

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL  
INSTITUTO DE INFORMÁTICA  
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

MATEUS GRELLERT

**Machine Learning Mode Decision for  
Complexity Reduction and Scaling in Video  
Applications**

Thesis presented in partial fulfillment  
of the requirements for the degree of  
Doctor of Computer Science

Advisor: Prof. Dr. Sergio Bampi  
Coadvisor: Prof. Dr. Bruno Zatt

Porto Alegre  
April 2018

## CIP — CATALOGING-IN-PUBLICATION

Grellert, Mateus

Machine Learning Mode Decision for Complexity Reduction and Scaling in Video Applications / Mateus Grellert. – Porto Alegre: PPGC da UFRGS, 2018.

196 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2018. Advisor: Sergio Bampi; Coadvisor: Bruno Zatt.

1. Video coding. 2. Video transcoding. 3. Complexity reduction. 4. Complexity scaling. 5. Machine Learning. 6. HEVC.  
I. Bampi, Sergio. II. Zatt, Bruno. III. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Reitor: Prof. Rui Vicente Oppermann

Vice-Reitora: Profª. Jane Fraga Tutikian

Pró-Reitor de Pós-Graduação: Prof. Celso Giannetti Loureiro Chaves

Diretora do Instituto de Informática: Profª. Carla Maria Dal Sasso Freitas

Coordenador do PPGC: Prof. João Luiz Dihl Comba

Bibliotecária-chefe do Instituto de Informática: Beatriz Regina Bastos Haro

*“O light! This is the cry of all characters of ancient drama brought face to face with their fate. This last resort was ours, too, and I knew it now. In the middle of winter I at last discovered that there was in me an invincible summer.”*

— ALBERT CAMUS



## ACKNOWLEDGEMENTS

First and foremost, I would like to thank all of my mentors, Professors Sergio Bampi, Bruno Zatt, and Luis Cruz, without whom this PhD would not be possible. Your demands required a lot of effort at times, but this is precisely what I needed to continue my work. You always believed in my capacity and helped me without compromising my own decisions at the same time. Now I see you more as friends than as superiors, but my respect for you remains the same. Your guidance was very important during this journey and will be always remembered.

An important thanks to my best friend, my most faithful companion, and the love of my life, Gregory. Being a post-grad student for so long is not always easy, but you always supported me and made me feel safe during my anxiety crises. We've been together for 11 years now. During this time, we faced many happy and some difficult moments, but we always got through it all. We make each other better every day, and I hope I'm lucky enough to keep it this way for the rest of my life.

I also want to thank the important friends of the laboratory 215, Ana, Brunno, Dieison, Duda, Leonardo, LM, Paim, and more recently Thomas and Gustavo. We are simply the best lab team of the Department of Informatics, and this is all thanks to you guys. Thank you for making the lab hours much more fun, full of laughter and coffee, and also for accepting me for who I am. Thank you to my friends in Porto Alegre, Andrws, Bruna, Cláudio, Cilene, Marta, with which I spent so many hours talking about life, politics, and everyday matters, always with a good drink and a good laugh. You all are part of this and will continue to be part of my life.

A big thank you to all the friends I made in Portugal. Starting with the ones from the LPI lab, Cristiano, Iago, Rafael, and Tiago. You helped me feel at home right at the beginning and always filled the lab with joy. A special thanks to Cristiano, for being the coffee maker and keeping everyone awake with his coffee and his terrible jokes (I miss those jokes for some strange reason though). The friends I made out of the lab also filled my days in Portugal with love. Irene, Nathale, Raquel, Gonçalo, and many others, our moments together were amazing, and I will treasure them always.

To my oldest friends, Andressa, Bruno, Digue, Finno, Julio, I want to say that we will always be together. I see us growing old, living together, going out to parties and dancing like it's the end of the world. A friendship like ours is not easy to find, and I feel very grateful that I was blessed with this fortune not one but five times.

I also want to thank my family, for being so supportive and for helping when I needed. I feel overflowed with love and happiness every time we are together.

Finally, thank you CAPES, CNPq, FCT, and every funding agency that supported my studies. I will do my best to show myself worthy of your investment, joining the effort to increase Brazil's participation in the scientific activities.

## ABSTRACT

The recent innovations in Machine Learning techniques have led to a large utilization of intelligent models to solve complex problems that are especially hard to compute with traditional data structures and algorithms. In particular, the current research on Image and Video Processing shows that it is possible to design Machine Learning models that perform object recognition and even action recognition with high confidence levels. In addition, the latest progress on training algorithms for Deep Learning Neural Networks was also an important milestone in Machine Learning, leading to prominent discoveries in Computer Vision and other applications. Recent studies have also shown that it is possible to design intelligent models capable of drastically reducing the optimization space of mode decision in video encoders with minor losses in coding efficiency. All these facts indicate that Machine Learning for complexity reduction in visual applications is a very promising field of study. The goal of this thesis is to investigate learning-based techniques to reduce the complexity of the HEVC encoding decisions, focusing on fast video encoding and transcoding applications. A complexity profiling of HEVC is first presented to identify the tasks that must be prioritized to accomplish our objective. Several variables and metrics are then extracted during the encoding and decoding processes to assess their correlation with the encoding decisions associated with these tasks. Next, Machine Learning techniques are employed to construct classifiers that make use of this information to accurately predict the outcome of these decisions, eliminating the time-consuming operations required to compute them. The fast encoding and transcoding solutions were developed separately, as the source of information is different on each case, but the same methodology was followed in both cases. In addition, mechanisms for complexity scalability were developed to provide the best rate-distortion performance given a target complexity reduction. Experimental results demonstrated that the designed fast encoding solutions achieve time savings of 37% up to 78% on average, with Bjontegaard Delta Bitrate (BD-BR) increments between 0.04% and 4.8%. In the transcoding results, a complexity reduction ranging from 43% to 67% was observed, with average BD-BR increments from 0.34% up to 1.7%. Comparisons with state of the art confirm the efficacy of the designed methods, as they outperform the results achieved by related solutions.

**Keywords:** Video coding. Video transcoding. Complexity reduction. Complexity scaling. Machine Learning. HEVC.



## **RESUMO**

As recentes inovações em técnicas de Aprendizado de Máquina levaram a uma ampla utilização de modelos inteligentes para resolver problemas complexos que são especialmente difíceis de computar com algoritmos e estruturas de dados convencionais. Em particular, pesquisas recentes em Processamento de Imagens e Vídeo mostram que é possível desenvolver modelos de Aprendizado de Máquina que realizam reconhecimento de objetos e até mesmo de ações com altos graus de confiança. Além disso, os últimos avanços em algoritmos de treinamento para Redes Neurais Profundas (Deep Learning Neural Networks) estabeleceram um importante marco no estudo de Aprendizado de Máquina, levando a descobertas promissoras em Visão Computacional e outras aplicações. Estudos recentes apontam que também é possível desenvolver modelos inteligentes capazes de reduzir drasticamente o espaço de otimização do modo de decisão em codificadores de vídeo com perdas irrelevantes em eficiência de compressão. Todos esses fatos indicam que Aprendizado de Máquina para redução de complexidade em aplicações de vídeo é uma área promissora para pesquisa. O objetivo desta tese é investigar técnicas baseadas em aprendizado para reduzir a complexidade das decisões da codificação HEVC, com foco em aplicações de codificação e transcodificação rápidas. Um perfilamento da complexidade em codificadores é inicialmente apresentado, a fim de identificar as tarefas que requerem prioridade para atingir o objetivo dessa tese. A partir disso, diversas variáveis e métricas são extraídas durante os processos de codificação e decodificação para avaliar a correlação entre essas variáveis e as decisões de codificação associadas a essas tarefas. Em seguida, técnicas de Aprendizado de Máquina são empregadas para construir classificadores que utilizam a informação coletada para prever o resultado dessas decisões, eliminando o custo computacional necessário para computá-las. As soluções de codificação e transcodificação foram desenvolvidas separadamente, pois o tipo de informação é diferente em cada caso, mas a mesma metologia foi aplicada em ambos os casos. Além disso, mecanismos de complexidade escalável foram desenvolvidos para permitir o melhor desempenho taxa-compressão para um dado valor de redução de complexidade. Resultados experimentais apontam que as soluções desenvolvidas para codificação rápida atingiram reduções de complexidade entre 37% e 78% na média, com perdas de qualidade entre 0.04% e 4.8% (medidos em Bjontegaard Delta Bitrate – BD-BR). Já as soluções para

transcodificação rápida apresentaram uma redução de 43% até 67% na complexidade, com BD-BR entre 0.34% e 1.7% na média. Comparações com o estado da arte confirmam a eficácia dos métodos desenvolvidos, visto que são capazes de superar os resultados atingidos por soluções similares.

**Palavras-chave:** Codificação de vídeo. Transcodificação de Vídeo Redução de complexidade. Complexidade escalável. Aprendizado de Máquina. HEVC.

## **LIST OF ABBREVIATIONS AND ACRONYMS**

ABR	Average Bitrate
ALF	Adaptive Loop Filter
AMP	Asymmetric Motion Partitions
AVC	Advanced Video Coding
BD	Bjøntegaard Difference
BD-BR	Bjøntegaard Difference Bitrate
BR	bitrate
CABAC	context-adaptive binary arithmetic coding
CBR	Constant Bitrate
CTC	Common Test Conditions
CTU	Coding Tree Unit
CU	Coding Unit
DCT	Discrete Cosine Transform
ELM	Extreme Learning Machines
ET	Early Termination
FME	Fractional Motion Estimation
FN	False Negative
FP	False Positive
FR	Feature Reduction
GOP	Groups of Pictures
HD	High Definition
HEVC	High Efficiency Video Coding
HM	HEVC Model
HQ	High Quality

HVS	Human Visual System
ID3	Iterative Dichotomizer 3
IG	Information Gain
IGR	Information Gain Ratio
IME	Integer Motion Estimation
IQ	Inverse Quantization
IT	Inverse Transforms
ITU-T	ITU Telecommunication Standardization
IV	Intrinsic Value
JCT-VC	Joint Collaborative Team on Video Technology
JVET	Joint Video Exploration Team on Future Video coding
LB	Low Delay
LGP	Local Gradient Pattern
LQ	Low Quality
ME	Motion Estimation
ML	Machine Learning
MPEG	Moving Picture Experts Group
MRF	Mean Random Forest
MSM	Merge/Skip mode
PCM	Pulse Code Modulation
PID	Proportional, Integral, Derivative
PM	Prediction Mode
PSNR	Peak Signal-to-Noise Ratio
PU	Prediction Unit
Q	Quantization
QP	Quantization Parameter

RA	Random Access
RBF	Radial Basis Function
RD	Rate Distortion
RDC	Rate Distortion Complexity
RDM	Rough Mode Decision
RDO	Rate-Distortion Optimization
RDOQ	Rate-Distortion Optimized Quantization
RF	Random Forest
RQT	Residual Quadtree
SAD	Sum of Absolute Differences
SATD	Sum of Absolute Transformed Differences
SMP	Symmetric Motion Partitions
SSE	Sum of Squared Error
SVM	Support Vector Machine
T	Transforms
TN	True Negative
TNR	True Negative Rate
TP	True Positive
TPC	Thermal Policy Characterization Table
TPR	True Positive Rate
TS	Time Savings
TU	Transform Unit
TZS	Test Zone Search
VBR	Variable Bitrate
VOD	Video On Demand



## LIST OF FIGURES

Figure 2.1 Group of Pictures (GOP) example using I, P, and B frames. (source: (BOSSEN; FLYNN; SÜHRING, 2013)) .....	29
Figure 2.2 HEVC encoding loop and the main tasks involved. ....	30
Figure 2.3 Example of a CTU and its respective CU and PU partitioning decisions. The gray-filled blocks are evaluated, but not used in the final encoding. ....	31
Figure 2.4 Example of a residual block (a), the DCT coefficients of this block (b), and the coefficients after quantization (c). ....	36
Figure 2.5 Random Access and Low Delay temporal configurations used in HEVC encoders. (source: (KIM et al., 2013)) .....	38
Figure 2.6 Rate-distortion plots between a reference (REF) and a test (TEST) en- coder and the respective BD-PSNR (a) and BD-BR (b) areas shaded in gray.....	39
Figure 2.7 DASH framework illustrating the interactions between the server (on the left side) and the client (right side) in an adaptive streaming system. (adapted from (BITMOVIN, 2018)) .....	41
Figure 3.1 Subdivisions of Machine Learning concepts.....	46
Figure 3.2 Class distribution over two labels and three different classifiers. (source: (LORENA; CARVALHO, 2007)) .....	47
Figure 3.3 A Decision Tree for classification of the Iris dataset. (adapted from (SCIKIT, 2018)).....	48
Figure 3.4 (a) Three classifiers with inefficient decision margins and (b) an SVM linear classifier with maximum margin.....	50
Figure 3.5 Example of the kernel function employed in SVMs to transform lin- early inseparable data (a) into an equivalent distribution that is separable by a hyperplane (b). .....	51
Figure 4.1 Main tasks required to encode a single CTU of an HEVC encoder. ....	58
Figure 4.2 Processing time and average complexity share of several encoding com- ponents for three video resolutions (sorted by average time share).....	58
Figure 4.3 Complexity share breakdown of encoding components for three video resolutions (sorted by average time share).....	59
Figure 4.4 Motion Estimation complexity share of in each reference frame (a) and in each Prediction Unit group (b) – sequence: BQMall ( $832 \times 480$ ).....	60
Figure 4.5 Encoding time share spent to process each CU size. The results were averaged for each QP (a) and for each sequence (b).....	60
Figure 4.6 Encoding time spent on the main HEVC tools for two sequences with $2560 \times 1600$ resolution. ....	61
Figure 4.7 Average encoding time spent on the RQT, IME, and FME processes for the BasketballDrive and ParkScene sequences ( $1920 \times 1080$ ). ....	62
Figure 5.1 Data collection scheme for the HEVC encoding decisions.....	77
Figure 5.2 Occurrence of CU splits on $64 \times 64$ CUs. ....	79
Figure 5.3 Occurrence of CU splits on $32 \times 32$ CUs. ....	80
Figure 5.4 Occurrence of CU splits on $16 \times 16$ CUs. ....	80
Figure 5.5 Information gain Ratio of the input features considering three types of data separation: using all samples (a); separating by QP size (b); and separat- ing by CU size (c). ....	83

