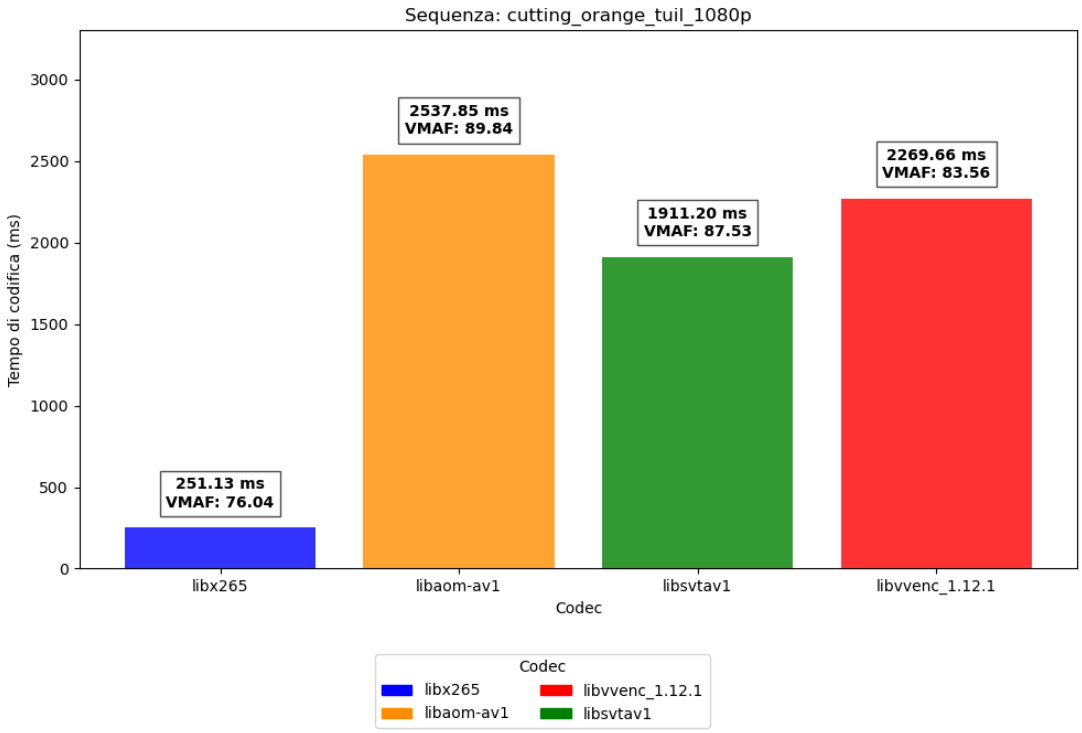

Figura 5.9: Grafico tempo di codifica per cutting_orange_tuil_1080p (600 KB/s)

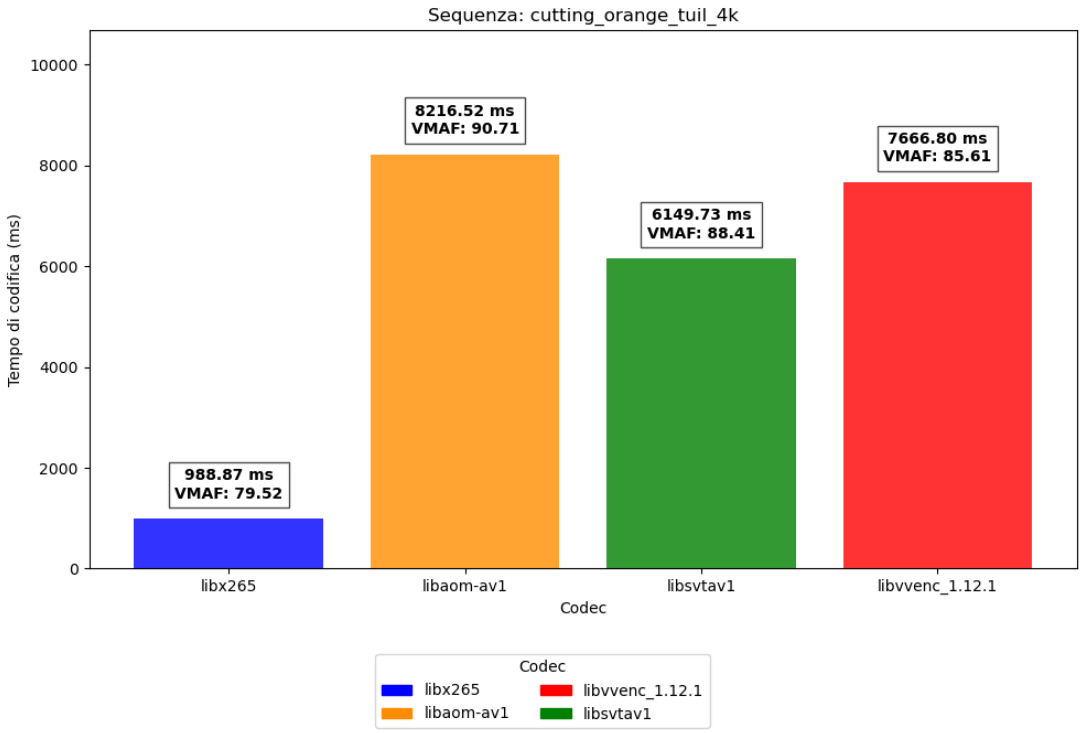

Figura 5.10: Grafico tempo di codifica per cutting_orange_tuil_4k (1600 KB/S)

5.1.5 Daydreamer_SDR_8s

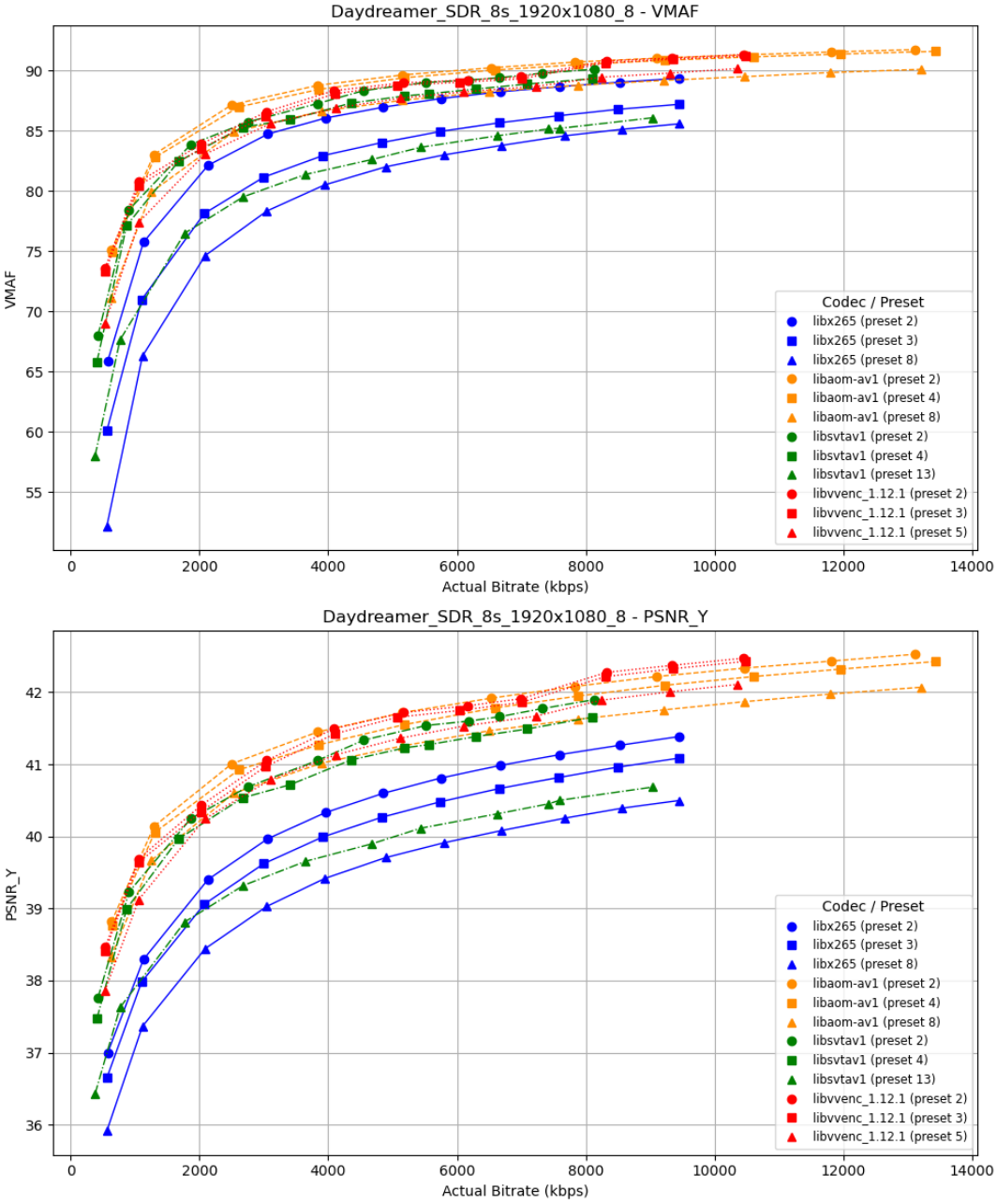

Figura 5.11: Grafici VMAF e PSNR per Daydreamer_SDR_8s_1920x1080_8

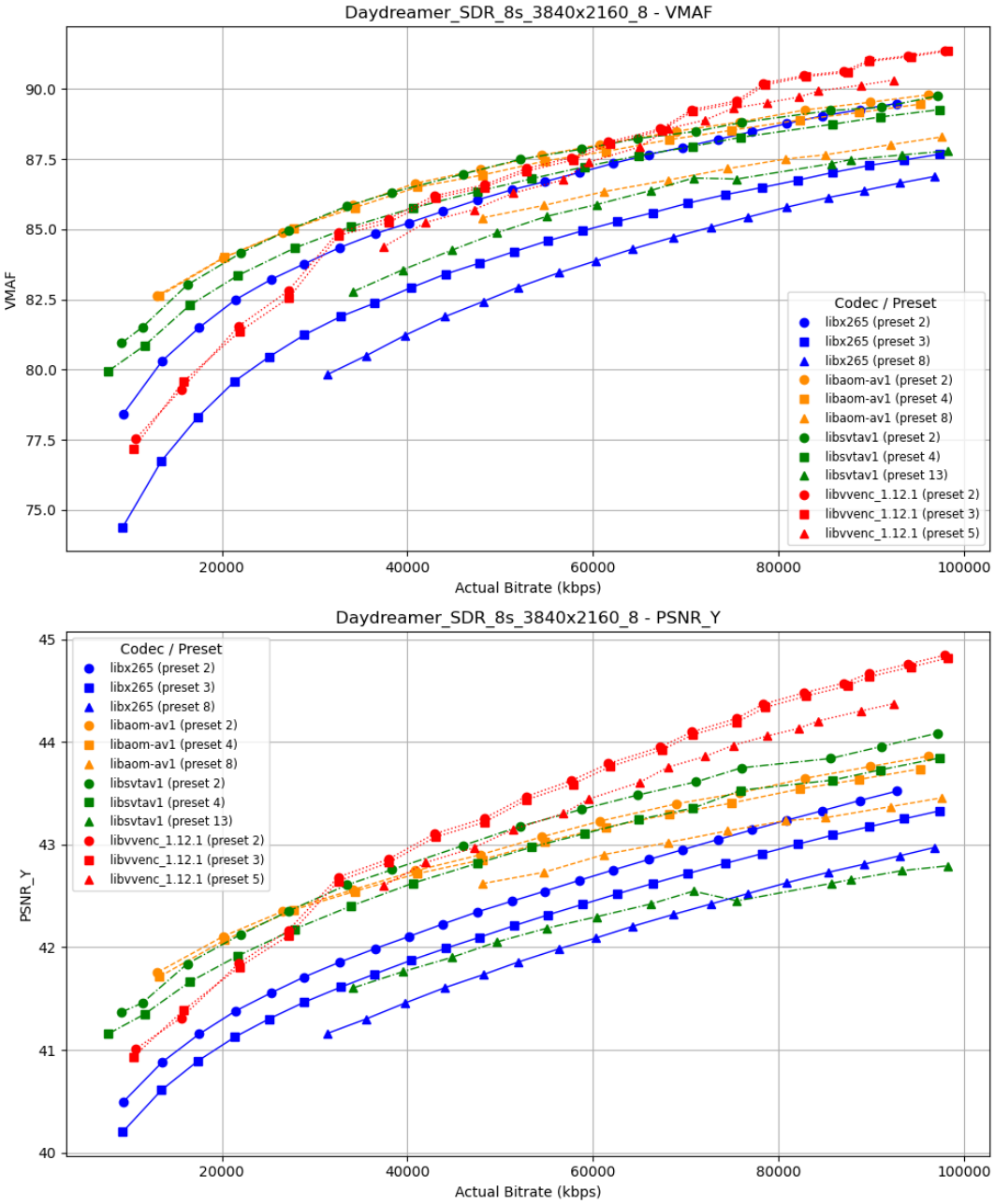

Figura 5.12: Grafici VMAF e PSNR per Daydreamer_SDR_8s_3840x2160_8

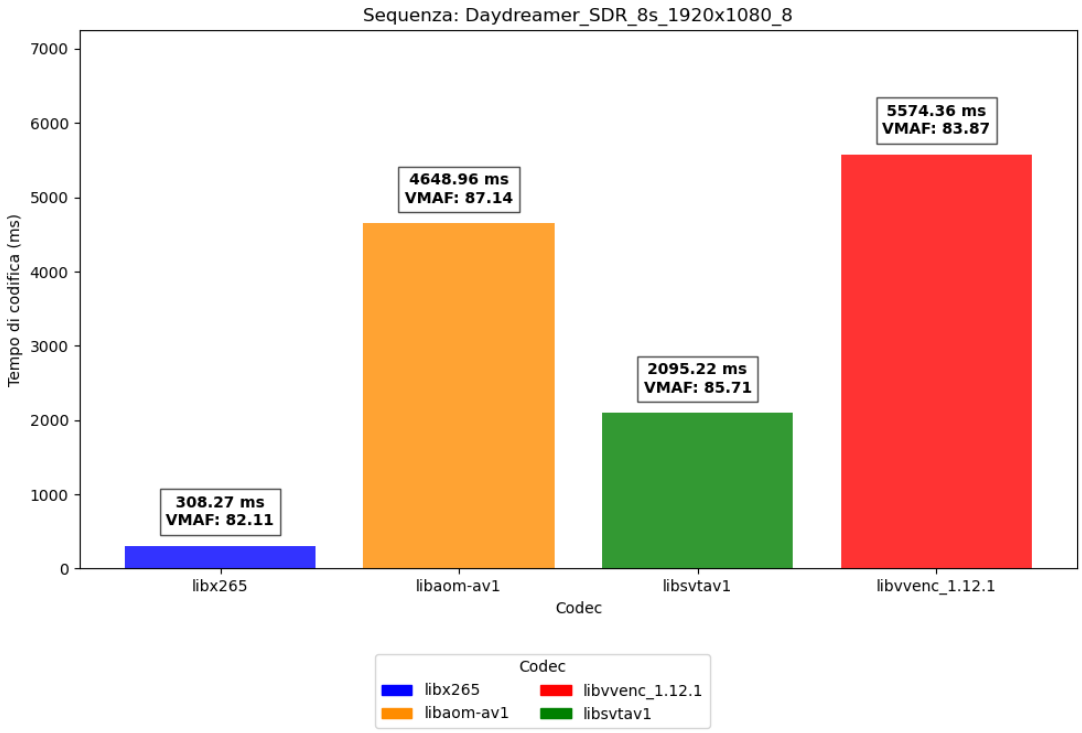

Figura 5.13: Grafico tempo di codifica per Daydreamer_SDR_8s_1920x1080_8 (2500 KB/s)

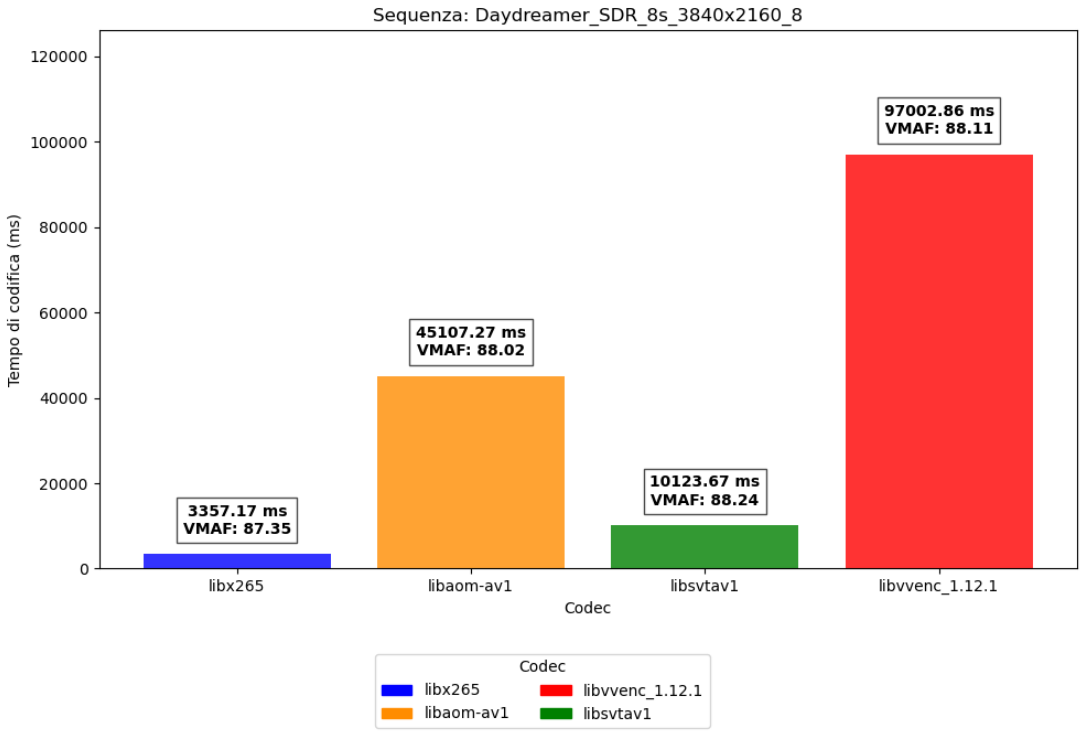

Figura 5.14: Grafico tempo di codifica per Daydreamer_SDR_8s_3840x2160_8 (60000 KB/s)

5.1.6 Sparks_cut_13

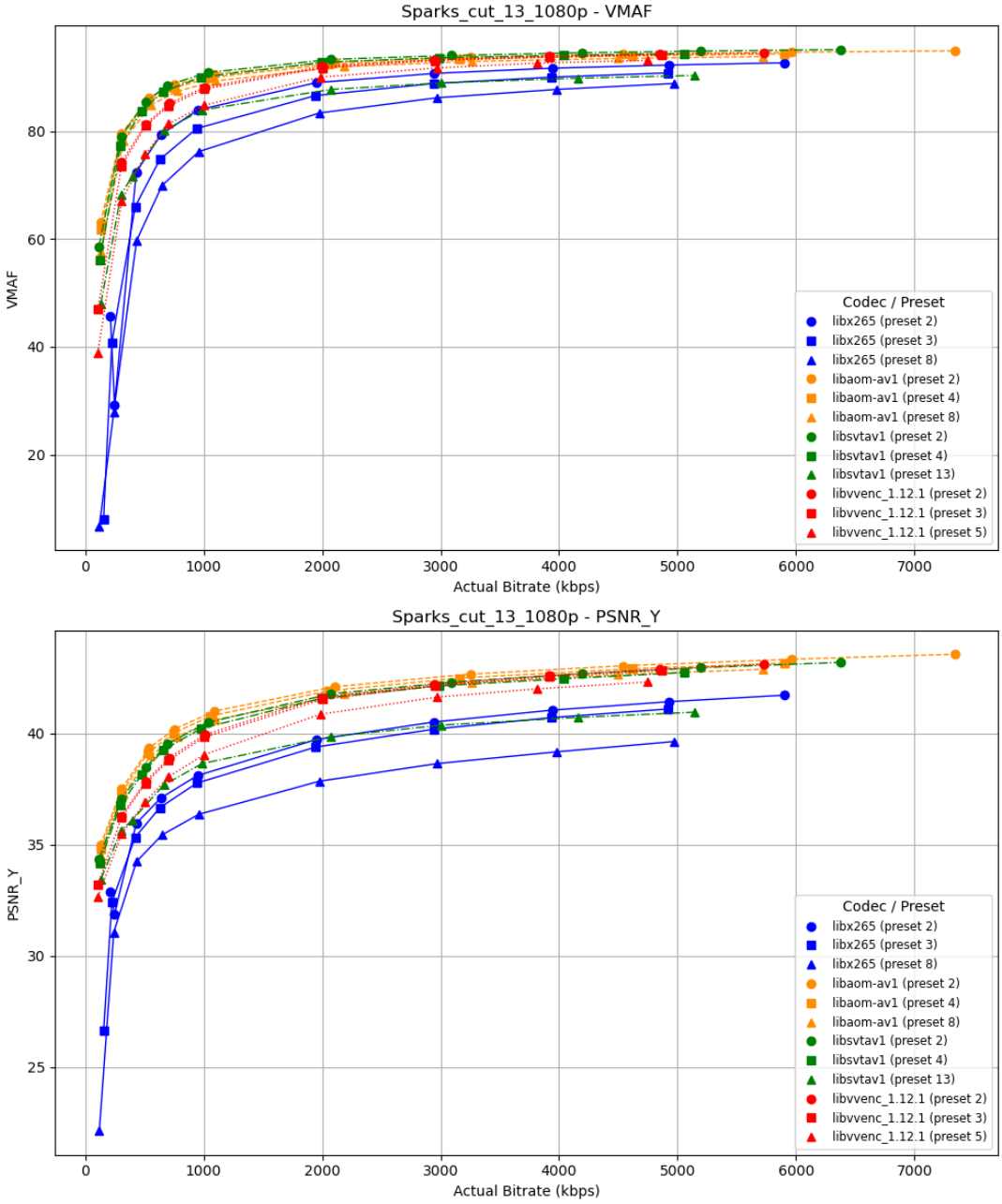

Figura 5.15: Grafici VMAF e PSNR per Sparks_cut_13_1080p

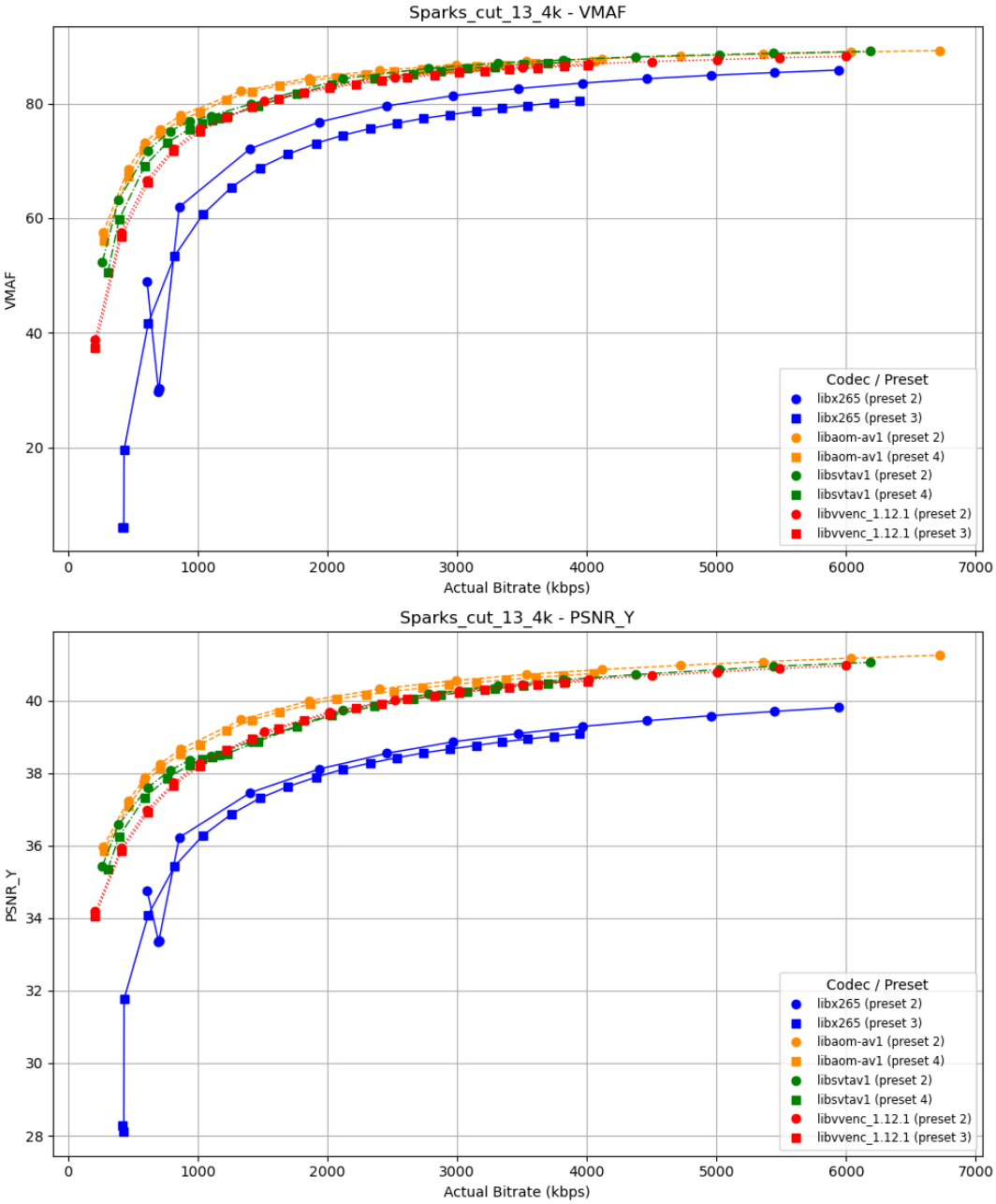

Figura 5.16: Grafici VMAF e PSNR per Sparks_cut_13_4k

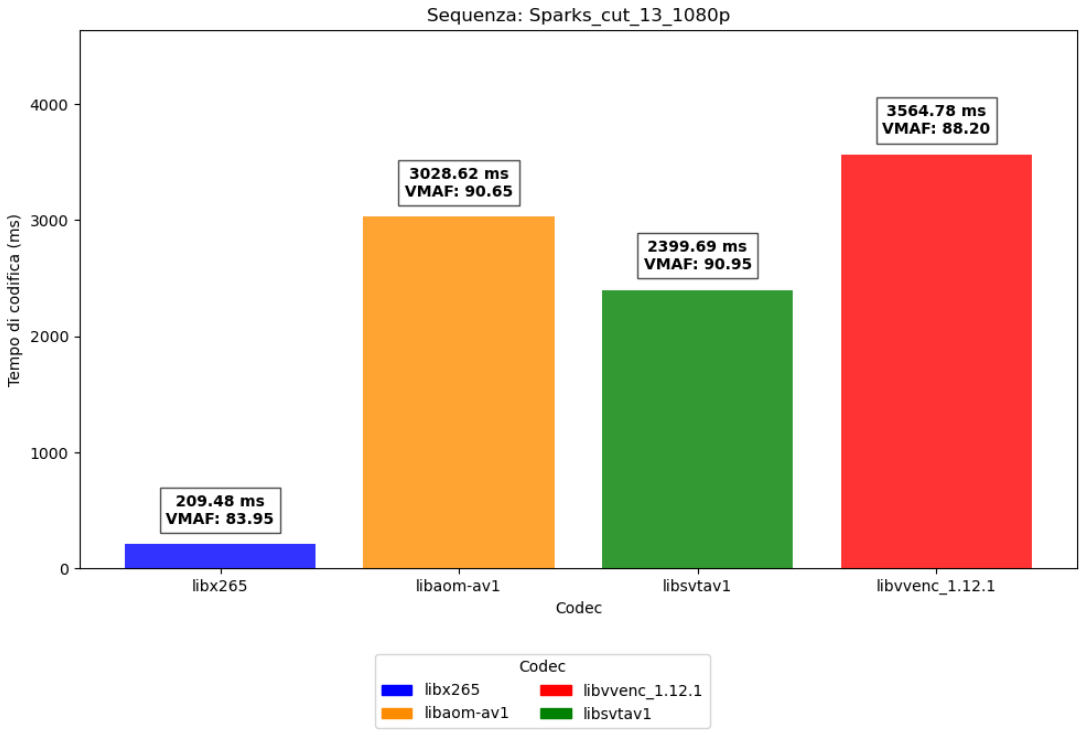

Figura 5.17: Grafico tempo di codifica per Sparks_cut_13_1080p (1000 KB/s)