Figure 5.6 F-Score of the top ten features using this criterion for CU sizes $64 \times 64$ , $32 \times 32$ , and $16 \times 16$ .....	84
Figure 5.7 NRF score and standard deviation of the top ten features using this criterion for CU sizes $64 \times 64$ , $32 \times 32$ , and $16 \times 16$ .....	84
Figure 5.8 Occurrence of non- $2N \times 2N$ PUs on $64 \times 64$ CUs.....	85
Figure 5.9 Occurrence of non- $2N \times 2N$ PUs on $32 \times 32$ CUs.....	86
Figure 5.10 Occurrence of non- $2N \times 2N$ PUs on $16 \times 16$ CUs.....	86
Figure 5.11 Occurrence of non- $2N \times 2N$ PUs on $8 \times 8$ CUs.....	87
Figure 5.12 IGR score of the top ten features for the PU partitioning problem (CU sizes $64 \times 64$ , $32 \times 32$ , $16 \times 16$ , and $8 \times 8$ ).....	87
Figure 5.13 F-score of the top ten features for the PU partitioning problem (CU sizes $64 \times 64$ , $32 \times 32$ , $16 \times 16$ , and $8 \times 8$ ).....	87
Figure 5.14 Occurrence of RQT splits on $64 \times 64$ CUs.....	88
Figure 5.15 Occurrence of RQT splits on $32 \times 32$ CUs.....	89
Figure 5.16 Occurrence of RQT splits on $16 \times 16$ CUs.....	89
Figure 5.17 Occurrence of RQT splits on $8 \times 8$ CUs.....	90
Figure 5.18 IGR score of the top ten features for the RQT partitioning problem (CU sizes $64 \times 64$ , $32 \times 32$ , $16 \times 16$ , and $8 \times 8$ ).....	90
Figure 5.19 F-score of the top ten features for the RQT partitioning problem (CU sizes $64 \times 64$ , $32 \times 32$ , $16 \times 16$ , and $8 \times 8$ ).....	90
Figure 5.20 Occurrence of non-zero reference frame indices on $64 \times 64$ CUs.....	91
Figure 5.21 Occurrence of non-zero reference frame indices on $32 \times 32$ CUs.....	92
Figure 5.22 Occurrence of non-zero reference frame indices on $16 \times 16$ CUs.....	92
Figure 5.23 Occurrence of non-zero reference frame indices on $8 \times 8$ CUs.....	93
Figure 5.24 IGR score of the top ten features for the ME early termination problem (CU sizes $64 \times 64$ , $32 \times 32$ , $16 \times 16$ , and $8 \times 8$ ).....	93
Figure 5.25 F-score of the top ten features for the ME early termination problem (CU sizes $64 \times 64$ , $32 \times 32$ , $16 \times 16$ , and $8 \times 8$ ).....	94
Figure 5.26 Occurrence of motion vectors with fractional offsets on $64 \times 64$ CUs.....	94
Figure 5.27 Occurrence of motion vectors with fractional offsets on $32 \times 32$ CUs.....	95
Figure 5.28 Occurrence of motion vectors with fractional offsets on $16 \times 16$ CUs.....	95
Figure 5.29 Occurrence of motion vectors with fractional offsets on $8 \times 8$ CUs.....	96
Figure 5.30 IGR score of the top ten features for the FME skip problem (CU sizes $64 \times 64$ , $32 \times 32$ , $16 \times 16$ , and $8 \times 8$ ).....	96
Figure 5.31 F-score of the top ten features for the FME skip problem (CU sizes $64 \times 64$ , $32 \times 32$ , $16 \times 16$ , and $8 \times 8$ ).....	96
Figure 5.32 Data collection scheme for the HEVC transcoding decisions.....	98
Figure 5.33 Average distribution of motion vector deltas between the HQ bitstream and the LQ representations.....	103
Figure 5.34 Maximum (a) and minimum (b) CU size on CTUs at different bitrate factors.....	104
Figure 5.35 Information Gain Ratio of the CTU-grain transcoding features for prediction the maximum (a) and minimum (b) CU size in each CTU.....	106
Figure 5.36 Probability density functions of all classes in function of the most important feature .....	107
Figure 6.1 Overview of the contributions presented in this thesis for encoding applications.....	109
Figure 6.2 Framework used in this work for training CU partitioning decision SVM classifiers.....	111

Figure 6.3 Decision margin for the 64x64 CU partition decision using (a) SVM with an RBF kernel and (b) Decision Tree classifiers (simplified instance).....	112
Figure 6.4 Split decision for different values of $Th_{Split}$ .....	118
Figure 6.5 Average Time Savings, BD-BR increment, and their linear and polynomial fits, for different values of $Th_{Split}$ .....	118
Figure 6.6 Partition Similarity of the BasketballDrive sequence for $Th_{Split} = 0.5$ ....	123
Figure 6.7 Rate-Distortion plots for the ChinaSpeed sequence, using different values of $Th_{Split}$ .....	124
Figure 6.8 CU Early Termination scheme.....	126
Figure 6.9 TS and BD-BR for different values of $Split_{th}$ .....	129
Figure 6.10 Block diagram of the RQT ET application.....	133
Figure 6.11 Block diagram of the ME ET application.....	135
Figure 6.12 Block diagram of the FME ET application. ....	136
Figure 6.13 Average BD-BR/TS plot of the tested decision threshold combinations combinations for sequences RaceHorsesC (832x480) and Basketball-Pass (416x240).....	138
Figure 6.14 Rate-distortion curves of the proposed CU+PU+RQT ET methods using several combinations of decision thresholds (sequence: BQTerrace, resolution: 1920x1080@60fps, QP=22, 27, 32, 37).....	140
 Figure 7.1 Overview of the contributions presented in this thesis for transrating applications .....	141
Figure 7.2 Methodology used to apply the trained Random Forests in a fast HEVC encoder .....	142
Figure 7.3 Methodology used to train the CTU-level classifiers. ....	143
Figure 7.4 Rate-distortion curves of the proposed statiscal-based ET methods (sequence: BQTerrace, resolution: 1920x1080@60fps, $\alpha_{BR} = 0.2, 0.4, 0.6, 0.8$ )....	156
Figure 7.5 Rate-distortion curves of the proposed statiscal-based ET methods combined with the $CTU_{RF}$ ET method (sequence: BQTerrace, resolution: 1920x1080@60fps, $\alpha_{BR} = 0.2, 0.4, 0.6, 0.8$ ).....	157



## LIST OF TABLES

Table 1.1 Coding tools used in the H.264/AVC and HEVC standards .....	25
Table 2.1 Enabled/Disabled partitions for each prediction mode and each CU size .....	32
Table 4.1 CU evaluation table of the method proposed by (VAN et al., 2013).....	71
Table 5.1 Video sequences used in the analysis .....	76
Table 5.2 Target bitrates (in kbps) used to encode the HQ bitstreams ( $HQ_{BR}$ ) as well as for every bitrate factor used in this work .....	98
Table 5.3 CU size probabilities given the size in the HQ bitstream.....	100
Table 5.4 PU partition probabilities given the partition in the HQ bitstream .....	101
Table 5.5 Minimum transform size probabilities given the minimum sizes in the HQ bitstream.....	101
Table 5.6 Prediction mode probabilities given the minimum sizes in the HQ bitstream	102
Table 6.1 Video sequences used for training and testing in the encoding fast decision methods .....	110
Table 6.2 Setup used for SVM training.....	112
Table 6.3 Characteristics of the SVM classifiers obtained using Algorithm 2 .....	115
Table 6.4 BD-BR, Time Savings and BDTS <sub>Ratio</sub> relative to HM 16.8 using SVMs under Variable Bitrate and Constant Bitrate conditions (VBR QPs = 22, 27, 32, 37 , CBR Targets = Bitrate <sub>QP22</sub> , Bitrate <sub>QP27</sub> , Bitrate <sub>QP32</sub> , Bitrate <sub>QP37</sub> ) .....	116
Table 6.5 BD-BR, Time Savings and BDTS <sub>Ratio</sub> relative to HM 16.8 using SVMs under Variable Bitrate and Constant Bitrate conditions (VBR QPs = 22, 27, 32, 37 , CBR Targets = Bitrate <sub>QP22</sub> , Bitrate <sub>QP27</sub> , Bitrate <sub>QP32</sub> , Bitrate <sub>QP37</sub> ) .....	117
Table 6.6 Rate-Distortion-Complexity of the test sequences when $Th_{Split} = 0.1$ (Random Access, QP=22, 27, 32, 37).....	120
Table 6.7 Rate-Distortion-Complexity of the test sequences when $Th_{Split} = 0.3$ (Random Access, QP=22, 27, 32, 37).....	121
Table 6.8 Rate-Distortion-Complexity of the test sequences when $Th_{Split} = 0.7$ (Random Access, QP=22, 27, 32, 37).....	122
Table 6.9 Comparison with related work .....	125
Table 6.10 Train accuracy, true positive rate, and tree depth of the trained classifiers (200,000 vectors per data set) .....	125
Table 6.11 Average usage of the top ten features considering both data sets .....	126
Table 6.12 Rate-distortion and complexity results of the proposed method for all tested sequences using $Th_{Split} = 0.5$ . (QP = 22, 27, 32, 37) .....	128
Table 6.13 Comparison with the related work .....	129
Table 6.14 Rate-Distortion and complexity reduction achieved using the PU fast decision method, as well as a combined CU+PU decision.....	132
Table 6.15 Rate-Distortion and complexity reduction achieved using the RQT ET method, as well as a combined CU+RQT ET.....	133
Table 6.16 Rate-Distortion and complexity reduction achieved using the RF skip early termination method, as well as a combined CU+PU+RQT+ME Early Termination methods. ....	136
Table 6.17 Rate-Distortion and complexity reduction achieved using the FME skip early termination method, as well as a combined decision with all the proposed methods.....	137

Table 6.18 BD-BR and Time savings achieved using the CU+PU+RQT ET methods for several combinations of decision thresholds .....	139
Table 7.1 Train and test sequences used in the fast transcoding schemes.....	144
Table 7.2 Top 10 Features Ranked by Information Gain .....	145
Table 7.3 Transrating results using the full set of features.....	146
Table 7.4 Average results for all dimensionality reduction methods and number of features.....	147
Table 7.5 Best results for each Dimensionality Reduction method .....	148
Table 7.6 Performance comparison with related works .....	149
Table 7.7 Performance of CU depth limitation strategy using direct mapping from Data <sub>HQ</sub> and comparison with the learning-based CTU-level fast decision .....	150
Table 7.8 PU partitions evaluated in the fast PU strategy using the information from the HQ reference (PU <sub>HQ</sub> ) .....	151
Table 7.9 Performance of MV inheritance strategy using the vector from the reference stream as candidate.....	154
Table 7.10 BD-BR and Time Savings of each fast transcoding method employed separately .....	154
Table 7.11 Rate-Distortion and Complexity Performance of all the statiscal-based method proposed, as well as their best combinations (in terms of combined BD-BR/TS ratio).....	155
Table 7.12 Rate-Distortion and Complexity Performance of all the statiscal-based methods combined with the CTU-level fast decision. ....	155
Table 7.13 Comparison of the combined fast transcoding methods with related work (sorted by average time savings).....	157

## CONTENTS

<b>1 INTRODUCTION.....</b>	<b>23</b>
<b>1.1 Problem Formulation .....</b>	<b>25</b>
<b>1.2 Thesis Claim .....</b>	<b>27</b>
<b>1.3 Thesis Organization .....</b>	<b>28</b>
<b>2 HIGH EFFICIENCY VIDEO CODING .....</b>	<b>29</b>
<b>2.1 Prediction in the HEVC standard .....</b>	<b>31</b>
<b>2.2 Intra Prediction .....</b>	<b>32</b>
<b>2.3 Inter Prediction .....</b>	<b>33</b>
<b>2.4 Transforms and Quantization.....</b>	<b>34</b>
<b>2.5 Entropy Coding.....</b>	<b>35</b>
<b>2.6 Rate-Distortion Optimization .....</b>	<b>36</b>
<b>2.7 Common Test Conditions .....</b>	<b>37</b>
<b>2.8 Metric for Assessing Coding Performance .....</b>	<b>37</b>
<b>2.9 Variable Bitrate and Average Bitrate Coding .....</b>	<b>39</b>
<b>2.10 HEVC Transcoding.....</b>	<b>40</b>
<b>2.11 Final Remarks .....</b>	<b>43</b>
<b>3 MACHINE LEARNING .....</b>	<b>45</b>
<b>3.1 Binary Classification Models .....</b>	<b>46</b>
<b>3.2 Algorithms for Building Classification Models .....</b>	<b>47</b>
3.2.1 Decision Trees.....	48
3.2.2 Support Vector Machines.....	50
3.2.3 Ensemble Learning .....	52
3.2.4 Random Forests .....	53
<b>3.3 Evaluating Models .....</b>	<b>54</b>
<b>3.4 Tools and Frameworks .....</b>	<b>55</b>
<b>4 CHALLENGES AND RELATED WORKS.....</b>	<b>57</b>
<b>4.1 HEVC Encoding Complexity Analysis .....</b>	<b>57</b>
<b>4.2 Related Works .....</b>	<b>62</b>
4.2.1 Machine Learning for Visual Applications.....	62
4.2.2 Complexity Reduction in Video Coding.....	63
4.2.2.1 Intra Prediction.....	63
4.2.2.2 Inter Prediction.....	64
4.2.2.3 Other HEVC Encoding Steps.....	65
4.2.3 Machine Learning for Video Coding Complexity Reduction.....	65
4.2.4 Complexity Management in Video Coding .....	67
4.2.5 Complexity Reduction in Video Transcoding.....	68
4.2.5.1 H.264 to HEVC Transcoding.....	69
4.2.5.2 HEVC Bitrate Transcoding .....	70
<b>4.3 Final Remarks .....</b>	<b>72</b>
<b>5 ANALYSIS OF HEVC ENCODING AND BITSTREAM INFORMATION.....</b>	<b>75</b>
<b>5.1 Feature Analysis for the HEVC Encoding Decisions.....</b>	<b>76</b>
5.1.1 CU Partitioning .....	79
5.1.2 PU Partitioning.....	85
5.1.3 RQT Partitioning.....	88
5.1.4 Reference Frame Index .....	91
5.1.5 Fractional Motion Vector .....	94

<b>5.2 Feature Analysis of the HEVC bitstream information for transcoding decisions .....</b>	<b>97</b>
5.2.1 CU-grain Analysis .....	99
5.2.2 CTU-grain Analysis .....	103
<b>5.3 Final Remarks .....</b>	<b>107</b>
<b>6 LEARNING-BASED MODE DECISION FOR HEVC ENCODING .....</b>	<b>109</b>
<b>6.1 Fast HEVC CU partitioning Decision with Support Vector Machines .....</b>	<b>110</b>
6.1.1 SVM Algorithm Analysis .....	111
6.1.2 Decision Threshold Effect on Rate-Distortion Performance .....	116
6.1.3 Results and Discussions .....	119
<b>6.2 Fast HEVC CU partitioning Decision with Decision Trees .....</b>	<b>123</b>
6.2.1 Low-Complexity HEVC Encoder .....	125
6.2.2 Decision Tree-based Complexity-Scalable HEVC Encoder.....	126
6.2.3 Results and Discussion .....	127
<b>6.3 Learning-based HEVC Mode Decision with Decision Trees.....</b>	<b>129</b>
6.3.1 PU Early Termination .....	130
6.3.2 RQT Early Termination .....	132
6.3.3 ME Early Termination .....	134
6.3.4 FME Early Termination .....	135
6.3.5 Decision Threshold effect on Decision Trees .....	137
<b>7 LEARNING-BASED MODE DECISION FOR HEVC TRANSRATING .....</b>	<b>141</b>
<b>7.1 CTU-level Partitioning Decision for Fast HEVC Transrating with Random Forests .....</b>	<b>141</b>
7.1.1 Feature Design and Processing .....	144
7.1.2 Results and Discussion .....	146
<b>7.2 Statistical- and Learning-Based Decisions for Fast HEVC Transrating .....</b>	<b>149</b>
7.2.1 PU Early Termination Heuristic.....	151
7.2.2 RQT Early Termination Heuristic.....	151
7.2.3 Prediction Mode Early Termination Heuristic .....	152
7.2.4 ME Early Termination Heuristic .....	152
7.2.5 Results and Discussion .....	154
<b>8 CONCLUSIONS AND FUTURE DIRECTIONS .....</b>	<b>159</b>
<b>8.1 Main Findings .....</b>	<b>159</b>
<b>8.2 Future Directions .....</b>	<b>160</b>
<b>REFERENCES.....</b>	<b>163</b>
<b>APPENDIX A — INPUT SEQUENCES .....</b>	<b>169</b>
<b>APPENDIX B — FEATURES AND METRICS .....</b>	<b>175</b>
<b>B.1 List of Features used in the Fast Encoding Decisions .....</b>	<b>175</b>
<b>B.2 Scores of the Features used in the Fast Encoding Decisions.....</b>	<b>177</b>
<b>B.3 List of Features used in the Fast Transcoding Decisions .....</b>	<b>191</b>
<b>B.4 Scores of the Features used in the Fast Transcoding Decisions .....</b>	<b>192</b>
<b>APPENDIX C — PAPERS SUBMITTED AND PUBLISHED DURING THE PHD .....</b>	<b>195</b>

## 1 INTRODUCTION

The rapid advances in semiconductor technologies allow an increasing number of processing cores in a single chip, motivating the industry to continuously develop new products. Naturally, this fast-paced technological race has affected the consumer market, establishing an increasing demand for more powerful electronic products and better quality media services. A practical example of this trend is observed in current mobile devices: a single smart phone can perform tasks that would require several apparatuses a few years ago, such as internet browsing, high-definition video recording/playback, GPS routing, etc.

A second side effect of these innovations is the growing use of digital video applications. Video-streaming services like YouTube and Netflix are steadily replacing the traditional means of entertainment. In addition, video conferences become more frequent as the communication technology improvements provide larger bandwidths, and stereo or multi-view videos are also increasingly common. To quantify this trend, a forecast paper published by Cisco shows that the internet-video bandwidth will go from 70% of the total traffic in 2015 to 82% by 2019 (CISCO, 2016).

The main concern that arises from this fact is that, as the use of digital video increases, so does the need for larger resolutions and higher frame rates. These two parameters, along with the number of views in multi-view sequences, are directly proportional to the bandwidth required to transmit a video, as well as to the memory capacity needed to store them. This can actually be quantified for uncompressed sequences. The equation below shows the bandwidth required to transmit an uncompressed video sequence considering a 4:2:0 color sampling:

$$BW = N_{VIEWS} \times W \times H \times FPS \times Bit_{depth} \times 1.5 bits/s \quad (1.1)$$

In (1.1),  $N_{VIEWS}$  represents the number of views (one for single-view sequences and 2 or more for multi-view),  $Bit_{depth}$ , the size of a luminance or chrominance sample in bits (usually 8), whereas  $W$ ,  $H$ , and  $FPS$  are respectively the width, height and frame rate of the sequence. Note that the 4:2:0 color subsampling is already a mechanism for compressing data, since it defines that, for each 4 luminance samples, only 2 chrominance samples (one for each layer) are required. That explains the 1.5 multiplication factor in the formula (if 4:4:4 subsampling is adopted, defining one chrominance sample for each luminance one, this factor would be 3).

For an uncompressed Full HD sequence ( $1920 \times 1080$  pixels, also called 1080p) captured at 30 frames per second, a bandwidth of 746.5 Mbps is required to transmit this video in real time. This also means that if we want to store a 60-minute Full HD sequence, it would take 336 GB to do so. These requirements are clearly prohibitive with current technology, especially if portable devices with tighter constraints like smart phones and tablets are considered. Therefore, compressing this information using video-coding techniques is imperative to enable the broad utilization of this media.

Video coding can be roughly described as a process that explores frames and regions inside frames (blocks) in search for redundant information that can somehow be compressed by exploiting such redundancies. The output of this process is a stream of bits that must be decoded whenever the video is displayed. There are several video-coding techniques available, each performing the same task through a different manner, so video-coding standards were created in order to allow a common language between encoders and decoders in different platforms.

The latest video-coding standard is commonly referred to as High Efficiency Video Coding (HEVC), registered under recommendation H.265 by ITU-T (ITU-T, 2013). HEVC emerged from a collective effort of many video-coding experts from both industrial and academic branches, known as Joint Collaborative Team on Video Technology (JCT-VC). Starting at April 2010, several JCT-VC meetings took place with the purpose of deciding whether proposed tools should be part of the codec or not.

The HEVC standard employs a block-based hybrid coding architecture, combining motion-compensated prediction and transform coding with high-efficiency entropy coding. However, in contrast to previous video coding standards, it provides a flexible quadtree coding block partitioning structure that enables the use of large and multiple sizes of coding, prediction, and transforms blocks. It also employs improved intra prediction and coding, adaptive motion parameter prediction and coding, a new loop filter and an enhanced version of context-adaptive binary arithmetic coding (CABAC) entropy coding (KIM et al., 2013). New high-level structures for parallel processing are also employed. A more detailed discussion on this standard is presented in Chapter 2. As seen in Table 1.1, despite the similarities with its predecessor, the data structures and the techniques defined in HEVC are more sophisticated than those of H.264/AVC (ITU-T, 2003).

Table 1.1: Coding tools used in the H.264/AVC and HEVC standards

Tool	H.264/AVC	HEVC
Data structures	Macroblock, block	CTU, CU, PU, TU
Intra-prediction	9 modes	35 modes
Inter-prediction	4 partitions	8 partitions
Transforms	2 modes	8 modes (RQT)
Filters	DF	SAO, DF

## 1.1 Problem Formulation

HEVC outperforms H.264/AVC in terms of coding efficiency by 39.3% on average for the same image quality (using the Bjøntegaard Difference metric – BD-Bitrate or simply BD-BR as the efficiency metric) (GROIS et al., 2013), (BJONTEGAARD, 2001).

The aforementioned innovations are key components that enable such coding improvements, but they also convey an issue that has been subject of research for many years: the computational requirements of real HEVC encoding applications. In fact, as stated in (VANNE et al., 2012), the HEVC encoder requires from 20% to 50% more computations to compress data compared to H.264.

This will apparently remain a problem when the next-generation encoder emerges. The same group of experts that developed HEVC is now working on a project called Joint Video Exploration Team on Future Video coding (JVET). The latest reports from the current implementation of the JVET encoder show that it is 11.3 times more complex than HEVC for a 25% BD-BR gain (KARCZEWICZ; ALSHINA, 2016).

Practical applications introduce many factors that cannot be ignored by complex systems such as video encoders. For instance, if a camera needs to record and instantly transmit a video (such as live transmissions), then real-time encoding is needed, which means that more than 20 frames per second (fps) must be encoded (the required throughput is usually 30 fps or greater). This is further aggravated if the resources used by the encoder are shared among other components of the device, meaning that a variable amount of computation is available at each time interval. Furthermore, if the device is battery-powered, the remaining battery life may be just enough to perform a very low-complexity encoding task.

The reference software for HEVC contains every tool defined in the JCT-VC work, and it is referred as HEVC Model (HM) (KIM et al., 2013). The HM code was designed to be the reference HEVC software and is not a suitable option for practical encoder implementations, mainly because its goal is to support and to document every tool defined

in the standard. In other words, the HEVC Model disregards computational costs or energy constraints typical of real-world platforms. Fast encoder implementations such as x265 (for HEVC) (MULTICOREWARE, 2017) and VP9 (MUKHERJEE et al., 2013) reduce this setback, but real-time throughput is still limited to specific cases and platforms, so even these implementations can benefit from reduced computation requirements. In addition, a variable amount of energy (in battery-powered platforms), computation, and memory resources are available at each time interval for video encoding. To ensure that the best output quality is obtained based on the current amount of resources available, adaptation techniques must be employed.

All these facts combined call for adjustments that somehow reduce encoding complexity. One possible way to solve this is to design a low-complexity encoder that reduces a fixed amount of computation compared to the traditional approach. However, this implies that the same process will be carried out even when there are more resources available to achieve a better compression. Alternatively, an encoding system that is capable of adapting itself according to the available processing budget is a more promising solution.