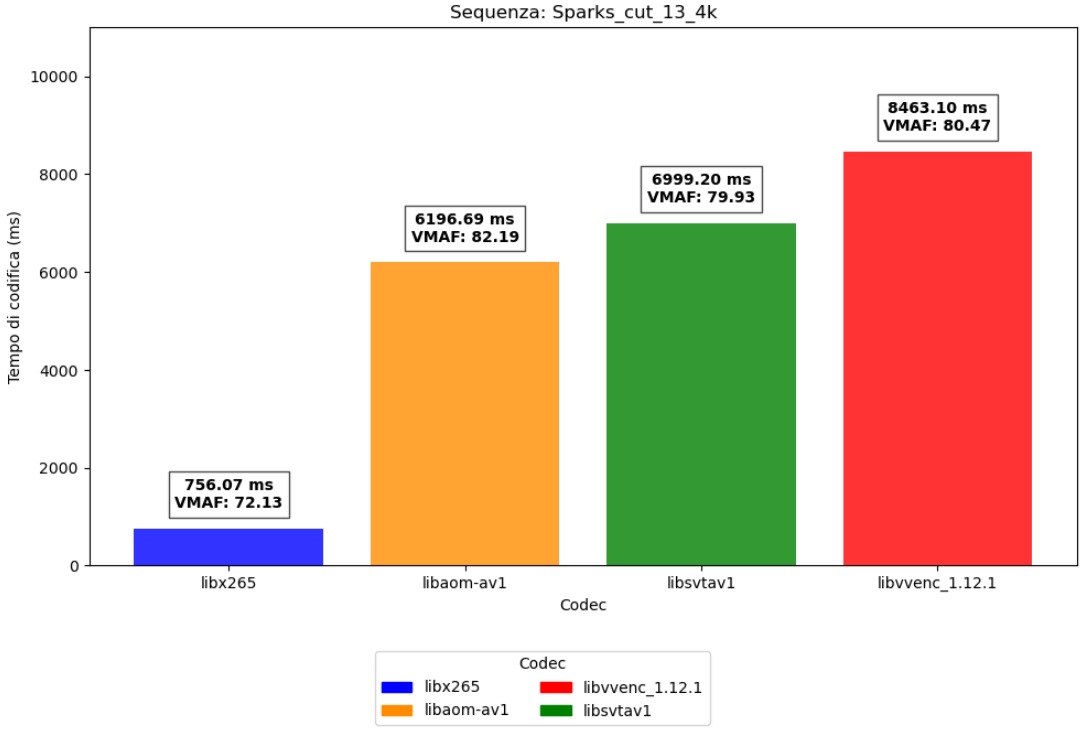

Figura 5.18: Grafico tempo di codifica per Sparks_cut_13_4k (1500 KB/s)

5.1.7 Sparks_cut_15

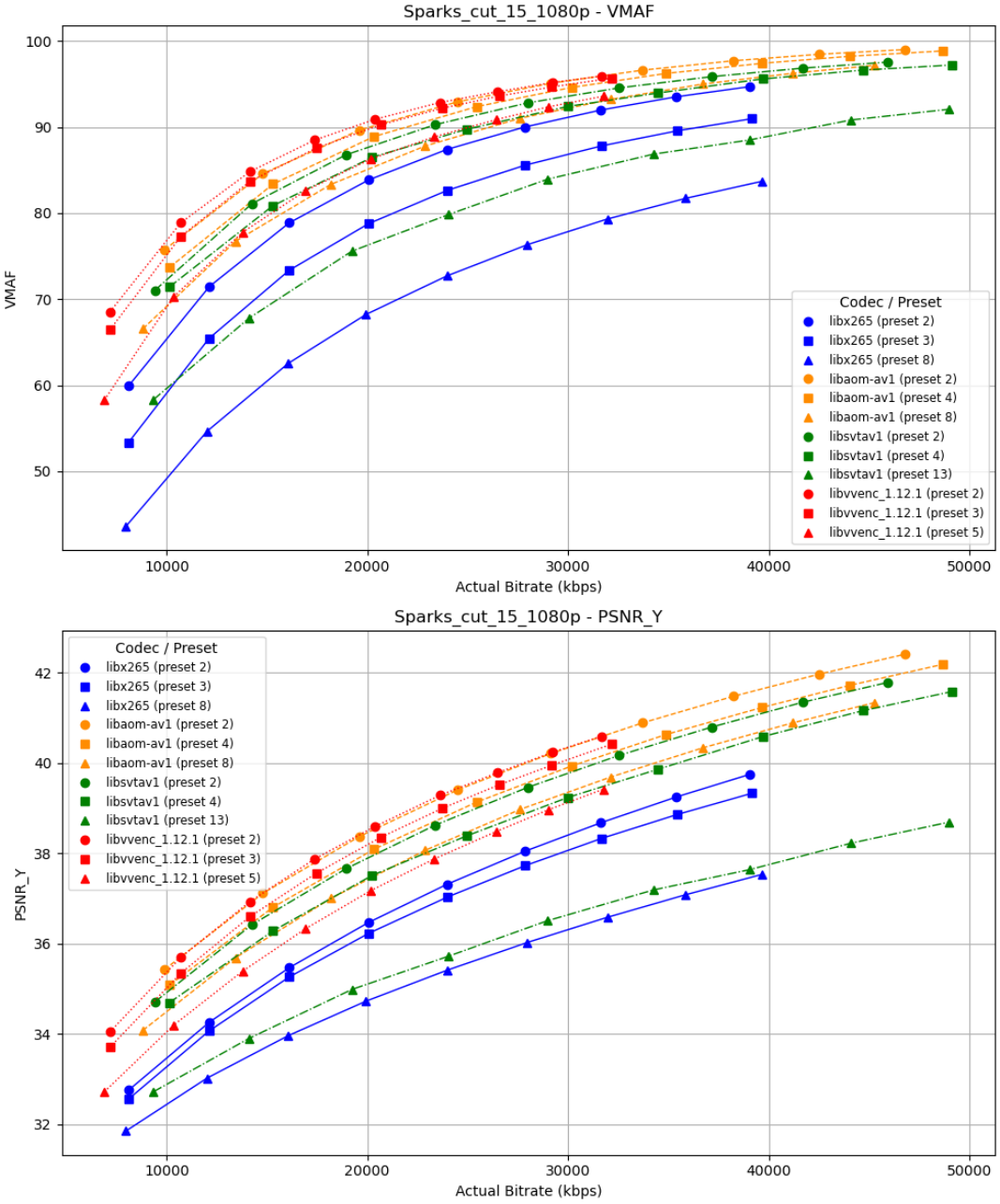

Figura 5.19: Grafici VMAF e PSNR per Sparks_cut_15_1080p

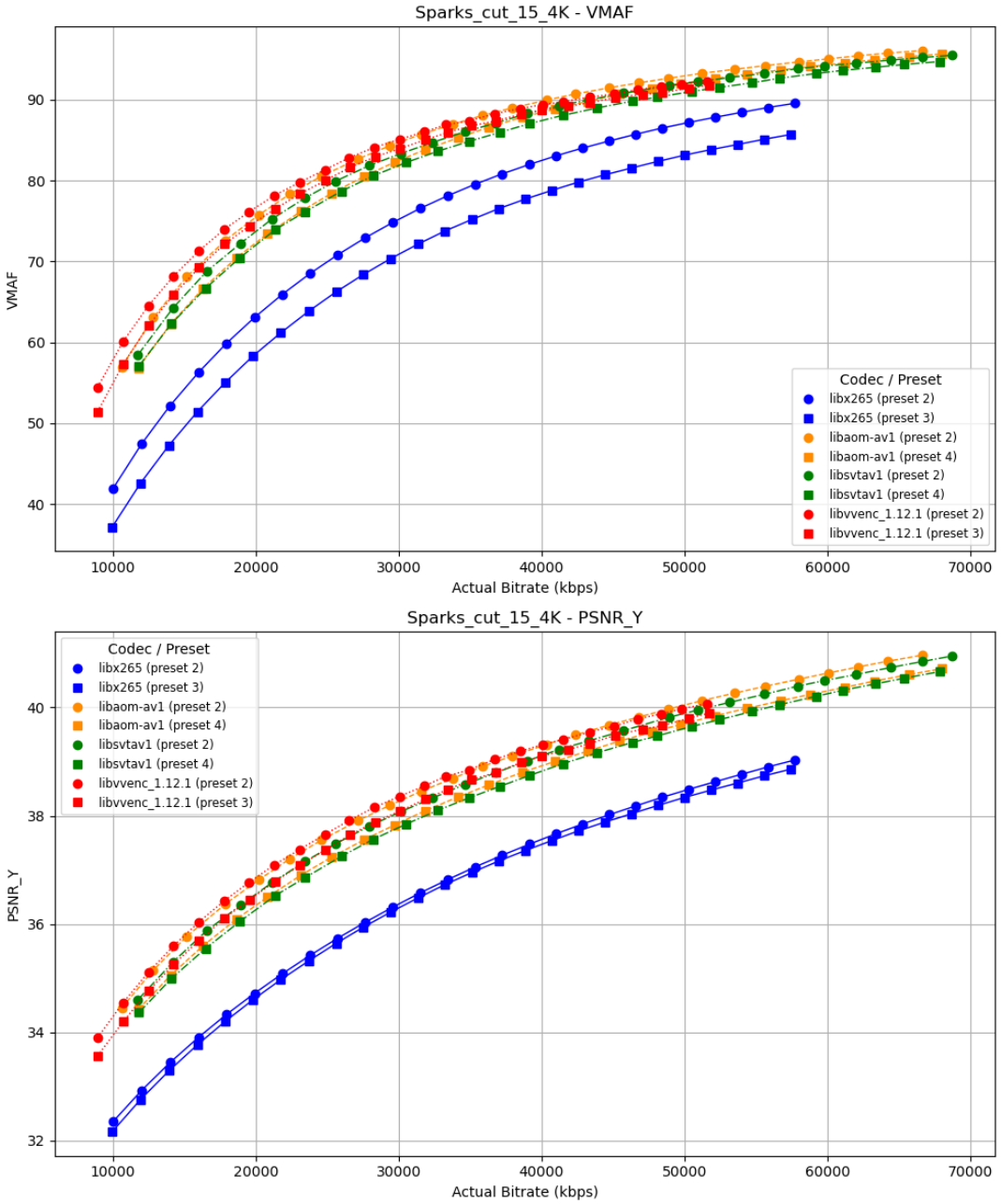

Figura 5.20: Grafici VMAF e PSNR per Sparks_cut_15_4K

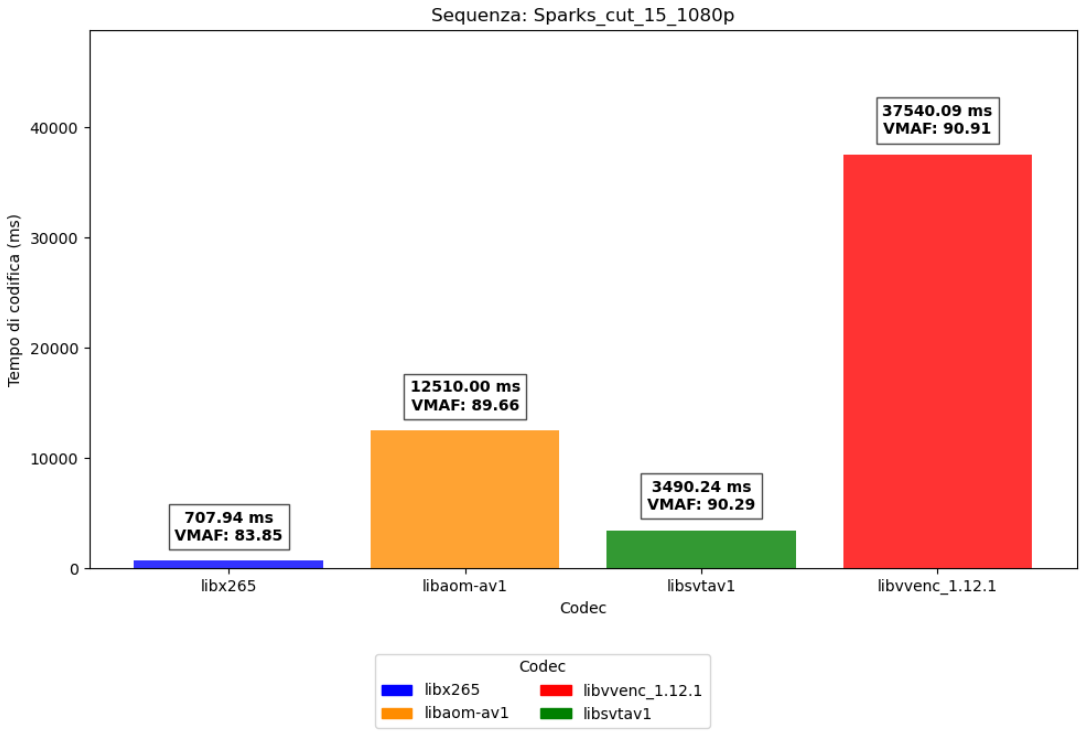

Figura 5.21: Grafico tempo di codifica per Sparks_cut_15_1080p (2000 KB/s)