A plethora of solutions that tackle video-coding complexity issues can be found in the literature, and the most relevant ones are discussed in Chapter 4. These solutions can be separated into two categories: the ones based in complexity reduction, and the ones targeting complexity scaling. Complexity reduction in video coding fundamentally consists in replacing a component of the system by one that is faster and does a similar job. This is done by designing new lightweight algorithms and faster heuristics. In contrast, complexity scaling relies on building solutions with scalable levels of complexity to achieve different computation targets. This takes advantage of the fact that the optimization space of the RDO mode decision is enormous and can be reduced in numerous ways. Both approaches are valid, but complexity reduction has been exhaustively studied for years, and most of them rely on heuristics built from statistical analysis, but the generalization property of learning-based models suggest that there is room for improvement. Additionally, the lack of adaptability from such solutions limits its applications, whereas complexity scaling enables a set of options to best address the computing constraints of each system. Lastly, a limited number of works focusing on complexity scalability are found in the current literature, indicating that significant potential innovations can be sought after.

Solutions for complexity reduction of HEVC-related transcoding, also presented in Chapter 4, have proven that the transcoding process can be guided using relevant information from a pre-encoded bitstream. The works that address this problem usually focus

on two applications: when the source bitstream is encoded with another standard and must be re-encoded in HEVC, which is called heterogeneous transcoding; and homogeneous transcoding, when both source and output bit streams are both encoded in HEVC, but under different constraints, such as scaled resolution or lower bitrates. This work will focus on the particular case of homogeneous HEVC transcoding with different bitrates, as this problem is very important for adaptive streaming systems.

## 1.2 Thesis Claim

The claim defended in this thesis is that Machine Learning techniques can be used to efficiently achieve complexity reduction and scaling on video-coding and transcoding applications. To prove this concept, this work will present several solutions that are capable of reducing the enormous computational effort required by these applications through the use of learning-based techniques. In addition, every method will consider the need for complexity scalability that exists in computation-constrained environments.

Our motivation comes from the recent innovations in Machine Learning techniques that have led to a large utilization of intelligent models to solve complex problems that are especially hard to compute with traditional data structures and algorithms. In particular, the current research on Image and Video Processing shows that it is possible to design Machine Learning models that perform object recognition and even action recognition with high confidence levels. In addition, the latest progress on training algorithms for Deep Learning Neural Networks was also an important breakthrough in Machine Learning, leading to prominent discoveries in Computer Vision and other applications. Lastly, recent studies have also shown that it is possible to design intelligent models capable of drastically reducing the optimization space of mode decision in video encoders with minor losses in coding efficiency. All these facts indicate that Machine Learning for complex reduction in video coding is a very promising field of study.

The need for fast transcoding solutions is reaffirmed in a recent Call for Evidence document release at the 119th MPEG meeting (MPEG, 2017), asking for contributions that enable the transcoding of videos with reduced computational complexity compared with a full re-encode approach. The call hints the possibility of using a highest bitrate stream and an additional side stream that is used to assist the transcoding process of lower bitrate encodings.

### 1.3 Thesis Organization

This thesis is organized as follows:

Chapter 2 presents the background information on HEVC encoding, showing the most important components of this process. Afterwards, the test conditions commonly applied in the video-coding research are explained. Finally, the constant quality and constant bitrate coding modes are described, followed by a brief explanation of transcoding systems.

Chapter 3 gives an overview of Machine Learning, as well as the main techniques that are relevant to this thesis. Model evaluation and the main software tools that can be used to develop ML models are also presented.

The challenges that are involved with the complexity reduction of video coding and transcoding are uncovered in Chapter 4, followed by a study of the main references related to the topics addressed in this thesis.

Chapter 5 provides an extensive analysis of the HEVC encoding partitioning decisions, identifying relationships between these decisions and the information that can be obtained from the encoding process (useful for fast encoding decisions) or from a pre-encoded bitstream (applicable in fast transcoding).

The fast encoding and transcoding solutions developed in this thesis are explained in Chapters 6 and 7, presenting extensive experimental results for the main methods and comparisons with state-of-the-art solutions. The complexity scalability mechanisms of each approach is also discussed.

Finally, Chapter 8 closes this thesis with the conclusions and future directions of this work.

Appendix A lists and describes all the sequences used in the analyses and performance assessments of this work.

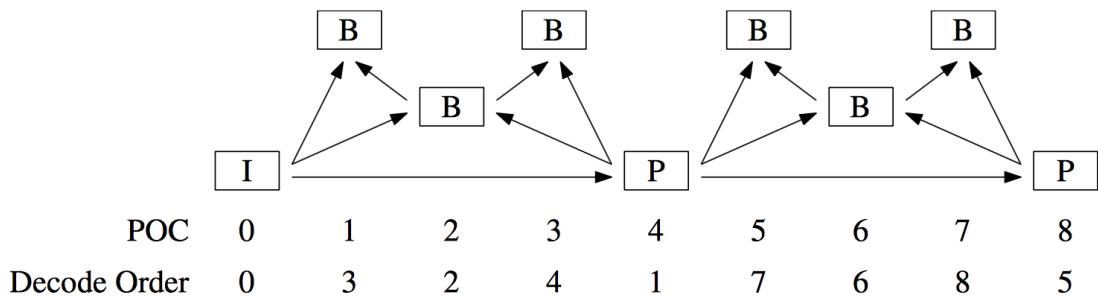
Appendix B describes the features extracted at the encoding and the transcoding data collection steps, as well as the scores that measure the utility of these features in the partitioning decisions.

Appendix C presents the publications that were produced from this research, which include one book chapter, one published journal paper (plus one under consideration), one published conference paper, two accepted conference papers, and one submitted conference paper.

## 2 HIGH EFFICIENCY VIDEO CODING

Despite its many innovations, which will be duly explained in this chapter, HEVC encoding shares many similarities with H.264/AVC. Prior to encoding, a sequence is separated into Groups of Pictures (GOPs). A GOP defines how the Intra (I) and P/B frames are arranged, as well as the QP values and references that will be used for each frame in the GOP. Only intra-prediction is possible in I frames, whereas P/B frames support both intra- and inter-prediction. Periodically, special I frames called Instant Decoder Refresh (IDR) are also defined in the encoding configuration. These frames establish that preceding frames cannot be used as reference in subsequent encodings. This allows seeking back and forth in a video stream using the IDR frames as anchors. Figure 2.1 shows a GOP example using an IBBBP structure.

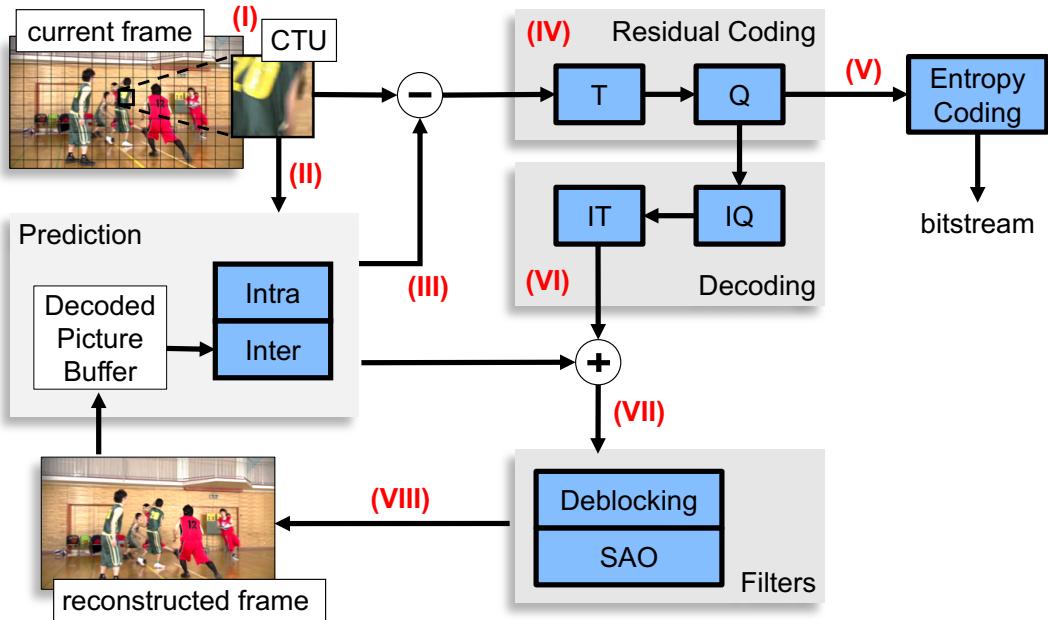
Figure 2.1: Group of Pictures (GOP) example using I, P, and B frames. (source: (BOSSEN; FLYNN; SÜHRING, 2013))



After the GOP is defined, each frame is partitioned into blocks – named Coding Tree Unit (CTU) – that undergo the encoding loop, as depicted in Figure 2.2.

The process can be roughly described as follows: (I) retrieve the next CTU from the input frame. In HEVC encoders, a partitioning decision is performed, in which the CTU is subdivided into smaller blocks, and the following steps (II to VIII) are repeated for each of these blocks. This was omitted in Figure 2.2 for visual purposes, but it will be explained in Section 2.1. Each block is sent to the (II) prediction stage in which its redundancies are exploited. (III) The predicted block is then subtracted from the current CTU, producing a residue block (this can be interpreted as an error). (IV) The residual information serves as input for the transforms (T) and quantization (Q) stages, in which the information with lower energy is discarded (the threshold for this elimination is computed based on the Quantization Parameter – QP). (V) The quantized output is sent to the entropy encoder, generating the final bit stream that can be either stored or transmitted. Next, (VI) the quantization output is also sent to a decoding stage, after which the residue

Figure 2.2: HEVC encoding loop and the main tasks involved.



is reconstructed and added to the predicted samples yet again. (VII) The resulting block is then filtered to remove visual artifacts introduced by the encoding process (due to its block-based operation). Finally, (VIII) this information is stored as a reconstructed frame, which will be used as a reference in the next encoding iterations.

HEVC introduced more complex data structures. This standard defines that each CTU can be encoded in a variety of distinct ways with the possibility to recursively subdivide themselves into four smaller Coding Units (CUs) to allow more flexibility, forming a quadtree structure. Each CU can be further subdivided into Prediction Units (PUs) during prediction and into Transforms Units (TUs) at the transforms stage. This flexibility was designed to achieve the best possible relation between the compression efficiency and quality, represented by a metric called Rate-Distortion cost. Each new subdivision introduces significant amount of computations to determine the best encoding mode.

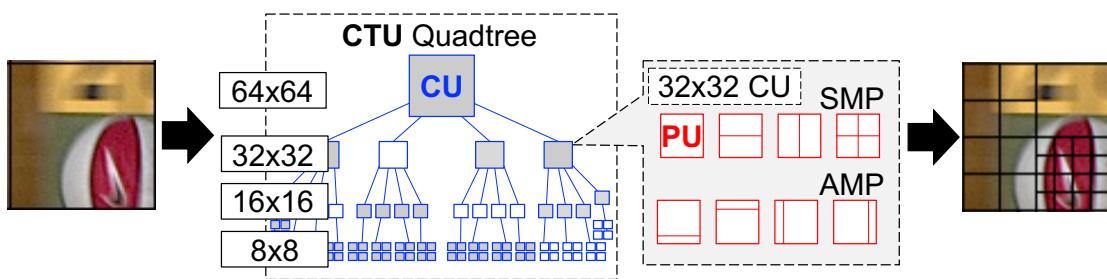
Aside from the CTU structure and its subdivisions, other novel techniques defined in HEVC also contribute to its coding efficiency, such as improved entropy with CABAC and Rate-Distortion Optimized Quantization (RDOQ). These tools are all implemented in the HEVC Model (KIM et al., 2013), a reference software developed by JCT-VC experts. The HM implements every tool and technique supported by HEVC (although not all of them are enabled under default configurations), so it is widely adopted as a test bed in the video coding community. The next sections describe the most important tools and coding modes supported in this standard. Following, the algorithm used to decide the best mode among the many possibilities available is explained.

## 2.1 Prediction in the HEVC standard

The current HEVC version defines that a frame must be subdivided into CTUs, which are broadly analogous to Macroblocks in the H.264/AVC standard (BROSS, HAN, et al., 2011). The largest CTU size currently supported is  $64 \times 64$  luminance samples, although this may increase in the near future (the current version of JVET supports  $128 \times 128$  blocks (CHEN, ALSHINA, et al., 2017)). As with Macroblocks, the chrominance samples are also part of the CTU. This size is fixed throughout and must be set prior to encoding a sequence.

To provide a more flexible coding structure, each CTU can be divided into Coding Units. The reason behind this is to separate distinct regions of a frame with similar redundancies that can be exploited by the prediction process. The process occurs recursively, so a  $64 \times 64$  CTU is divided into four  $32 \times 32$  CTUs, and each of these can be subdivided into four  $16 \times 16$  CTUs. This is repeated until the smallest CTU size is reached (currently defined as  $8 \times 8$ ), or when a terminating condition is satisfied (since the HM reference implements heuristics in its mode decision), implicitly forming a quadtree. Figure 2.3 shows an example of a quadtree partitioning.

Figure 2.3: Example of a CTU and its respective CU and PU partitioning decisions. The gray-filled blocks are evaluated, but not used in the final encoding.



During prediction, CUs are again partitioned into Prediction Units (PUs), which store the information related to this process, such as prediction mode (inter, intra), motion vectors, reference frames indices etc. Four Symmetric Motion Partitions (SMPs) and four Asymmetric Motion Partitions (AMPs) are defined. The SMPs are depicted as the top four partitions in Figure 2.2, whereas the AMPs are the bottom four ones. For each PU, the prediction process is executed, which encompasses the Integer (IME) and Fractional Motion Estimation (FME) searches. The PU partitions available for a CU depend on its size and its prediction mode. Table 2.1 displays all the enabled partitions for each prediction mode.

Table 2.1: Enabled/Disabled partitions for each prediction mode and each CU size

Mode	CU	$2N \times 2N$	$2N \times N$	$N \times 2N$	$N \times N$	$2N \times nU$	$2N \times nD$	$nL \times 2N$	$nR \times 2N$
Intra	$64 \times 64$	E	D	D	D*	D	D	D	D
	$32 \times 32$	E	D	D	D*	D	D	D	D
	$16 \times 16$	E	D	D	D*	D	D	D	D
	$8 \times 8$	E	D	D	E	D	D	D	D
Inter	$64 \times 64$	E	E	E	D*	E	E	E	E
	$32 \times 32$	E	E	E	D*	E	E	E	E
	$16 \times 16$	E	E	E	D*	E	E	E	E
	$8 \times 8$	E	E	E	D	D	D	D	D

\* Only available if the minimum CU size is greater than  $8 \times 8$

In Table 2.1,  $N$  represents half of the CU dimension, i.e., a  $2N \times N$  PU in a  $64 \times 64$  CU stands for a  $64 \times 32$  partition. In addition,  $n$  represents a quarter of the CU dimension, so a  $2N \times nU$  PU is actually a  $64 \times 16$ , whereas the complementary partition must be dimensioned as  $64 \times 48$ . Note that  $N \times N$  PUs are available for inter-prediction only if the minimum CU size is other than  $8 \times 8$ , as opposed to the default HM implementation. The experts behind this software stated that  $N \times N$  was disabled in  $8 \times 8$  CUs to reduce bandwidth requirements (KIM et al., 2013).

The following sections describe the intra- and inter-prediction processes, granting more focus to the latter due to its dominant share in the overall computational effort.

## 2.2 Intra Prediction

This prediction mode is responsible for compressing information by exploiting redundant information in a frame, such as homogeneous regions or repeating patterns. This type of prediction is used in notorious image compressing standards, such as Portable Network Graphics (PNG) (ISO/IEC, 2004). The abstract notion behind the intra prediction is simple: use the borders of neighboring blocks to predict the pixels of the current one.

HEVC defines 35 ways to predict the information. 33 of them, called directional modes, are useful when there are similarities in the texture direction of the luminance samples, such as diagonal or straight lines. The other two are called DC mode, which predicts that the average sample is repeated throughout the block; and planar mode, that repeats information from more than one border, useful in gradient patterns. For  $64 \times 64$  PUs, only 5 modes are evaluated, and on  $4 \times 4$  PUs, 17 modes are considered.

In order to minimize computations, a heuristic called Rough Mode Decision (RMD) is implemented in the HM software. In RMD, only a subset of the 33 directional modes

available is evaluated, instead of testing all of them. After the best direction is selected, a refinement step is applied, testing the neighboring directions.

### 2.3 Inter Prediction

The inter-prediction is based on exploiting redundant information between two or more frames in a sequence. High frame rates and absence of movement are two typical situations that raise the odds of achieving good compression through inter-prediction.

During inter-prediction, HEVC defines that a PU can be coded under three modes: inter, merge, SKIP. Each will be described in the following paragraphs.

- **Inter:** when a PU is inter-coded, its motion information is derived from the Motion Estimation (ME) process. The ME implements a block-matching algorithm that follows a given search pattern. The main goal is to find the most similar block to the one being encoded in a reference frame (a previously encoded frame), and reuse this information to provide compression. Different ME algorithms were proposed throughout the years, each with a different search pattern or different starting point. The starting point for the search can be, for instance, the collocated position in the reference frame, or the same as motion vector of the surrounding PUs. Two ME algorithms are implemented in the HM software: Full Search and Test Zone Search (TZ-Search), the latter being a fast algorithm that reduces ME complexity with minimum RD penalties (PAN et al., 2013).
- **Skip:** This mode significantly contributes to compression gains, because no movement or residual information is transmitted. In this mode, the PU partition is invariably square ( $8 \times 8$  up to  $64 \times 64$ , represented always as a  $2N \times 2N$  PU). In inter-coded CUs, the motion information is derived from spatial and temporal neighbors.
- **Merge:** consists in deriving the motion parameters for a PU based on the information obtained from spatial and temporal neighbors. This provides significant compression gains, because only the merge flag and the PU index are sent to the bitstream; the motion information is inferred from the pointed PU. The merge mode is used for SKIP-coded PUs (when the residue is not included in the bitstream), but can also be used for inter-coded ones (including the residual information).

Inter-predicted frames also support two other modes: intra, and Pulse Code Modulation (PCM).

The ME process, performed in inter-coded PUs, is responsible for most of the encoding computational effort. This process is divided into Integer ME (IME) and Fractional ME (FME). During IME, the block-matching algorithm uses one or more reference frames to look for the most similar match. The same process is applied in the FME, but blocks with fractional pixels are also computed. Since the fractional pixels are not originally represented in the sequence, they must be interpolated using 4-, 7- or 8-tap filters, further increasing the computational effort of this process.

The FME introduces motion vectors that represent movements smaller than a pixel, which increases the precision of Motion Estimation. In HEVC, half-pixel, quarter-pixel and 1/8-pixel (chrominance pixels only) precisions are supported. Half-pixels of luminance are interpolated using an 8-tap filter, and quarter-pixels with a 7-tap filter. Chrominance pixels are always interpolated with 4-tap filters. As opposed to H.264/AVC, the fractional part of ME in the HM software cannot be disabled by any encoding parameter; so fractional vectors are always expected in the bitstream.

As pointed in (VANNE, VIITANEN, et al., 2012), IME and FME share 17% and 54% of the overall coding computational effort, adding up to 71%. This shows that the ME computations pose a serious concern, so it must be prioritized in complexity solutions that aim to achieve significant reductions.

## 2.4 Transforms and Quantization

The residue produced between the current CTU and the one predicted is sent to a process that can be referred to as residual coding. During this process, the transforms, quantization, and their inverse counterparts are applied, thus it is also called T/Q/IQ/IT loop.

The transforms are responsible for translating pixel values to the frequency domain. This process also compacts the energy in the upper-left side of the block (low frequencies coefficients), which is useful for quantization and entropy that follow it. Like in prediction, the HEVC transforms also have different modes. The data structure used in this step is called TU, and it can assume sizes of  $4 \times 4$  up to  $32 \times 32$  samples, introducing a mode decision tree, called Residual Quadtree (RQT), that is similar to the CTU one. Note that one RQT is nested in each CU of the CTU quadtree, aggravating the computational costs of the latter. In the HM implementation, the transform is performed by partial butterfly structure for low computational complexity (KIM, MCCANN, et al., 2013).

Next in the residual coding process is the quantization. With the transformed coefficients of a block given as input, it is possible to use Human Visual System (HVS) theories to discard the frequencies that are less relevant to our vision. The threshold used decides the frequencies ranges that will be attenuated, and is calculated based on a very important parameter called Quantization Parameter (QP). Higher QP values increase the frequencies discarded, causing significant compression gains (the explanation is presented in the next section), but this also increases the quality losses, since frequencies that are actually relevant to our vision get discarded as well. The opposite occurs for lower QPs. The best QP in terms of rate-distortion depends on the characteristics of each region in the frame. Therefore, a technique called Rate-Distortion Optimized Quantization can be employed. The basic concept behind the RDOQ is to use different QP values for each coefficient, given both its impact on the bitrate and quality. To estimate the impact of a coefficient in the number of bits, the tabulated values stored in CABAC (employed in the entropy coding) are used (KIM, MCCANN, et al., 2013). A simpler alternative to RDOQ is to perform the Quantization using a fix QP in every CTU of the frame. Both modes are supported in the HEVC encoders.

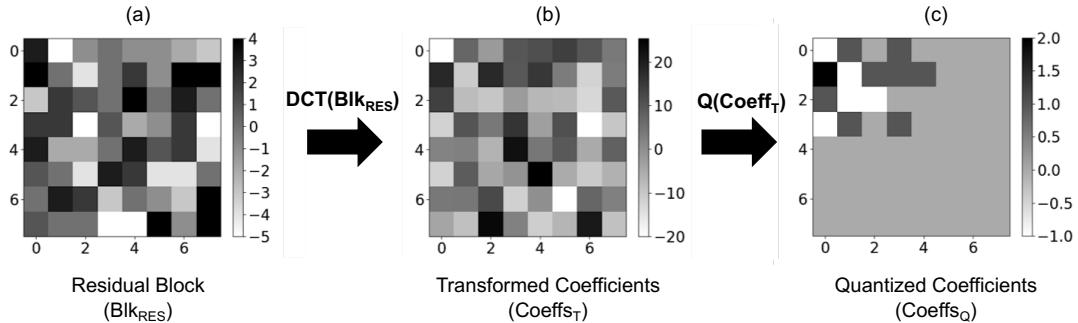
## 2.5 Entropy Coding

This is the step where the compressed bitstream is generated. The entropy coding reduces statistical redundancies by applying data compression techniques similar to the ones used in famous algorithms such as ZIP. The entropy of a data set (in this context, a block of pixels) is measured based on how predictable each data point (or pixel) is. This explains why transforms and quantization are so important for compressing visual information. The transforms energy-compaction property combined with the quantization discarding produce low entropy regions in the lower-right region of a Macroblock. This is better illustrated in Figure 2.4, which shows a residual block ( $\text{Blk}_{\text{RES}}$ ), its Discrete Cosine Transform (DCT) result and then its quantized output.

Note that the residue was spread in the blocks before and after the DCT, but only a few non-zero samples appear after quantization, and they are compacted in the upper-left corner of the block.

In HEVC, a lossless compression technique based on arithmetic coding named CABAC is used. This algorithm implements efficient compression based on statistical analysis, which is updated online, granting adaptability to the video characteristics.

Figure 2.4: Example of a residual block (a), the DCT coefficients of this block (b), and the coefficients after quantization (c).



## 2.6 Rate-Distortion Optimization

Based on what was explained so far, it is possible to conclude that there are many combinations possible to encode a single CTU: different prediction modes, partition sizes, transforms sizes, quantization parameters etc. In order to decide which one is best, a decision called Rate-Distortion Optimization (RDO) is used.

As the name suggests, this decision is based on optimizing the Rate-Distortion Cost, which is calculated as follows:

$$RD_{cost} = Distortion + \lambda_{mode} * B_{mode} \quad (2.1)$$

In (2.1),  $\lambda$  and  $B$  are respectively a Lagrange multiplier and the cost in bits of a particular mode (e.g., the bit cost to encode a  $64 \times 64$  CU using SKIP mode). The Lagrange multipliers for each mode are hardcoded in the reference software (KIM, MCCANN, et al., 2013). The distortion corresponds to the SAD (Sum of Absolute Differences), SATD (Sum of Absolute Transformed Differences), or SSE (Sum of Squared Error) result using the residual block (current block minus predicted block).

Note that, to obtain the value of  $B_{mode}$ , it is necessary to go through the transforms, quantization and entropy steps. This introduces a great computation overhead, especially during prediction, because the RD cost of each CU must be calculated during the CTU quadtree decision, so transforms and quantization are called many times just to decide the best prediction mode. The same occurs during RQT decision, in which entropy must also be performed to decide the best TU size. Therefore, reducing the number of nodes in both, CTU and RQT quadtrees is an important way of reducing the overall complexity.

## 2.7 Common Test Conditions

Video encoders have several applications, from broadcasting to videoconferencing on smart phones, thus video-coding standards must provide a wide range of configuration sets in order to support them all. Nonetheless, this can become a problem when it is necessary to compare two different implementations, since it is important that the same configuration was used in both cases to maintain fair comparisons.

With that in mind, the JCT-VC made an effort on creating a document that describes the Common Test Conditions (CTC) for the HEVC Model reference (BOSSEN, 2011). This document establishes several conditions, for instance how the reference frames should be distributed in a GOP. Three distributions are defined: All Intra (AI), Random Access (RA) and Low Delay (LB, the B stands for B frames). The RA and LB distributions, often referred to as temporal structures, are depicted in Figure 2.5(a) and Figure 2.5(b). All Intra was omitted as it simply consists of encoding every frame using only the intra prediction.