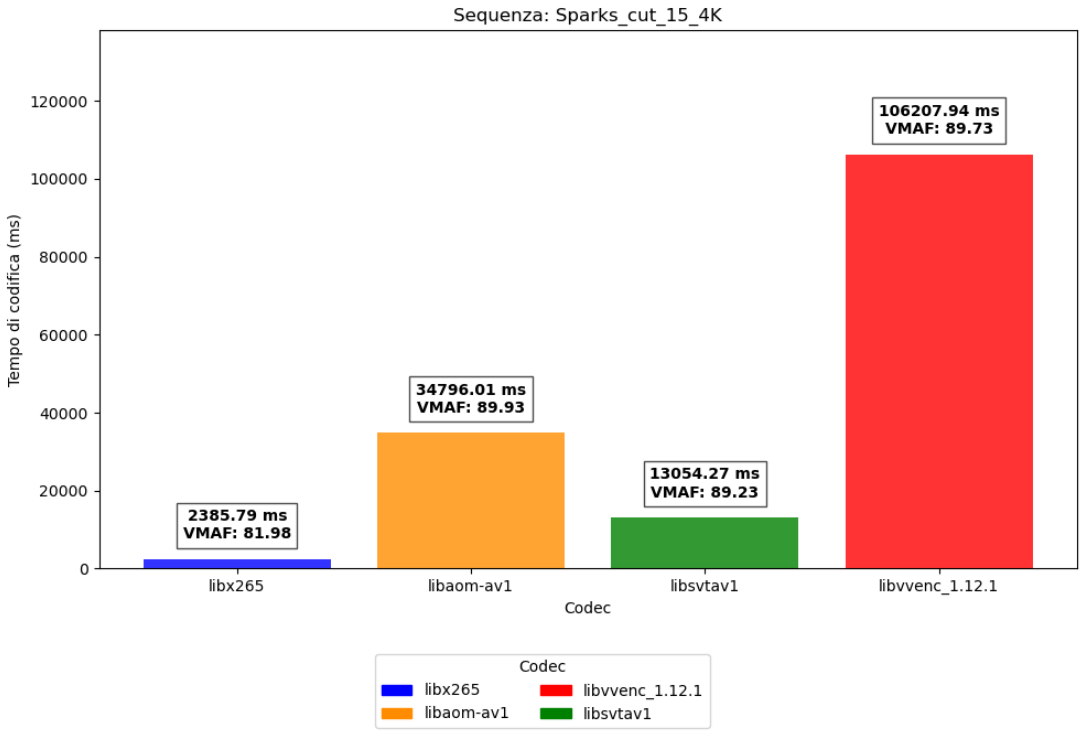

Figura 5.22: Grafico tempo di codifica per Sparks_cut_15_4K (40000 KB/s)

5.1.8 vegetables_tuil

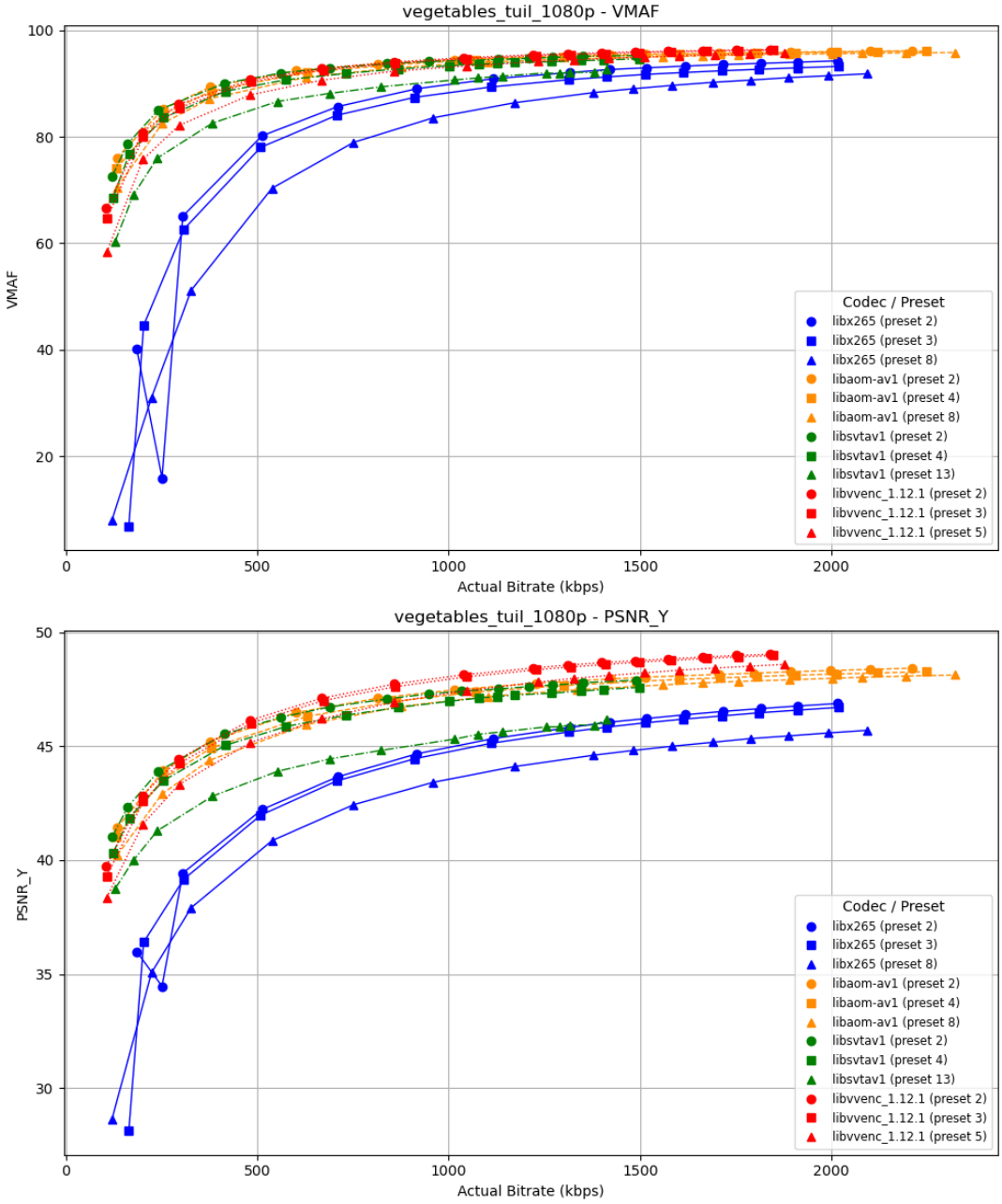

Figura 5.23: Grafici VMAF e PSNR per vegetables_tuil_1080p

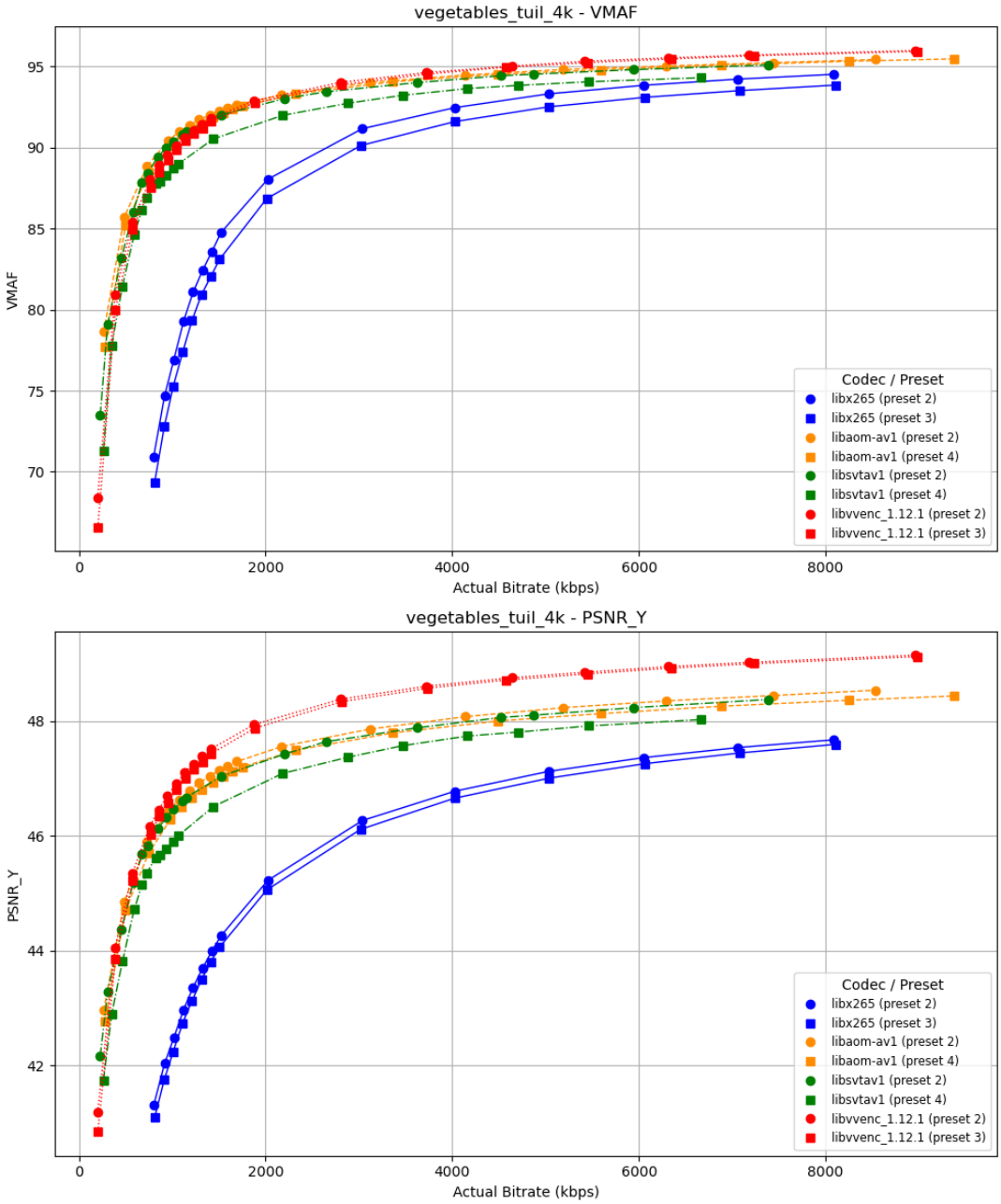

Figura 5.24: Grafici VMAF e PSNR per vegetables_tuil_4k

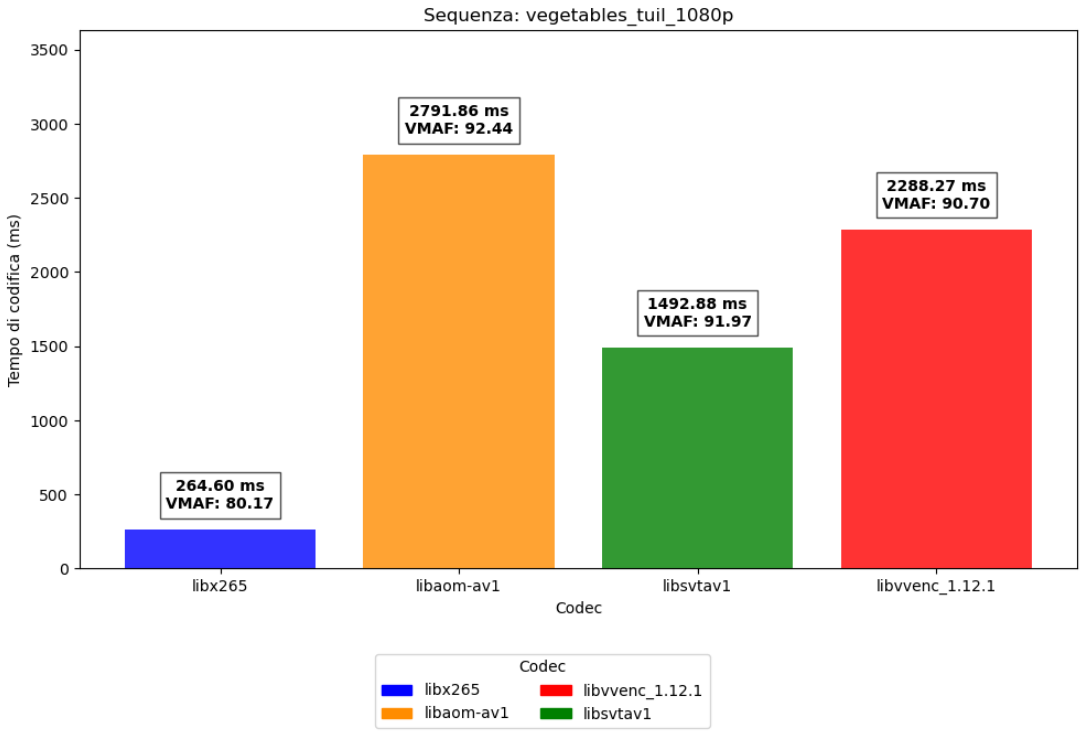

Figura 5.25: Grafico tempo di codifica per vegetables_tuil_1080p (500 KB/s)