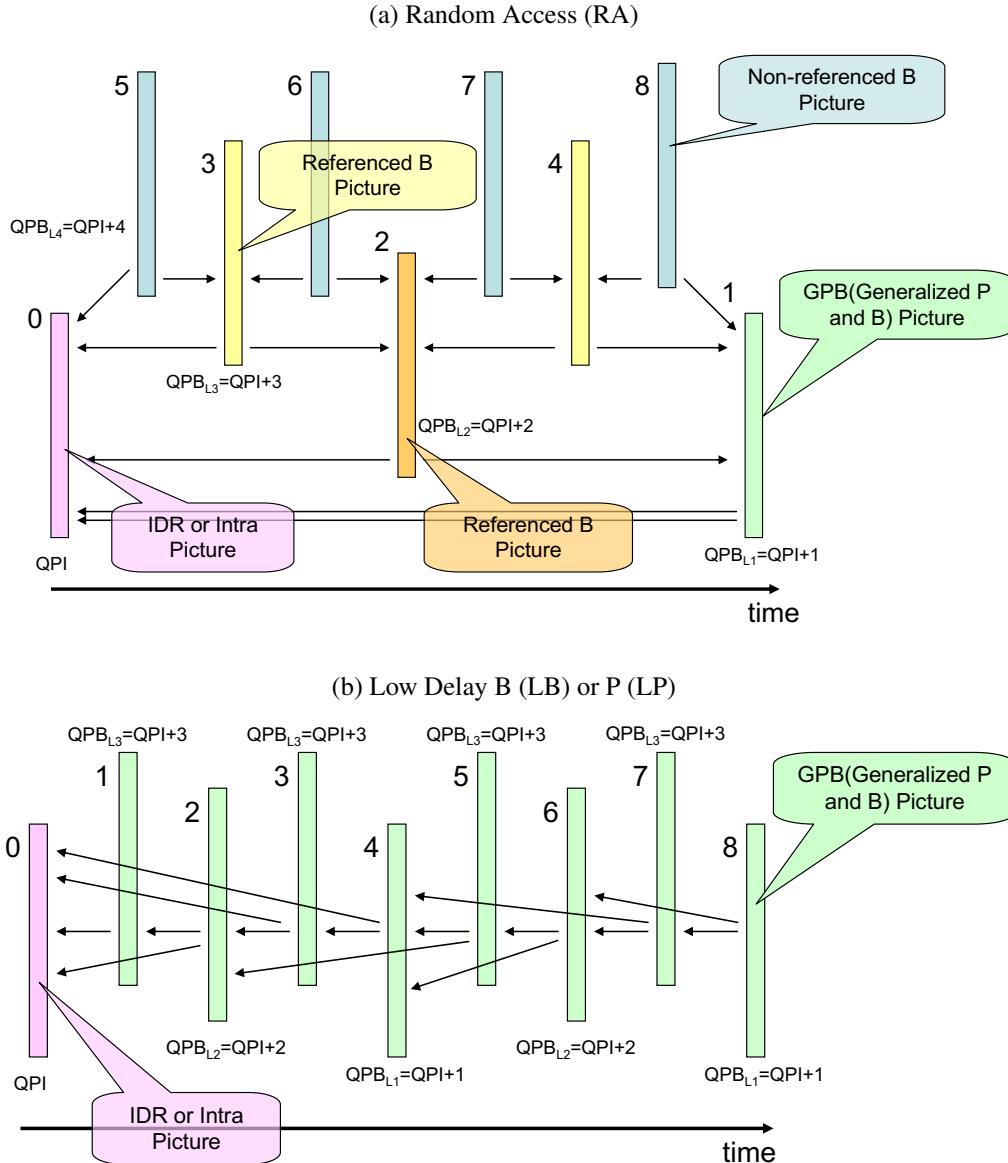
The CTC document also defines six classes (from A to F) of sequences that should be used when running tests on HEVC. They are classified according to their resolution and target application. For instance, classes A to E refer to broadcasting applications, whereas class F targets user-generated content, such as video-conferences and gaming.

## 2.8 Metric for Assessing Coding Performance

The metrics employed to measure encoding efficiency are also discussed in the CTC. This document encourages the use of metrics based on the Bjøntegaard Difference (BD) (BJONTEGAARD, 2008). These metrics are called BD-bitrate (BD-BR), measured in percentage, and BD-Peak Signal-to-Noise Ratio (BD-PSNR), measured in dB. To calculate them, the following steps must be executed:

1. The sequences are encoded under two target configurations: one considered as reference, the other being the object of comparison. For each target, four QP values (22, 27, 32, and 37, as established in the CTC) are used, producing eight (bitrate, PSNR) pairs (four from each target).
2. A piece-wise cubic Hermite interpolator is then used to generate two Rate-Distortion (RD) curves, given four (bitrate, PSNR) points for each curve. These curves will

Figure 2.5: Random Access and Low Delay temporal configurations used in HEVC encoders. (source: (KIM et al., 2013))



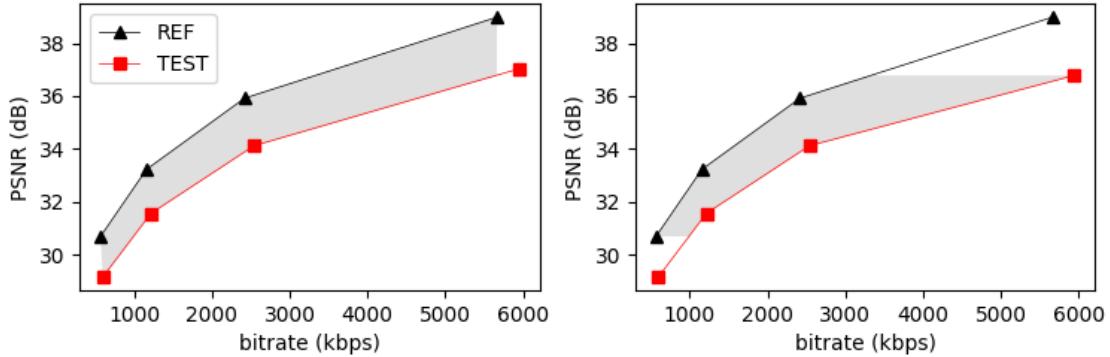
be referred to as REF and TEST, representing the reference HEVC encoder and a modified version respectively.

3. The differential area between both curves is integrated, using the X-axis as reference for BD-BR and the Y-axis for BD-PSNR. The final value is obtained from dividing the integral by its integration interval. For understanding purposes, the BR-BR formula is displayed in the equation below:

$$BD_{BR} = \frac{\int_a^b (REF_{PSNR}(x) - TEST_{PSNR}(x))dx}{b - a} \quad (2.2)$$

In (3),  $REF_{PSNR}(x)$  and  $TEST_{PSNR}(x)$  respectively represent the PSNR values obtained with the reference and test encoders, and  $a$  and  $b$  are the second minimum and second maximum PSNR values from both curves. The BD-PSNR formula is analogous, but the bit rate values are used instead. This is better illustrated in Figure 2.6 (consider each mark as the results for QPs 22, 27, 32 and 37, from right to left).

Figure 2.6: Rate-distortion plots between a reference (REF) and a test (TEST) encoder and the respective BD-PSNR (a) and BD-BR (b) areas shaded in gray.



The BD-BR metric can be interpreted as how lower/higher is the bitrate of the test subject compared to its reference considering the same quality. Thus, most of the works found divulge only BD-BR results, given that this value is sufficient to determine how one target compares against the other (BJONTEGAARD, 2008).

For solutions that deal with complexity, time is also an important metric, so this work will make use of a ratio between BD-BR and time savings, defined as:

$$BDTS_{ratio} = \frac{BD_{BR}}{TS} = \frac{BD_{BR}}{1 - \frac{T_{test}}{T_{ref}}} \quad (2.3)$$

The  $BDTS_{ratio}$  can be interpreted as the amount of BD-BR loss required to achieve a 1% time savings using the proposed method.

## 2.9 Variable Bitrate and Average Bitrate Coding

Regardless of the application, quality and bitrate must always be optimized, but in some cases it is preferable to sacrifice quality or even compression to generate a stream with a stable bitrate. When the communication bandwidth is not an issue, it is better to maintain the image quality at similar levels throughout the whole sequence even though some scenes will require more bitrate to be transmitted than others. On the other hand, live transmission is a typical case where the user experience is severely affected by frame

drops caused by bitrate peaks in the incoming stream. In the first scenario, Variable Bitrate (VBR) encoding is usually applied. In this mode, the same QP (or very small variations) is used during encoding to maintain a sense of constant quality. The second situation introduces a harder constraint on the bitrate, so it is necessary to employ a Constant Bitrate (CBR) or Average Bitrate (ABR) technique.

In CBR encodings, the bitrate will remain close to the assigned target throughout the entire video stream. This means that the same amount of bits will be spent for simple and complex scenes. In contrast, ABR does not promise a constant bitrate for every part of the video stream, introducing the possibility of adapting the target bitrate to the scene characteristics. However, predicting the best bitrate distribution while moving the average close to the target is very difficult. To overcome this problem, rate control algorithms have been developed with the aim of achieving the best possible quality under a target bitrate.

Rate control algorithms usually follow two steps: (1) allocate a suitable number of bits to each coding hierarchy, such as GOPs, pictures, and (possibly) coding units; and (2) modify the encoding parameters so that the assigned bits of each level are satisfied. As mentioned in Section 2.4, the QP is used to tune the trade-off between quality and compression efficiency, so this parameter is usually used in the second step.

In the HEVC model, the rate-control algorithm follows a  $\lambda$ -domain control, which defines a relationship between the target bitrate and the Lagrange multiplier  $\lambda$  (LI et al., 2014). The  $\lambda$ -Rate model can be updated for each frame or for each CTU with minor adaptations. After computing  $\lambda$ , the model parameters are adjusted dynamically using the actual bits spent to encode the CTU or frame. Afterwards, the algorithm adjusts the QP with a technique proposed in (LI et al., 2013), using the computed  $\lambda$  as input.

## 2.10 HEVC Transcoding

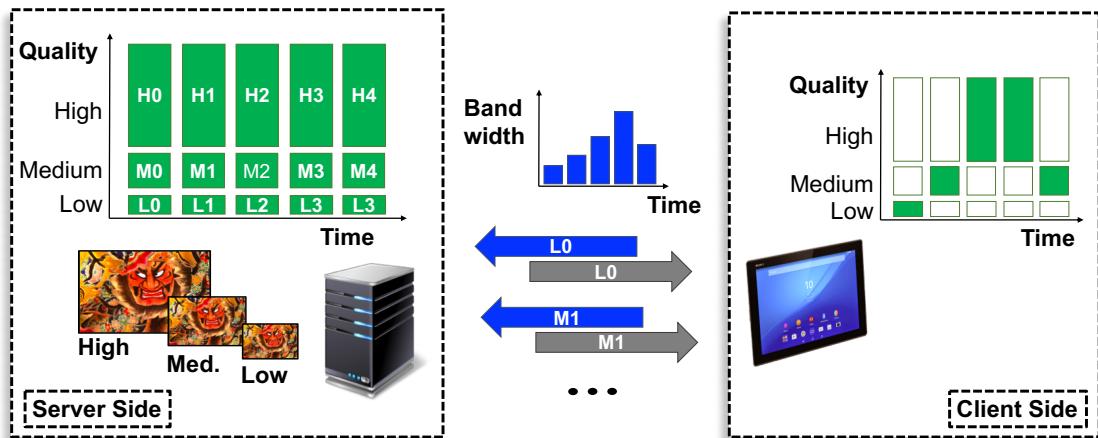
In the recent years, the advances in telecommunication technologies are steadily paving the way for a world that is fully connected to the Internet. The ubiquity of this service has revolutionized not only the way information and knowledge is transferred, but also the means of entertainment. Nowadays, Video On Demand (VOD) services are growing in popularity, replacing the traditional television shows. Companies like Netflix (NETFLIX, 2018) and Amazon (AMAZON, 2018) took advantage of this trend and are now counting with millions of user subscriptions worldwide, and this number is expected to increase in the upcoming years. According to (STATISTA, 2017), Netflix has closed

the year of 2016 with more than 90 million subscribers worldwide, 49% of which in the United States alone.

A typical challenge faced by streaming services is that they have to transmit videos in accordance with the capabilities of each client. More specifically, the server has to ensure two things: (1) the streams are encoded following a standard that can be decoded on the client device; and (2) the bitrate required to transmit the stream takes into account the network bandwidth available on the client side. In a realistic scenario, different devices request the same content to the server, each with its own network capabilities, forming a heterogeneous environment.

Protocols based on adaptive streaming have emerged to address the heterogeneity of end-user device capabilities and network bandwidth variations in real-time and on demand video streaming. Two main examples are the HTTP Live Streaming (HLS) (PANTOS, 2017), developed by Apple, and the Dynamic Adaptive Streaming over HTTP (DASH), created by the Motion Picture Experts Group (MPEG) (ISO/IEC, 2014a). One of the advantages of using DASH is that it is codec agnostic and can therefore deliver streams from any type of codec. The general idea is that the streaming server has multiple bitrate and resolution versions (representations) of the contents, which are served to the users by choosing the highest quality version that the network and the rendering device are able to transmit and process. Figure 2.7 shows a typical client-server interaction in a DASH-based system.

Figure 2.7: DASH framework illustrating the interactions between the server (on the left side) and the client (right side) in an adaptive streaming system. (adapted from (BITMOVIN, 2018))



The server is responsible for dividing the sequence into independent segments and storing representations with different quality levels (thus different bitrate ranges). In ad-

dition, the server must provide a MPEG-DASH Media Presentation Description (MPD) (ISO/IEC, 2014b), an XML document that details how the segments are organized and their URLs. The control of which segments must be requested at each time is implemented on the client side, taking into account, for instance, a network bandwidth model to estimate how much bitrate can be received in the next time steps.

The most straightforward way of producing different representations for all segments of a video is to simply encode them all and store the resulting bitstreams, which is referred to as simulcast coding (WALLENDAEL; COCK; WALLE, 2012). This is considered a low complexity solution, because encoding can be performed without timing constraints before actually storing the bitstreams. However, this approach has two major drawbacks: (1) encoded data volume increases with the number of representations, creating data storage constraints especially if several representations are demanded; (2) some of the stored content can be a dead weight since some representations might never be requested or only very rarely.

A second solution would be employing a scalable compression technique, like H.264/AVC Scalable Video Coding (SVC) extension (SCHWARZ; MARPE; WIEGAND, 2007). An SVC stream is composed of a Base Layer and one or more Enhancement Layers that can be used to increase spatial/temporal resolutions or to improve image quality. This requires less space than our first suggestion, because Enhancement Layers are much less expensive than full bitstreams. However, using SVC reduces compatibility, because it requires an SVC decoder on the client side.

It is also possible to generate the bitstreams on demand using transcoding techniques. In this solution, only a high quality version of each segment is stored, and the other representations are generated upon request through transcoding. To implement a transcoder, one can simply decode the input bitstream and (re)encode the decoded video frames to meet the new requirements. This is called cascaded pixel domain transcoder (CPDT) (VETRO; CHRISTOPOULOS; SUN, 2003). The advantages of this approach include less storage requirements (only a reference bitstream must be stored per segment), and high flexibility, as it allows any type of representation to be created, so long as the server has the necessary tools. The main drawback is that CPDT transcoding is computationally complex, especially with recent complex encoders like H.264/AVC, HEVC and VP9, making real-time implementations a challenge.

## 2.11 Final Remarks

The HEVC standard defined many novel techniques, but the excessive number of configurations supported by this standard cause many computations to be performed. If complexity-constrained applications are considered, it is important to find alternative solutions that reduce the computational effort of this process. The number of modes tested is clearly one of the aspects that could be reduced, since only one is effectively used to encode the video.

All the tools and techniques implemented in this standard must work for different applications, from broadcasting to video-surveillance, each with particular requirements. With that in mind, most of these tools were designed to work under several operation modes, which are assigned with encoding parameters. It is possible exploit this characteristic to provide complexity scalability to encoders. For instance, it is possible to reduce the optimization space of the CU splitting decision to reduce amount of computations. With fewer computations, the time to encode the sequence will naturally be shorter, as shown in the preliminary results of this thesis proposal (Chapter 6).

Hence, instead of designing new algorithms for this standard, which could cause decoding incompatibility, the idea of skipping modes that will not be used to generate the final bitstream is a promising approach. In addition, as reported in the state-of-the-art survey presented in Chapter 4, the results that follow this line of thought found in the literature are very encouraging. Nevertheless, most investigations present a superficial analysis, so there are many venues open to new research in this field. This work sets out to make contributions in this direction.



### 3 MACHINE LEARNING

Machine learning is a branch of study in Artificial Intelligence that is concerned with designing algorithms capable of obtaining knowledge from observations (RUSSELL et al., 2003). In computational problems, this knowledge is represented as a model that estimates the output of a task based on some indicators. This definition is intentionally broad, because it shows that machine learning can be applied to many cases, as long as errors are allowed to some degree. The following sections will discuss all steps and techniques involved to build such models. The steps for building predictive models with machine learning techniques are quite straightforward. Given a set of examples (or samples) related to the prediction task:

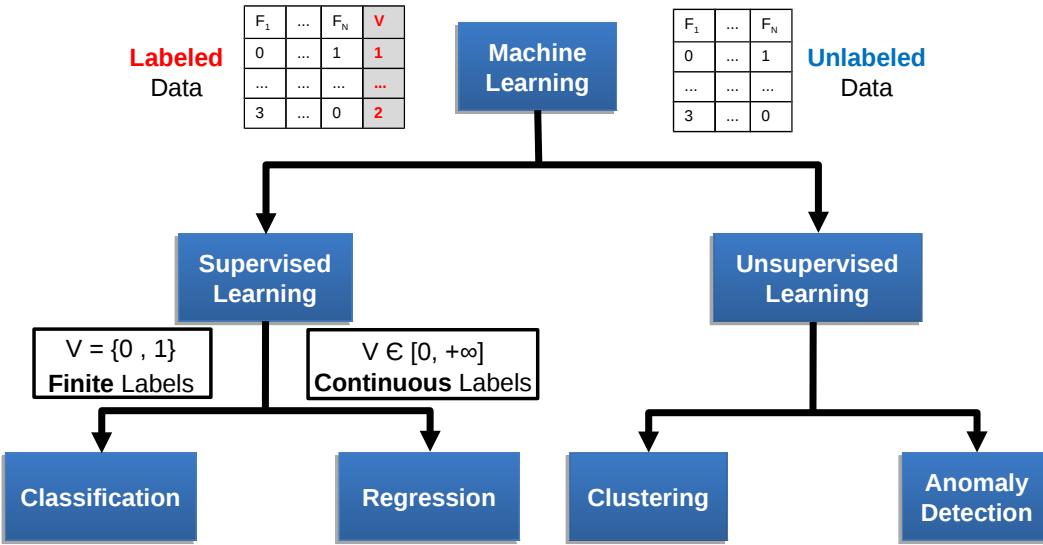
- The input samples are initially pre-processed in order to remove spurious or noisy information. This step is also applied to obtain new features from the input. For instance, an edge detection algorithm can be applied to an image, and its output can then be used as a feature to train a face-detection model.
- The pre-processed data is then used to train the model. Many techniques can be applied here, like Decision Trees or Neural Networks. Each method performs differently depending on the nature of the problem and the type of training data, so usually more than one should be analyzed.
- The generated model is tested using input samples different from the ones used during training. This phase is necessary to avoid over-fitting, a common phenomenon in machine learning that should be avoided in most cases.

The set of possible algorithms to be used depends on the kind of training (supervised, unsupervised) and desired output (classification, regression).

Unsupervised training takes place when there is no predicted outcome for samples. In this case, clustering techniques are applied to separate examples into groups and discover hidden similarities among them. In contrast, supervised training is possible when the input data contains the expected output of each case. For instance, the input for a weather forecast model can be a set of relevant measurements from previous days (temperature, air humidity, wind speed etc.), followed by a label stating whether it rained that day or not.

Supervised training can be further categorized into classification or regression. The former consists in classifying the input samples based on a group of known, discrete

Figure 3.1: Subdivisions of Machine Learning concepts.



values, also referred to as classes or labels, whereas the latter is based on predicting an expected value with several or even infinite possibilities. Figure 3.1 illustrates how these definitions are associated with one another.

In most cases, training is performed offline using the entire training set. If the model generalizes well, a good performance is expected when it is tested against an evaluation set, which must be different from the training one.

However, the task to be modeled can sometimes be too difficult to generalize. This could happen due to input variability. In some cases, one can simply not build a generic model if the input changes drastically for each case. A possible solution for this is to train models using online sampling. In this case, the process is initially executed normally for feature extraction. When enough data is gathered, the model is trained, and the execution can then operate on prediction mode.

In this thesis, we will focus on classification techniques to skip time-consuming mode decision steps during RDO optimization in video-coding applications. For simplicity, the output of each step will be limited to a binary value that signifies whether RDO should continue or not, configuring a binary classification problem.

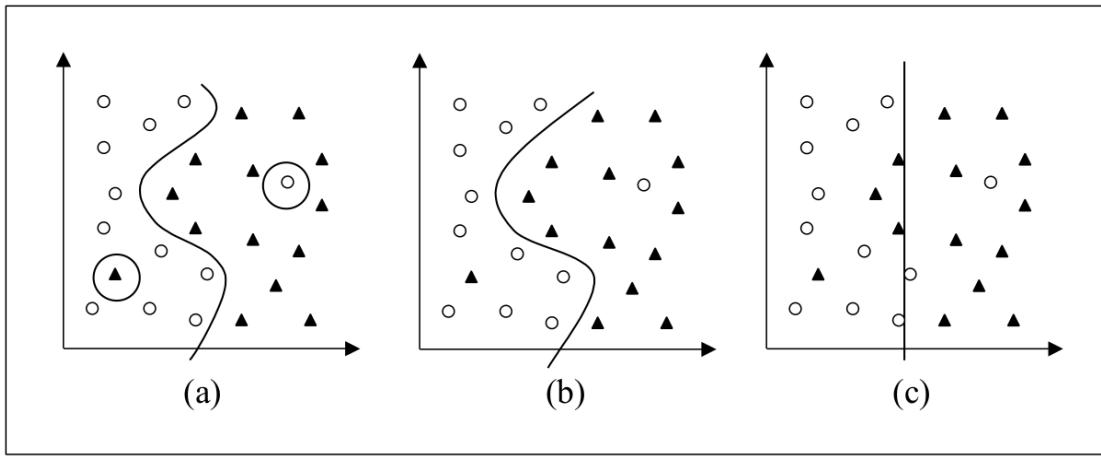
### 3.1 Binary Classification Models

Binary classification is the most studied case in classification algorithms. Binary problems are easier to interpret, easier to train, and multiclass problems can be reduced

to an ensemble of several binary problems.

The objective of a binary classifier is to formulate a hypothesis  $h$  that is capable of separating two classes given a set of variables (also called features). Figure 3.2 depicts the distribution of binary classes as a function of two variables. In Figure 3.2(c),  $h$  is a linear model, but functions of higher degree like (a) and (b) can be applied for better accuracy in some cases.

Figure 3.2: Class distribution over two labels and three different classifiers. (source: (LORENA; CARVALHO, 2007))



In Figure 3.2(a), the hypothesis is capable of separating both classes completely, while (b) and (c) had some misclassifications. However, it is possible that (a) became too specific to maximize accuracy on the training set. This is a common phenomenon called over-fitting, and should be avoided in most cases, as it reduces the generalization of classifiers.

A visual analysis of Figure 3.2(b) lets us conclude that the polynomial hypothesis gives better results for this particular training set, but this does not mean that it will better than the linear classifier (c) for future instances. This is a problem that many machine-learning academics faced when deciding what kind of classifier is better. A possible approach is to apply the Ockham's Razor, a principle named after the philosopher William of Ockham, stating that the simpler solution should always be prioritized.

### 3.2 Algorithms for Building Classification Models

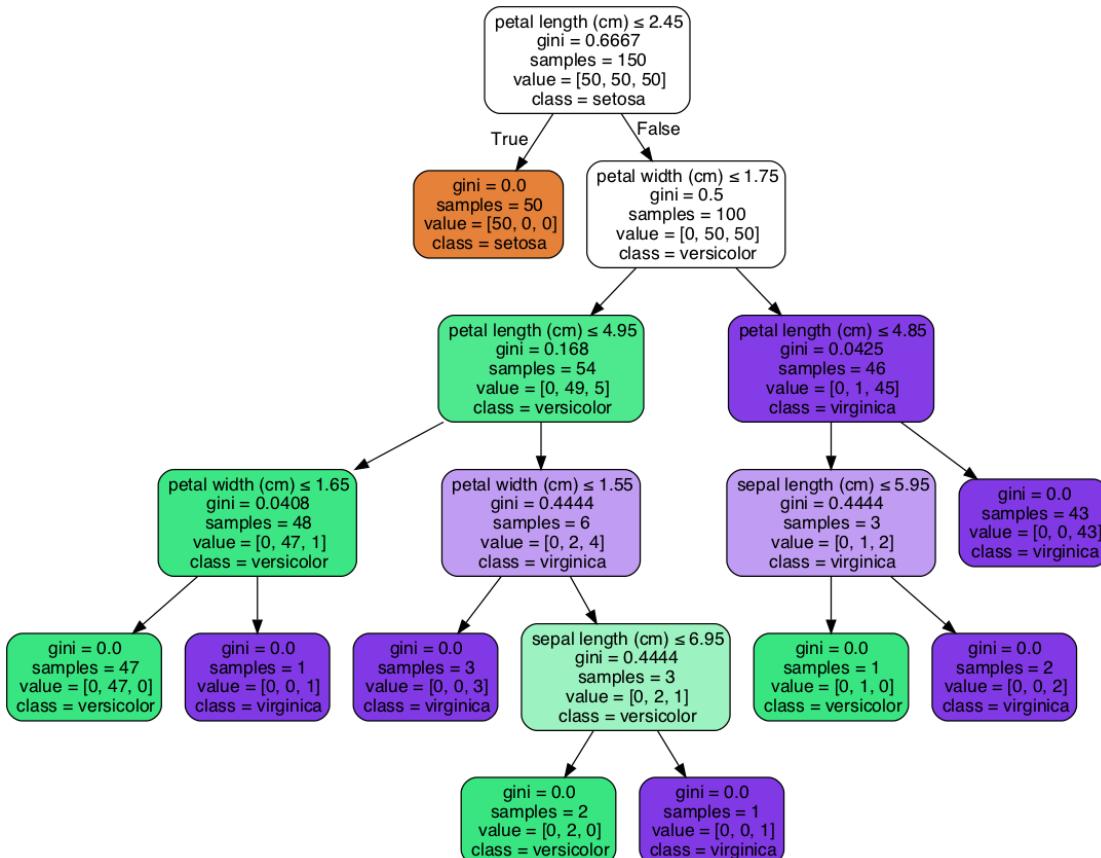
There are several algorithms that can be used to train classifiers. Each technique will dictate the type of model and the prediction performance. There is no ground rule to define which technique is better for each application, so usually more than one case

should be considered. The following sections describe the techniques investigated in this work.