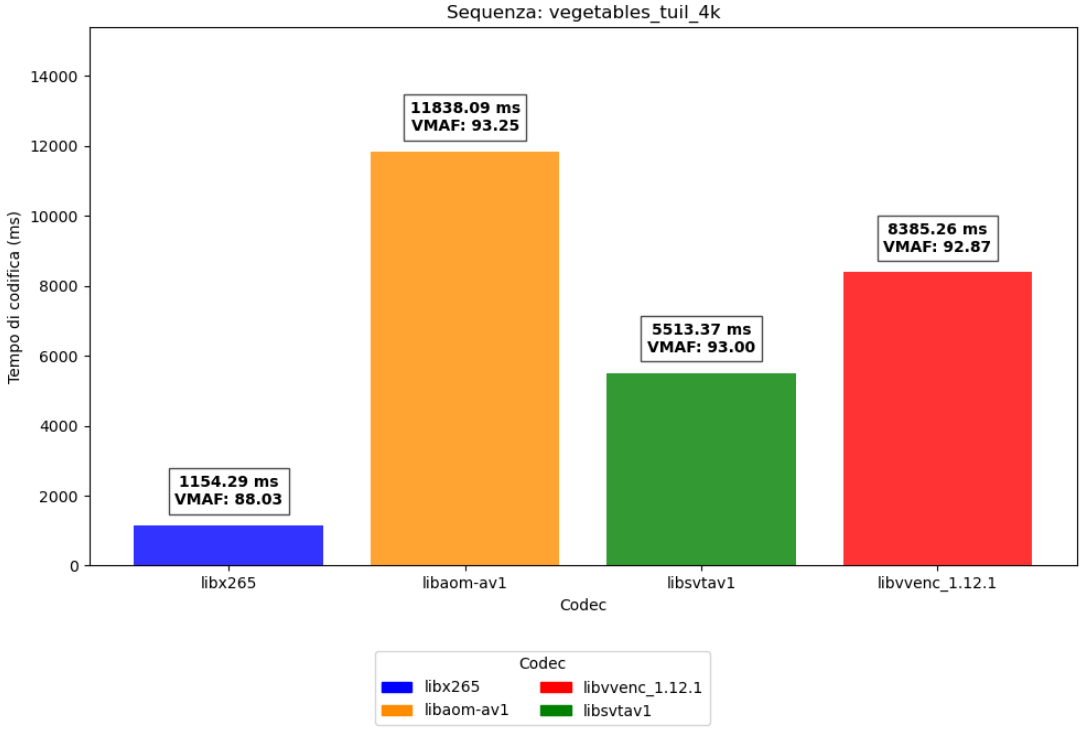

Figura 5.26: Grafico tempo di codifica per vegetables_tuil_4k (2000 KB/s)

5.1.9 water_netflix

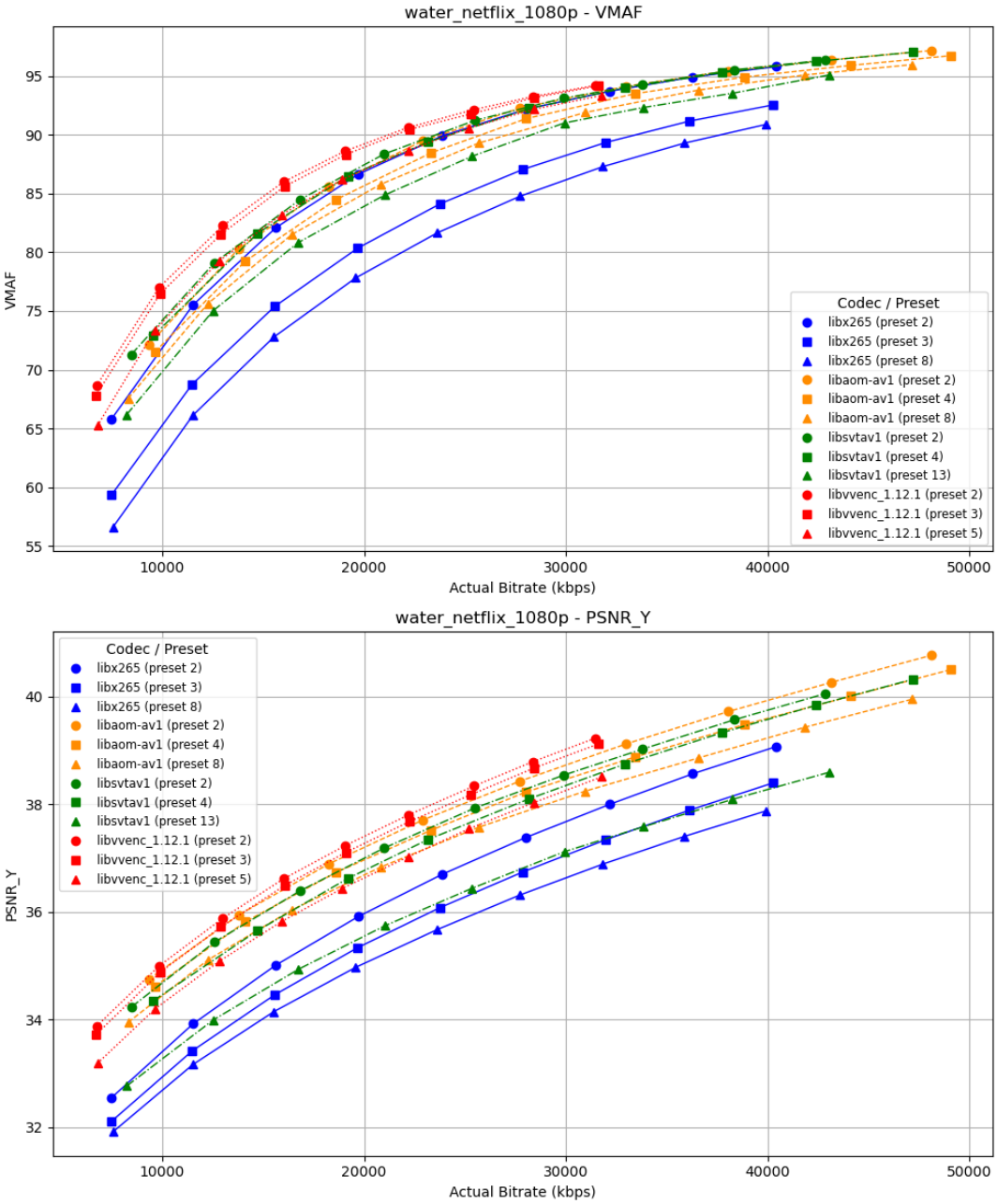

Figura 5.27: Grafici VMAF e PSNR per water_netflix_1080p

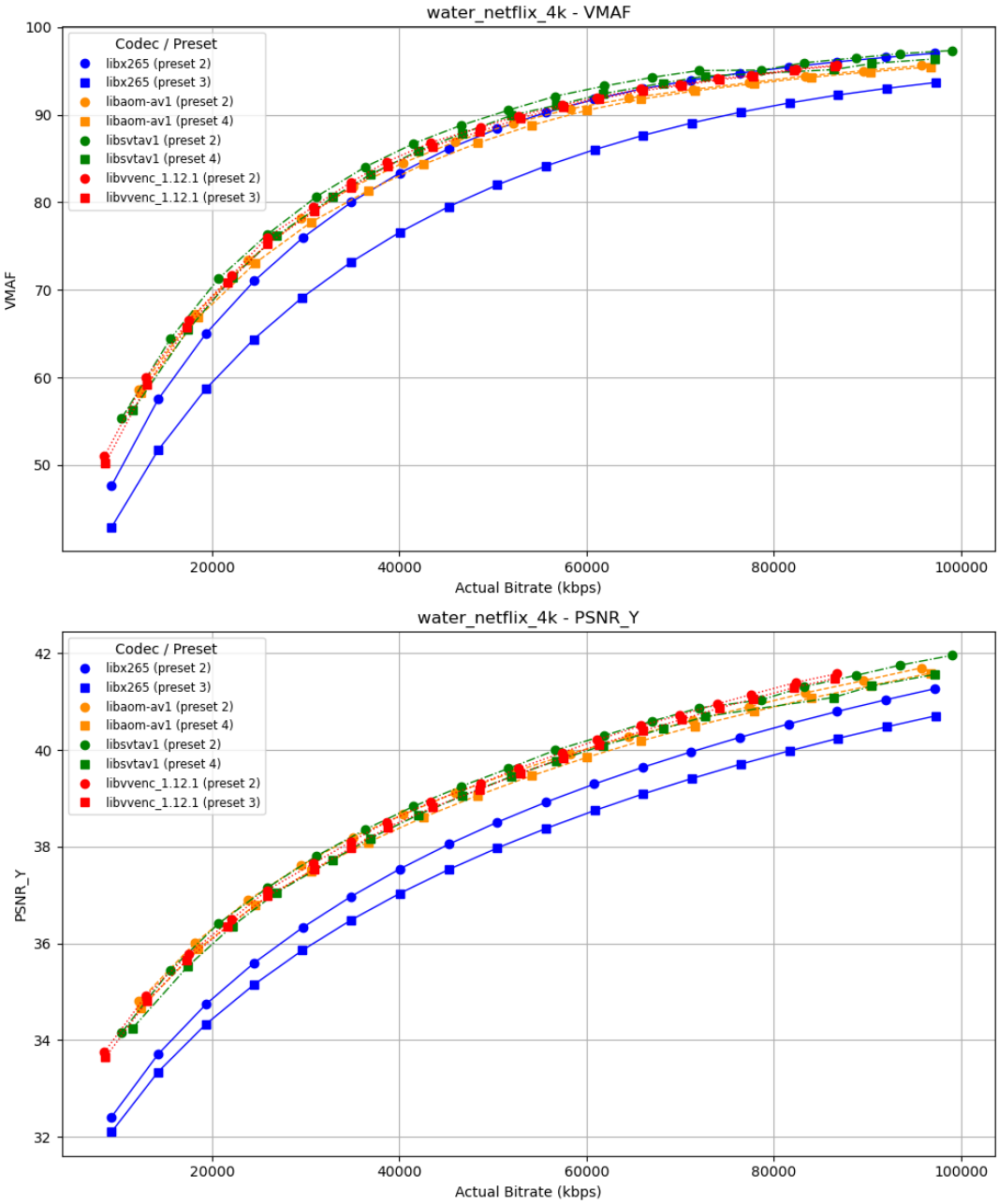

Figura 5.28: Grafici VMAF e PSNR per water_netflix_4k

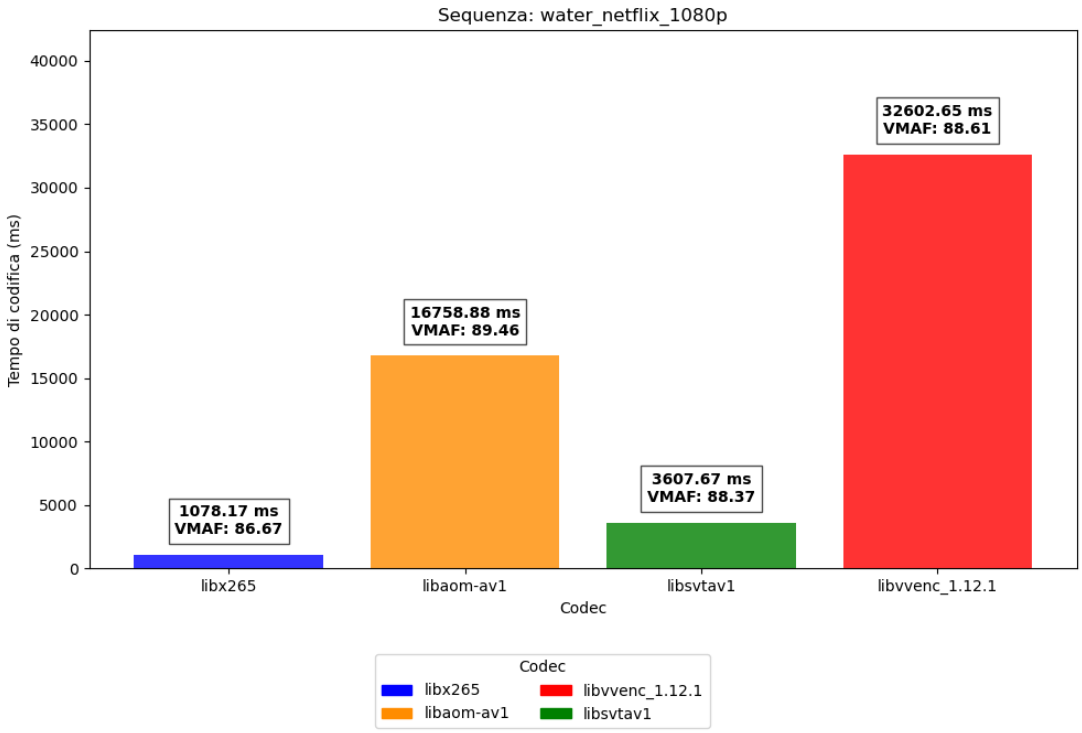

Figura 5.29: Grafico tempo di codifica per water_netflix_1080p (20000 KB/s)

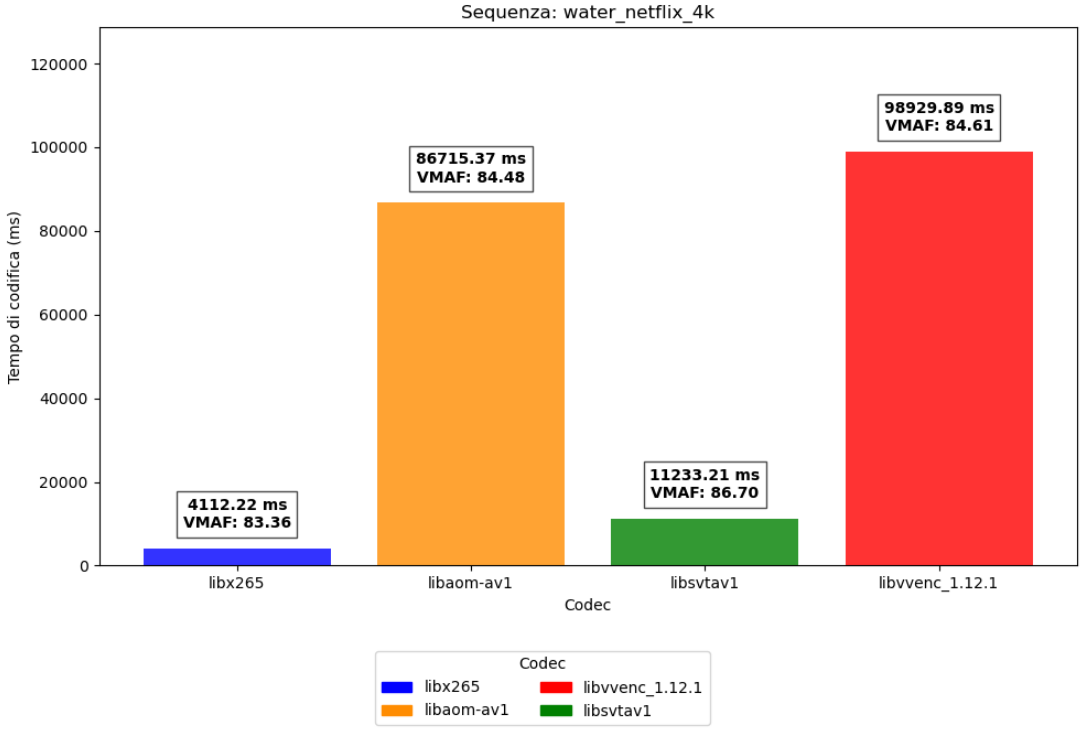

Figura 5.30: Grafico tempo di codifica per water_netflix_4k (40000 KB/s)

5.2 Analisi dei risultati

Osservando i grafici è possibile fare alcune considerazioni.
Alcune sequenze compresse mostrano un incremento notevole dei bit-rate rispetto alle altre, ad esempio le sequenze Sparks_cut_15 e water_netflix spiccano molto da questo punto di vista. Entrambe hanno in comune il fatto di avere una grande quantità di gocce d'acqua che si muovono a video, cosa che aumenta di molto la difficoltà di compressione.
Un'altra considerazione da fare è legata alle prestazioni dei vari codec. È evidente infatti come HEVC risulti essere, nella quasi totalità dei casi, il meno performante tra i codec analizzati, per quanto riguarda le metriche VMAF e PSNR.
VVenC, SVT-AV1 e AOM-AV1 tendono in generale a ottenere risultati abbastanza simili in media per VMAF e PSNR.
Per quanto riguarda i tempi di esecuzione, invece, le cose cambiano radicalmente. In questo caso, infatti, a spiccare per la lentezza nella codifica è VVenC che, nella maggior parte dei casi, a parità di bit-rate, mostra dei tempi di compressione nettamente superiori agli altri. È interessante notare come, in alcune sequenze, come ad esempio nella Sparks_cut_15, i tempi di codifica di VVenC siano 40-50 volte superiori a HEVC, a parità di bit-rate. La maggiore complessità computazionale di VVenC infatti si fa sentire nel momento in cui si deve comprimere sequenze con un alto livello di dettaglio, magari con una moltitudine di oggetti in movimento a schermo.
Di seguito, nella tabella 5.1, ho riportato i dati relativi al risparmio in termini di bit-rate, a parità di VMAF, rispetto ad HEVC, che ho preso come base per valutare le prestazioni degli altri codec e confrontarli.
Il risparmio è stato calcolato prendendo in considerazione solamente il preset "slow" per ogni codec (2 per AOM-AV1 e SVT-AV1), dove i codec sfruttano al massimo le loro potenzialità e dovrebbero dare i migliori risultati in termini di qualità visiva.

codec	**Risparmio bit-rate**
libaom-av1	38.28 %
libsvtav1	37.18 %
libvvenc	30.37 %

Tabella 5.1: Risparmio in termini di bit-rate rispetto ad HEVC

Oltre a guardare i dati è anche interessante fare un'analisi visiva di alcuni frame delle sequenze compresse, confrontantoli con la sequenza originale e tra di loro.
In seguito ho riportato un frame estratto dalle sequenze compresse di "cutting_orange_tuil" in versione 4K da cui è possibile apprezzare alcuni dettagli, confrontando il frame compresso con il frame originale.

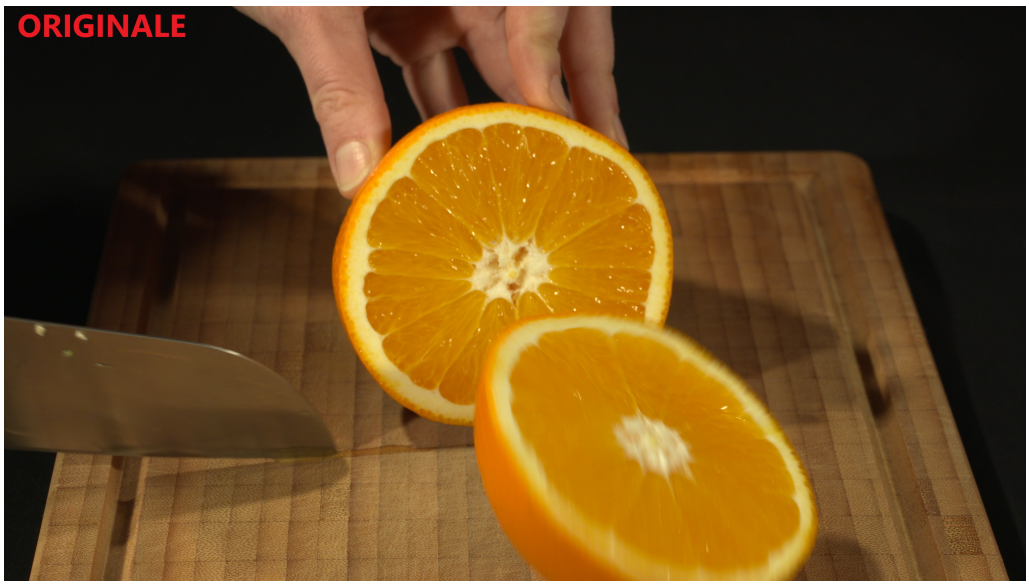

Figura 5.31: Frame originale della sequenza cutting_orange_tuil_4k

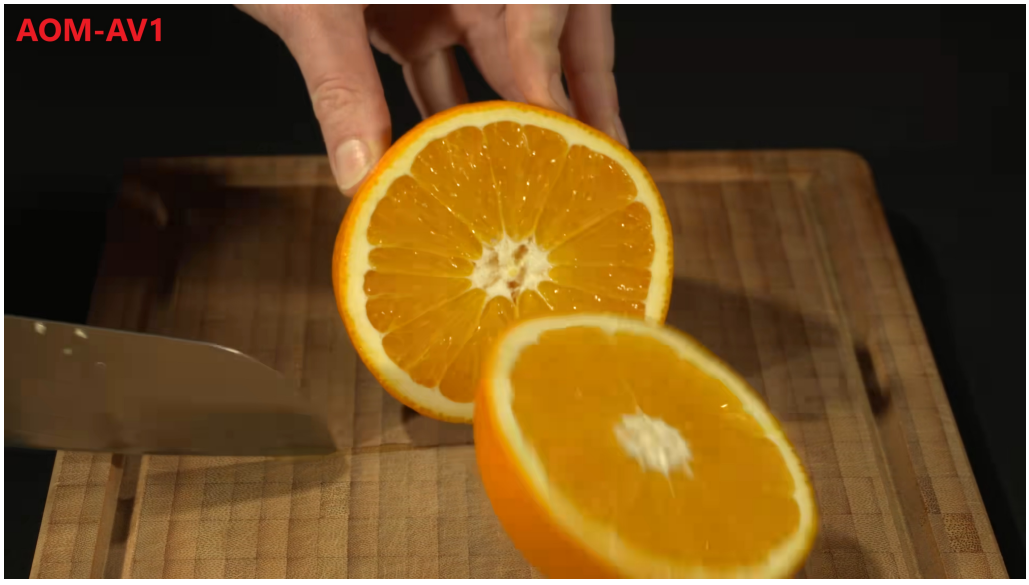

Figura 5.32: Frame della sequenza cutting_orange_tuil_4k compresso con AOM-AV1

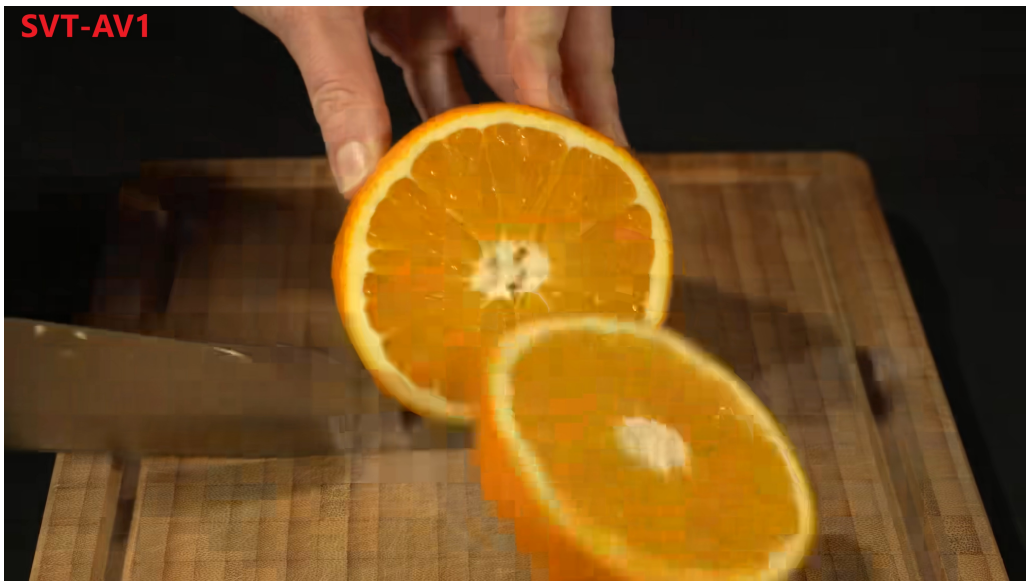

Figura 5.33: Frame della sequenza cutting_orange_tuil_4k compresso con SVT-AV1

Osservando le due figure 5.32 e 5.33 è facile notare come ci sia una sostanziale differenza di qualità tra i due, con AOM-AV1 che, a parità di bit-rate (400 kbit/s in questo caso) offre una qualità decisamente superiore ad AOM-SVT. La differenza è molto visibile nelle parti in movimento, ovvero nel coltello e nelle due fette di arancia, dove è presente una "squadrettatura" evidente dell'immagine nella versione SVT-AV1. È facile notare come anche le ombre presentino delle differenze sostanziali: quelle di AOM-AV1 sono infatti molto più morbide e definite.
La versione AOM-AV1 presenta un qualità molto alta, con i dettagli della fetta inferiore dell'arancia ben distinguibili e una "bloccosità" molto limitata.

Figura 5.34: Frame della sequenza cutting_orange_tuil_4k compresso con VVenC

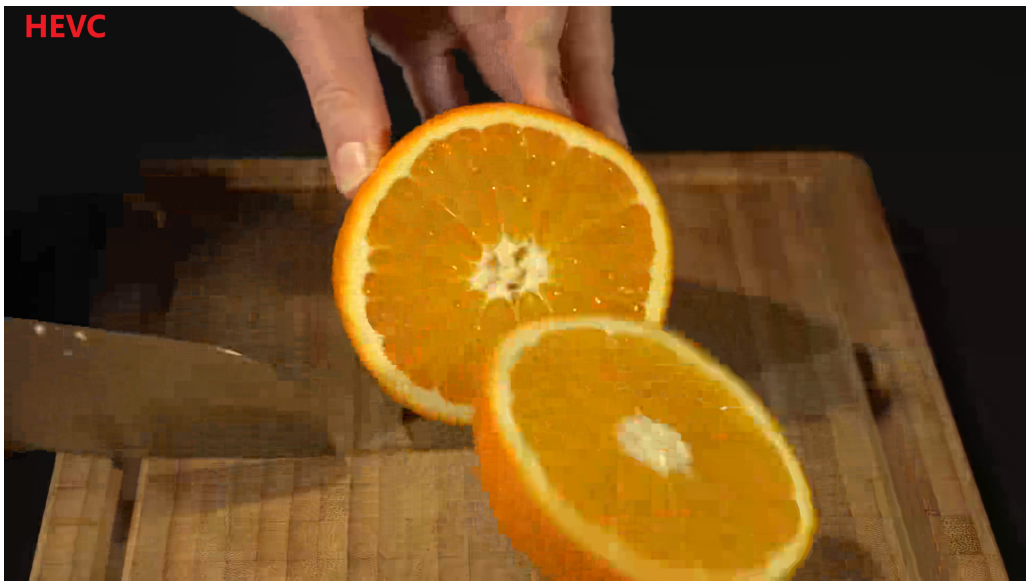

Figura 5.35: Frame della sequenza cutting_orange_tuil_4k compresso con HEVC

Da queste immagini è possibile notare come VVenC presenti una qualità discreta, che però perde molto dettaglio nelle parti in movimento. In questo caso VVenC, rispetto a HEVC e SVT-AV1, gestisce meglio le zone di ombra.
Il codec che restituisce la qualità complessiva migliore, molto simile all'originale, è AOM-AV1. Questo risultato è in accordo con i grafici di VMAF e PSNR, dimostrando così che le metriche scelte sono buoni indicatori della qualità percettiva delle immagini.
Un altro frame interessante da analizzare è relativo alla sequenza "bigbuck_bunny_8bit" in versione 4k.

Figura 5.36: Frame originale della sequenza bigbuck_bunny_8bit_4k

Le differenze di qualità si notano maggiormente nella parte di immagine dove è rappresentato lo scoiattolo e la farfalla. Anche in questo caso, come nell'immagine relativa alla sequenza analizzata in precedenza, le parti con maggiore movimento sono quelle che danno

più problemi in fase di compressione.

Figura 5.37: Frame della sequenza bigbuck_bunny_8bit compresso con AOM-AV1

Figura 5.38: Frame della sequenza bigbuck_bunny_8bit compresso con SVT-AV1

Figura 5.39: Frame della sequenza bigbuck_bunny_8bit compresso con VVC

Tra i due frame relativi a SVT-AV1 e VVenC si può notare come la versione VVenC abbia una qualità complessiva molto simile a SVT-AV1; a parte la zona dello scoiattolo e della farfalla, in cui si nota una certa perdita di dettaglio in VVenC.
È anche possibile notare il lavoro del filtro di deblocking di VVenC che cerca di eliminare la "bloccosità" di quella zona. Tutto sommato per quanto riguarda lo sfondo più statico e il pelo del coniglio, VVenC si comporta molto bene, praticamente eguagliando i codec AV1 come qualità dei dettagli.

Figura 5.40: Frame della sequenza bigbuck_bunny_8bit compresso con HEVC

Anche in questo caso, tra tutte le versioni del frame, la versione peggiore è data da HEVC, in cui si nota una notevole presenza di "bloccosità" dell'immagine concentrata soprattutto nel pelo del coniglio, nello scoiattolo e nella farfalla.

Capitolo 6

Conclusioni

L'analisi che ho condotto in questa tesi mi ha permesso di confrontare, in termini di velocità di codifica e qualità, le prestazioni dei codificatori AV1 e VVC rispetto a HEVC, utilizzando un sistema HPC per ottimizzare l'esecuzione degli esperimenti e la raccolta dei risultati. Utilizzando le metriche oggettive (VMAF e PSNR) per l'analisi della qualità delle sequenze compresse ho potuto trarre diverse considerazioni riguardo le prestazioni di ciascun codificatore.
I risultati mostrano come AV1 e VVC offrano un migliore rapporto qualità-compressione rispetto a HEVC, permettendo di ottenere un notevole risparmio in termini di bit-rate, a parità di qualità di compressione. Complessivamente, il codificatore migliore è stato AV1, sia nella versione AOM-AV1 che SVT-AV1. AV1 ha fornito delle prestazioni superiori sia in termini di qualità percepita che di tempi di codifica rispetto agli altri codificatori.
VVC, nell'implementazione VVenC, nella maggior parte dei casi ha mostrato delle prestazioni leggermente inferiori rispetto a AV1, in termini di qualità video. Il suo punto debole è stato il tempo di esecuzione delle codifiche, nettamente superiore sia a AV1 che HEVC.
Tempi di codifica elevati potrebbero rappresentare una limitazione notevole in contesti che richiedono encoding in tempo reale, come ad esempio nel cloud gaming, dove la bassa latenza gioca un ruolo fondamentale per garantire un'esperienza fluida e responsiva all'utente. Da questo punto di vista, le attuali implementazioni di VVC, come VVenC, vanno ancora perfezionate.
Un'altra osservazione importante riguarda la "difficoltà" di codifica di alcune sequenze video rispetto ad altre. Sequenze con scene ad elevata complessità visiva, come Sparks_cut_15 e water_netflix hanno richiesto un aumento significativo del bit-rate necessario ad ottenere una qualità visiva soddisfacente. Questo suggerisce che nonostante AV1 e VVC siano superiori in termini di qualità, a parità di bit-rate, rispetto a HEVC, le loro prestazioni variano molto in relazione alla tipologia di sequenza video da comprimere.
Infine, l'uso dell'HPC si è rivelato fondamentale per riuscire ad eseguire gli esperimenti in modo efficiente, permettendomi di ridurre al minimo possibile i tempi di esecuzione delle codifiche. Se avessi utilizzato altri sistemi, come personal computer o workstation, per lavorare su un numero elevato di codifiche in risoluzione FULL-HD o superiore, i tempi di esecuzione sarebbero aumentati notevolmente, rendendo difficoltoso l'andamento del lavoro e la correzione di eventuali errori. La possibilità di parallelizzare le operazioni ha reso molto agevole l'analisi dei dati e ha anche reso possibile la riesecuzione di alcuni esperimenti, senza intaccare o rallentare il lavoro successivo.
In conclusione, i codificatori moderni come AV1 o VVC, nonostante rappresentino un passo

in avanti rispetto a HEVC, soprattutto in termini di efficienza di compressione, presentano ancora alcune problematiche che possono precluderne l'attuale utilizzo su larga scala, specialmente in contesti in cui la rapidità di elaborazione è un fattore critico.
Per adesso, le attuali implementazioni di VVC e AV1 che ho analizzato, si sono dimostrate particolarmente indicate per essere utilizzate in scenari dove la compressione efficiente è l'obiettivo principale, come ad esempio nella memorizzazione e trasmissione di contenuti on-demand su piattaforme come Netflix o YouTube. La capacità di questi codificatori di garantire una qualità video elevata riducendo molto il bit-rate li rende ideali in scenari del genere, dove i video vengono codificati una sola volta, magari a diverse risoluzioni, e riprodotti molteplici volte, ottimizzando così l'uso della banda e dello spazio di archiviazione.

Appendice A

Script di supporto per la codifica e la raccolta dei risultati

A.1 Encoding.sh

```
1  #!/bin/bash
2
3  file_config="$2"
4  file_config_name_with_ext=$(basename "${file_config}")
5  file_config_name_no_ext="${file_config_name_with_ext%.*}"
6  file_config_extension="${file_config_name_with_ext##*.}"
7
8  if [ ! -f "${file_config}" ]; then
9    echo "File di configurazione non trovato."
10   exit 1
11 fi
12
13 if [ "${file_config_extension}" != "cfg" ]; then
14   echo "Errore: estensione file di configurazione non corretta."
15   exit 1
16 fi
17
18 . ./${file_config}
19
20 file_name=$1
21 file_name_ext=$(basename "$file_name")
22 file_name_no_ext="${file_name_ext%.*}"
23 width=$3
24 height=$4
25 fps=$5
26 vvc_preset="$VVC_PRESET"
27 vvc_enc_mode=$VVC_ENCODING_MODE
28
29 for arg in "$@"; do
```

```
30   case "$arg" in
31     bitrate=*)
32       bit_rate_param="${arg}"
33       BIT_RATES=${arg#*=}
34       ;;
35   esac
36 done
37
38 bit_rates_=$(echo "$BIT_RATES" | tr ' ' '_')
39
40 path_to_results=\
41 "./results_${file_name_no_ext}_${file_config_name_no_ext}_${bit_rates_}/"
42
43
44 if [ "${ENCODE}" = "on" ]; then
45   rm -rf ${path_to_results}
46   mkdir -m 755 -p ${path_to_results}
47   cp ${file_config} ${path_to_results}
48 fi
49
50 for codec in $CODECS; do
51     echo "Processing file: ${file_name_ext} (codec: $codec)"
52     sh ./encoding_scripts/encoding_bit_rate.sh $file_name $file_config \
53     $codec $width $height $fps $bit_rate_param
54     echo "Encoding finished ($codec)."
55 done
56
57
58 if [ "${EVALUATE}" = "on" ]; then
59   chmod +x ./vmaf
60   sh ./vmaf_scripts/vmaf_bit_rate.sh $file_name $file_config $codec $width \
61   $height $fps $bit_rate_param
62 fi
```

A.2 Encoding_bit_rate.sh

```
1  #!/bin/bash
2
3  file_config="$2"
4  file_config_name_with_ext=$(basename "${file_config}")
5  file_config_name_no_ext="${file_config_name_with_ext%.*}"
6  file_config_extension="${file_config_name_with_ext##*.}"
7  . ./${file_config}
8
9  file_name=$1
10 file_name_ext=$(basename "$file_name")
11 file_name_no_ext="${file_name_ext%.*}"
12 codec=$3
13 codecs_=$(echo "$CODECS" | tr ' ' '_')
14 file_extension="${file_name##*.}"
```

```
15 width=$4
16 height=$5
17 fps=$6
18 pix_fmt="-pix_fmt ${PIX_FMT_FOR_ENC_DEC}"
19 s="-s ${width}x${height}"
20 r="-r "$fps""
21 hevc_preset="$HEVC_PRESET"
22 vvc_preset="$VVC_PRESET"
23 av1_svt_preset="$AV1_SVT_PRESET"
24 av1_preset="$AV1_PRESET"
25 vvc_enc_mode=$VVC_ENCODING_MODE
26 ffmpeg_vvc_prmts=$FFMPEG_VVC_PARAMETERS
27 vvenc_prmts=$VVENC_PARAMETERS
28 ffmpeg_hevc_prmts=$FFMPEG_HEVC_PARAMETERS
29 hevc_prmts=$HEVC_PARAMETERS
30
31 for arg in "$@"; do
32   case "$arg" in
33     bitrate=*)
34       BIT_RATES="${arg#*=}"
35       ;;
36   esac
37 done
38
39 bit_rates_=$(echo "$BIT_RATES" | tr ' ' '_')
40
41 path_to_results_base=\
42 "./results_${file_name_no_ext}_${file_config_name_no_ext}_${bit_rates_}/"
43
44 if [ "${file_extension}" = "y4m" ]; then
45   pix_fmt=""
46   s=""
47   r=""
48 fi
49
50 if [ "${vvenc_prmts}" != "" ]; then
51   vvenc_prmts="-vvenc-params ${vvenc_prmts}"
52 fi
53
54 if [ "${hevc_prmts}" != "" ]; then
55   hevc_prmts="-x265-params ${hevc_prmts}"
56 fi
57
58 if [ ${VVC_ENCODING_MODE} = "ABR" ]; then
59   path_to_results="${path_to_results_base}$codec/"
60   mkdir -m 755 -p ${path_to_results}
61   for rate in $BIT_RATES; do
62     echo "."
63     if [ $codec = "HEVC" ]; then
64       if [ "${ENCODE}" = "on" ]; then
65         { time ffmpeg $s $r ${pix_fmt} -i $file_name -c:v libx265 \
```

```
66          -preset ${ffmpeg_hevc_prmts} ${hevc_preset} -b:v ${rate}k \
67          ${hevc_prmts} -f hevc ${path_to_results}output_${rate}k.h265 ; \
68          } 2>> ${path_to_results}execution_times_${rate}k.txt
69        fi
70        if [ "${DECODE}" = "on" ]; then
71          ffmpeg -i ${path_to_results}output_${rate}k.h265 \
72          -pix_fmt ${PIX_FMT_FOR_ENC_DEC}\
73          ${path_to_results}output_decoded_HEVC_${rate}k.${file_extension} \
74          -loglevel error
75        fi
76      fi
77      if [ $codec = "VVC" ]; then
78        if [ "${ENCODE}" = "on" ]; then
79          { time ffmpeg $s $r ${pix_fmt} -i $file_name -c:v libvvenc \
80          ${ffmpeg_vvc_prmts} -preset ${vvc_preset} -b:v ${rate}k ${vvenc_prmts}
81        fi
82        if [ "${DECODE}" = "on" ]; then
83          ffmpeg -i ${path_to_results}output_${rate}k.266 \
84          -pix_fmt ${PIX_FMT_FOR_ENC_DEC} \
85          ${path_to_results}output_decoded_VVC_${rate}k.${file_extension} \
86          -loglevel error
87        fi
88      fi
89      if [ $codec = "AV1" ]; then
90        if [ "${ENCODE}" = "on" ]; then
91          { time ffmpeg $s $r ${pix_fmt} -i $file_name -c:v libaom-av1 \
92          -cpu-used ${av1_preset} -b:v ${rate}k \
93          -f ivf ${path_to_results}output_${rate}k.ivf ;\
94          } 2>> ${path_to_results}execution_times_${rate}k.txt
95        fi
96        if [ "${DECODE}" = "on" ]; then
97          ffmpeg -i ${path_to_results}output_${rate}k.ivf \
98          -pix_fmt ${PIX_FMT_FOR_ENC_DEC} \
99          ${path_to_results}output_decoded_AV1_${rate}k.${file_extension} \
100         -loglevel error
101       fi
102     fi
103     if [ $codec = "AV1-SVT" ]; then
104       if [ "${ENCODE}" = "on" ]; then
105         { time ffmpeg $s $r ${pix_fmt} -i $file_name -c:v libsvtav1 \
106         -preset ${av1_svt_preset} -b:v ${rate}k -f \
107         ivf ${path_to_results}output_${rate}k.ivf ; } \
108         2>> ${path_to_results}execution_times_${rate}k.txt
109       fi
110       if [ "${DECODE}" = "on" ]; then
111         ffmpeg -i ${path_to_results}output_${rate}k.ivf \
112         -pix_fmt ${PIX_FMT_FOR_ENC_DEC} \
113         ${path_to_results}output_decoded_AV1-SVT_${rate}k.${file_extension} \
114         -loglevel error
115       fi
116     fi
```

```
117   done
118 fi
```

A.3 VMAF_bit_rate.sh

```
 1 #!/bin/bash
 2
 3 file_config="$2"
 4 file_config_name_with_ext=$(basename "${file_config}")
 5 file_config_name_no_ext="${file_config_name_with_ext%.*}"
 6 file_config_extension="${file_config_name_with_ext##*.}"
 7 . ./${file_config}
 8
 9 for arg in "$@"; do
10   case "$arg" in
11     bitrate=*)
12       BIT_RATES="${arg#*=}"
13       ;;
14   esac
15 done
16
17 file_name=$1
18 file_name_ext=$(basename "$file_name")
19 file_name_no_ext="${file_name_ext%.*}"
20 bit_rates_=$(echo "$BIT_RATES" | tr ' ' '_')
21 codecs_=$(echo "$CODECS" | tr ' ' '_')
22 file_extension="${file_name##*.}"
23 pix_fmt="-p ${PIX_FMT_FOR_VMAF}"
24 bit_depth="-b ${BIT_DEPTH_FOR_VMAF}"
25
26 extract_y4m_metadata() {
27   fps=$(ffprobe -v error -select_streams v:0 -show_entries \
28   stream=r_frame_rate -of csv=p=0 "$file_name" | bc -l)
29   width=$(ffprobe -v error -select_streams v:0 -show_entries \
30   stream=width -of csv=p=0 "$file_name")
31   height=$(ffprobe -v error -select_streams v:0 -show_entries \
32   stream=height -of csv=p=0 "$file_name")
33   duration=$(ffprobe -v error -select_streams v:0 -show_entries \
34   format=duration -of csv=p=0 "$file_name")
35 }
36
37 get_preset_for_codec() {
38   case $1 in
39     "AV1")
40       echo "$AV1_PRESET"
41       ;;
42     "AV1-SVT")
43       echo "$AV1_SVT_PRESET"
44       ;;
45     "VVC")
```

```
46        case $VVC_PRESET in
47          "faster") echo "5" ;;
48          "fast") echo "4" ;;
49          "medium") echo "3" ;;
50          "slow") echo "2" ;;
51          "slower") echo "1" ;;
52          *) echo "Unknown" ;;
53        esac
54        ;;
55      "HEVC")
56        case $HEVC_PRESET in
57          "ultrafast") echo "8" ;;
58          "superfast") echo "7" ;;
59          "veryfast") echo "6" ;;
60          "faster") echo "5" ;;
61          "fast") echo "4" ;;
62          "medium") echo "3" ;;
63          "slow") echo "2" ;;
64          "veryslow") echo "1" ;;
65          "placebo") echo "0" ;;
66          *) echo "Unknown" ;;
67        esac
68        ;;
69      *)
70        echo "Unknown"
71        ;;
72    esac
73  }
74
75  calculate_actual_bitrate() {
76    file=$1
77    duration=$2
78    file_size=$(stat -c%s "$file")
79    echo "scale=2; ($file_size * 8) / ($duration * 1000)" | bc -l
80  }
81
82  extract_info_from_log() {
83    log_file=$1
84    real_time=$(grep "elapsed" "$log_file" | awk '{print $3}' \
85    | sed 's/elapsed//')
86    user_time=$(grep "user" "$log_file" | awk '{print $1}' | sed 's/user//')
87    sys_time=$(grep "sys" "$log_file" | awk '{print $2}' | sed 's/system//')
88    sys_time=$(echo "$sys_time" | grep -oP '[\d\.]+') # Extract only numeric \
89    value
90
91    case $codec in
92      "AV1")
93        codec_library="libaom-av1"
94        ;;
95      "AV1-SVT")
96        codec_library="libsvtav1"
```

```
97        ;;
98      "VVC")
99        codec_library="libvvenc"
100       ;;
101     "HEVC")
102       codec_library="libx265"
103       ;;
104   esac
105
106   echo "$real_time,$user_time,$sys_time,$codec_library"
107 }
108
109 calculate_averages() {
110   json_file=$1
111   vmaf_avg=$(jq '[.frames[].metrics.vmaf] | add / length' "$json_file")
112   psnr_y_avg=$(jq '[.frames[].metrics.psnr_y] | add / length' "$json_file")
113   float_ssim_avg=$(jq '[.frames[].metrics.float_ssim] \
114   | add / length' "$json_file")
115   echo "$vmaf_avg,$psnr_y_avg,$float_ssim_avg"
116 }
117
118 if [ "$file_extension" = "y4m" ]; then
119   pix_fmt=""
120   bit_depth=""
121   fps=""
122   duration=""
123   width=""
124   height=""
125 else
126   width="-w $3"
127   height="-h $4"
128   pix_fmt="-p ${PIX_FMT_FOR_VMAF}"
129   bit_depth="-b ${BIT_DEPTH_FOR_VMAF}"
130   duration="N/A"
131 fi
132
133 path_to_results_base=\
134 "./results_${file_name_no_ext}_${file_config_name_no_ext}_${bit_rates_}/"
135 actual_bitrate_file="${path_to_results_base}actual_bitrate.txt"
136
137 mkdir -p "$path_to_results_base"
138 echo "codec,rate,actual_bitrate" > "$actual_bitrate_file"
139
140 output_csv=\
141 "./results_${file_name_no_ext}_${file_config_name_no_ext}_${bit_rates_}.csv"
142
143 echo "seq_name,fps,duration,w,h,bitrate,codec,preset,vmaf,psnr_y,float_ssim, \
144 real,user,sys,actual_bitrate" > $output_csv
145
146 if [ ${VVC_ENCODING_MODE} = "ABR" ]; then
147   for codec in $CODECS; do
```

```
148     path_to_results="${path_to_results_base}$codec/"
149     mkdir -p "$path_to_results"
150     for rate in $BIT_RATES; do
151       echo "_________________________________"
152       echo "$codec (bit rate: $rate kbit/s)"
153
154       enc_info=\
155       $(extract_info_from_log \
156       "${path_to_results}execution_times_${rate}k.txt")
157       real_time=$(echo $enc_info | cut -d',' -f1)
158       user_time=$(echo $enc_info | cut -d',' -f2)
159       sys_time=$(echo $enc_info | cut -d',' -f3)
160       codec_library=$(echo $enc_info | cut -d',' -f4)
161
162       preset=$(get_preset_for_codec "$codec")
163
164       case $codec in
165         "AV1")
166           compressed_file="${path_to_results}output_${rate}k.ivf"
167           ;;
168         "AV1-SVT")
169           compressed_file="${path_to_results}output_${rate}k.ivf"
170           ;;
171         "VVC")
172           compressed_file="${path_to_results}output_${rate}k.266"
173           ;;
174         "HEVC")
175           compressed_file="${path_to_results}output_${rate}k.h265"
176           ;;
177       esac
178
179       ./vmaf --reference "$file_name" --distorted \
180       "${path_to_results}output_decoded_${codec}_${rate}k.${file_extension}" \
181         $width $height $pix_fmt $bit_depth \
182         --model version=vmaf_float_v0.6.1 -o \
183         "${path_to_results}results_${codec}_${rate}k.json" \
184         --json --feature psnr --feature float_ssim
185       echo "_________________________________"
186
187       if [ "$file_extension" = "y4m" ]; then
188         extract_y4m_metadata
189       fi
190
191       actual_bitrate=\
192       $(calculate_actual_bitrate "${compressed_file}" "${duration}")
193       metrics=\
194       $(calculate_averages "${path_to_results}results_${codec}_${rate}k.json")
195
196       echo "$codec,$rate,$actual_bitrate" >> "$actual_bitrate_file"
197
198       echo \ "$file_name_no_ext,$fps,$duration,$width,$height,$rate, \
```

```
199       $codec_library,$preset,$metrics,$real_time,$user_time,$sys_time, \
200       $actual_bitrate" >> $output_csv
201
202       fps=""
203       duration=""
204       width=""
205       height=""
206     done
207   done
208 fi
```

A.4 Script di Configurazione e Building del Container Singularity

```
1  Bootstrap: docker
2  From: ubuntu:24.04
3
4  %post
5      apt-get update
6      apt-get install -y cmake g++ nasm pkg-config git libxml2-dev \
7      libx265-dev libaom-dev bc jq
8      apt-get install -y python3-full python3-matplotlib
9      apt-get install -y time
10
11     git clone https://github.com/fraunhoferhhi/vvenc.git
12     cd vvenc
13     git checkout v1.12.1
14     mkdir build && cd build
15     cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/usr/local ..
16     make
17     make install
18     cd ../..
19
20     git clone https://gitlab.com/AOMediaCodec/SVT-AV1.git
21     cd SVT-AV1
22     git checkout v2.3.0
23     mkdir build && cd build
24     cmake .. -G"Unix Makefiles" -DCMAKE_BUILD_TYPE=Release
25     make -j$(nproc)
26     make install
27     ldconfig
28     cd ../../
29
30     git clone https://git.ffmpeg.org/ffmpeg.git ffmpeg
31     cd ffmpeg
32     git checkout master
33     ./configure \
34     --enable-pthreads \
35     --enable-pic \
```

```
36      --enable-shared \
37      --enable-rpath \
38      --arch=amd64\
39       --enable-demuxer=dash \
40       --enable-libxml2 \
41       --enable-libvvenc \
42       --enable-gpl \
43       --enable-libx265 \
44       --enable-libaom \
45       --enable-libsvtav1 \
46       --enable-gpl \
47       --enable-nonfree
48      make -j$(nproc)
49      make install
50
51  %environment
52      # Imposta l'ambiente virtuale nel PATH
53      export PATH="/usr/local/bin:/opt/vmaf_env/bin:/root/.local/bin:$PATH"
54
55  %runscript
56      echo "VVC container"
57
58  %labels
59      Author="Federico Baldassi"
60      Version="1.0"
```

Bibliografia

Peak signal-to-noise ratio, from Wikipedia, the free encyclopedia. URL https://it.wikipedia.org/wiki/Peak_signal-to-noise_ratio.

AV1 Codec Library, a. URL https://aomedia.googlesource.com/aom.

aomenc, from Codec Wiki., b. URL https://wiki.x266.mov/docs/encoders/aomenc.

AV1 Video Encoding Guide. URL https://trac.ffmpeg.org/wiki/Encode/AV1.

Average bitrate. URL https://en.wikipedia.org/wiki/Average_bitrate.

Cisco Annual Internet Report (2018–2023) White Paper. URL https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

Compressione video digitale, from Wikipedia, the free encyclopedia. URL https://it.wikipedia.org/wiki/Compressione_video_digitale.

Constant Quality vs Average Bit Rate. URL https://handbrake.fr/docs/en/latest/technical/video-cq-vs-abr.html.

FFmpeg Integration. URL https://github.com/fraunhoferhhi/vvenc/wiki/FFmpeg-Integration.

High Efficiency Video Coding (HEVC), a. URL https://aws.amazon.com/it/media/tech/high-efficiency-video-coding/.

H.265/HEVC Video Encoding Guide, b. URL https://trac.ffmpeg.org/wiki/Encode/H.265.

Cos'è l'High Performance Computing (HPC)? URL https://www.redhat.com/it/topics/high-performance-computing/what-is-high-performance-computing.

Singularity documentation, a. URL https://docs.sylabs.io/guides/3.0/user-guide/quick_start.html.

Singularity (software), from Wikipedia, the free encyclopedia., b. URL https://en.wikipedia.org/wiki/Singularity_(software).

SVT-AV1. URL https://gitlab.com/AOMediaCodec/SVT-AV1.

Video Multimethod Assessment Fusion. URL https://en.wikipedia.org/wiki/Video_Multimethod_Assessment_Fusion.

VVC (VVenC) vs AV1 (Aomenc) comparison. URL https://www.reddit.com/r/AV1/comments/kcvjpo/vvc_vvenc_vs_av1_aomenc_comparison/.

VVenC, from Codec Wiki., a. URL https://wiki.x266.mov/docs/encoders/VVenC.

VVenC Presets, b. URL https://github.com/fraunhoferhhi/vvenc/wiki/Presets.

VVenC, c. URL https://github.com/fraunhoferhhi/vvenc.

VVenC Usage, d. URL https://github.com/fraunhoferhhi/vvenc/wiki/Usage.

x265 Documentation. URL https://x265.readthedocs.io/en/master/.

[BAR2018MLNR] N. Barman, E. Jammeh, A. Ghorashi, and M. G. Martini. No-reference Video Quality Estimation Based on Machine Learning for Passive Gaming Video Streaming Applications. *IEEE Access, May 2019. 10.1109/ACCESS.2019.2920477.*

Rajitha Weerakkody James Bruce, Marta Mrak. Testing AV1 and VVC. 2019. URL https://www.bbc.co.uk/rd/blog/2019-05-av1-codec-streaming-processing-hevc-vvc.

Juraj Bienik Lenka Smatanova Miroslav Uhrina, Lukas Sevcik. Performance comparison of vvc, av1, hevc, and avc for high resolutions. 2024. URL https://www.mdpi.com/2079-9292/13/5/953.

N. Barman, S. Schmidt, S. Zadtootaghaj, M. G. Martini, and S. Möller. An Evaluation of Video Quality Assessment Metrics for Passive Gaming Video Streaming. *In Proceedings of the 23rd Packet Video Workshop (PV '18). ACM, Amsterdam, Netherlands, 2018, pp. 7-12. DOI: https://doi.org/10.1145/3210424.3210434.*

N. Barman, S. Zadtootaghaj, S. Schmidt, M. G. Martini and S. Möller. GamingVideoSET: A Dataset for Gaming Video Streaming Applications. *2018 16th Annual Workshop on Network and Systems Support for Games (NetGames), Amsterdam, Netherlands, 2018, pp. 1-6. DOI: 10.1109/NetGames.2018.8463362.*

Rao, Rakesh Rao Ramachandra. AVT-VQDB-UHD-1: A large scale video quality database for UHD-1. *2019 IEEE International Symposium on Multimedia (ISM). IEEE*, 2019.

Kaley Torres. H.266 VVC vs AV1, Which is Better? 2023. URL https://www.winxdvd.com/video-transcoder/h266-vvc-vs-av1.htm.

Krishna Rao Vijayanagar. Comparing SVT-AV1 Presets: Size, Quality, and Speed with CRF Variations. 2023. URL https://ottverse.com/analysis-of-svt-av1-presets-and-crf-values/.