### 3.2.1 Decision Trees

This is one of the simplest and yet most successful machine learning techniques. A Decision Tree receives a feature vector as input and returns a decision (class) based on a series of logical tests over the input values. Figure 3.3 illustrates a tree for deciding the species of a plant based on a famous data set called Iris. The first line in each node represents the test to be applied in the corresponding feature. The second line contains the gini impurity (a measure of how often a randomly chosen element from the set would be incorrectly labeled) at that node. Note that only continuous values were used as features, but discrete values are also supported.

Figure 3.3: A Decision Tree for classification of the Iris dataset. (adapted from (SCIKIT, 2018))



Computationally, Decision Trees are modeled either as Directed Acyclic Graphs (DAG) or as a set of rules formed with logical tests. The canonical algorithms for building

Decision Trees were proposed by Quinlan (QUINLAN, 1986), namely the seminal Iterative Dichotomizer 3 (ID3), its first improvement, the C4.5, and more recently the C5.0 implementation. All versions follow the same strategy:

1. Select the feature with highest importance ( $f_{best}$ );
2. Split the data into the values of  $f_{best}$  the separate classes the most;
3. Repeat steps (1) and (2) to each split subset until the stopping criterion is reached.

The differences of each implementation lie on how these steps are performed. The ID3 uses Information Gain (IG) as metric to select  $f_{best}$ , whereas C4.5 and C5.0 use the improved Information Gain Ratio (IGR). The ID3 algorithm did not support continuous features or incomplete data samples (when the values of some features is unknown), but this was implemented in both subsequent versions. Finally, the C4.5 and C5.0 implement a bottom-up pruning technique to reduce over-fitting and generate smaller trees.

The main differences between C4.5 and C5.0 is that the latter provides support for boosting, automatic removal of irrelevant features (attribute winnowing), and misclassification weighting.

The IG metric used in the ID3 algorithm measures the expected gain in entropy ( $H$ ) of the output when some information is given. In other words, given a training set  $T$  with  $P$  positive and  $N$  negative samples, the IG computes how  $H(T)$  is reduced with the information from feature  $f$ . This is formulated as:

$$IG(T, f) = H(T) - H(T|f) \quad (3.1)$$

$$H(T) = -(P \log_2 P + N \log_2 N) \quad (3.2)$$

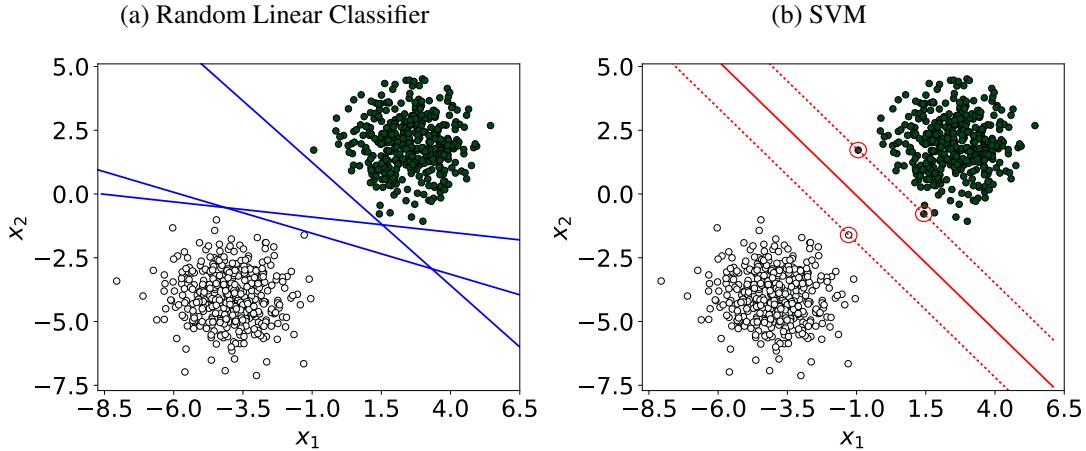
$$H(T|f) = \sum_{v \in vals(f)} \frac{|T_{f=v}|}{|T|} * H(T_{f=v}) \quad (3.3)$$

In (3.3),  $vals(f)$  represents the possible values of feature  $f$ , and  $T_{f=v}$  corresponds to the set of samples where the value of feature  $f$  is equal to  $v$ , i.e.,  $T_{f=v} = \{x \in T \mid x_f = v\}$ .

For features that can take on a large set of values, the IG might lead to over-fitting. This happens, for instance, when a feature has a unique value in each training sample, leading to a high IG but reducing the generalization of the model.

The Information Gain Ratio used in C4.5 and C5.0 is based on the ratio between

Figure 3.4: (a) Three classifiers with inefficient decision margins and (b) an SVM linear classifier with maximum margin



the Information Gain and the Intrinsic Value (IV), which measures the potential information generated by splitting the training data into each value of  $f$ .

$$IV(T, f) = - \sum_{v \in vals(f)} \frac{|T_{f=v}|}{|T|} * \log_2 \left( \frac{|T_{f=v}|}{|T|} \right) \quad (3.4)$$

This equation shows that the IV increases with the number of subdivisions that a feature produces in the training set, so lower values indicate better efficiency. Finally, the IGR is obtained from the division of both IG and IV:

$$IGR(T, f) = \frac{IG(T, f)}{IV(T, f)} \quad (3.5)$$

Features with high IG but also with high IV will have a reduced IGR value, which shows why this metric is used in the recent Decision Tree algorithms.

### 3.2.2 Support Vector Machines

Support Vector Machines emerged as an approach that solved an important issue in classifiers: the decision margin optimization. To illustrate this problem, let us consider Figure 3.4.

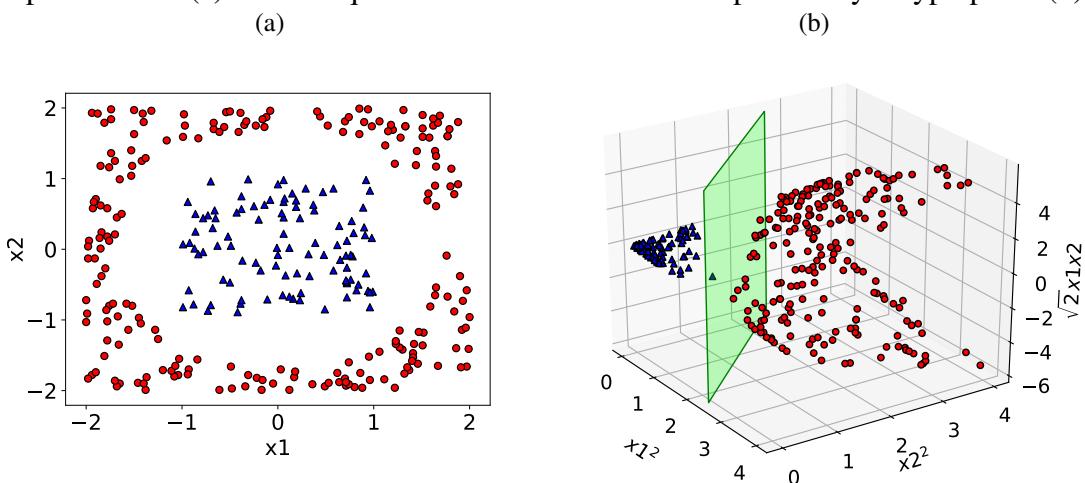
The hypotheses in Figure 3.4(a) are all capable of separating both classes, but the decision margin never seems to grasp the actual behavior of the data. In contrast, Figure 3.4(b) shows the maximum margin between the training samples of each class that are closest to the decision boundary (circled points in Figure 3.4(b)).

The basic idea behind SVMs is that some training samples are more important than others and paying more attention to them might lead to a better generalization.

The SVM training algorithm consists in discovering those circled points (called support vectors) and using them to make decisions on future instances.

When training data is not linearly separable, SVMs apply a transform function called kernel, which maps the representation to a higher dimensional space. This is better illustrated in Figure 3.5. Note in Figure 3.5(a) that the decision boundary is not linear ( $x_1^2 + x_2^2 < 1$ ), but it becomes linear after mapping the samples to a tridimensional space ( $x_1^2, x_2^2, \sqrt{2}x_1x_2$ ) in Figure 3.5(b).

Figure 3.5: Example of the kernel function employed in SVMs to transform linearly inseparable data (a) into an equivalent distribution that is separable by a hyperplane (b).



There are several possible kernels to apply, but SVM implementations come with a preset that contains a Gaussian, a polynomial, and a linear kernel.

Since this work will focus on using pre-built implementations, we will not dive into the mathematical details of SVMs. However, it is important to note that finding the decision boundary is a Lagrangian optimization problem represented by the following formula:

$$\arg \max_a \left\{ \sum_{i=1}^n a_i - \frac{1}{2} \sum_{i,j=1}^n a_i a_j y_i y_j K(x_i, x_j) \right\} \quad (3.6)$$

$$\text{subject to } \begin{cases} 0 \leq a_i \leq C, & \forall i \\ \sum_{i=1}^n a_i y_i = 0 \end{cases} \quad (3.7)$$

The  $C$  parameter is used to tune the trade-off between misclassification and model

generalization. High values of  $C$  will drive the SVM algorithm to train classifiers with few misclassifications, but it also increases the number of support vectors and might lead to model overfitting.

Training SVMs is a particularly hard task, as it includes performing the Lagrange optimization for different values of  $C$ , along with the hyper-parameters of each kernel. For instance, the Gaussian kernel has an extra gamma parameter  $G$ , which defines the area of influence of each unitary training sample in the decision boundary, where lower values mean higher influence for each sample.

There are no heuristics to find the optimal values for  $C$  or  $G$ , so usually a search is done over all possible combinations, which is very time consuming. Despite this drawback, SVMs are still widely used due to their maximum margin property and better data separability characteristics.

### 3.2.3 Ensemble Learning

Both Decision Trees and SVMs generate a single model for classification. The idea behind Ensemble Learning is to select a collection of trees or SVM hypotheses and combine their predictions to form a compound model.

The motivation behind this is simple: if we have five models, we can use majority voting to make a decision, which will be incorrect only when three models make wrong predictions.

The most widely used Ensemble method is called Boosting. This technique uses instance weighting on the input samples, i.e., each training sample will contain a weighting factor that is adjusted iteratively.

Boosting starts by setting an equal weight ( $w_j = 1$ ) to all training samples, which is the same as usual training procedure. The resulting model ( $h_1$ ) will be tested against these training samples and make eventual misclassifications. Afterwards, the misclassified instances are assigned with larger weights, whereas the correctly classified ones will have a smaller weight. Samples with higher weights will be of more importance in the next iteration. The next hypothesis ( $h_2$ ) will be trained with the adjusted samples, increasing the odds of correctly classifying the instances misclassified by  $h_1$ .

AdaBoost (FREUND; SCHAPIRE, 1995) is a successful Boosting algorithm, implemented as default method for building Ensemble Decision Trees in the C4.5 and C5.0 implementations. The pseudo-code is described in Algorithm 1.

---

**Algorithm 1** The AdaBoost algorithm used to build ensemble models
 

---

**Input:** **examples:** set with  $N$  training samples identified as  $(x_1, y_1), \dots, (x_N, y_N)$

**L:** a learning algorithm

**K:** the number of classifiers to train

```

1:  $h \leftarrow \emptyset$                                  $\triangleright$  vector with K classifiers
2:  $w \leftarrow 1/N$                                  $\triangleright$  vector with N example weights
3:  $z \leftarrow \emptyset$                                  $\triangleright$  vector with K classifier weights
4: for each  $k = 1$  to  $K$  do
5:    $h[k] \leftarrow L(\text{examples}, w)$ 
6:   error  $\leftarrow 0$ 
7:   for each  $j = 1$  to  $N$  do
8:     if  $h[k](x_i) \neq y_i$  then
9:       error  $\leftarrow$  error +  $w[j]$ 
10:      else
11:         $w[j] \leftarrow w[j]*\text{error}/(1-\text{error})$ 
12:      end if
13:    end for
14:  end for
15: bestModel  $\leftarrow \text{MAX}(\vec{models}, \text{key} = \text{BDC}_{Ratio})$ 

```

---

### 3.2.4 Random Forests

Random Forests are a special case of Ensemble Learning that consists in combining several Decision Trees into a single classifier. This technique was created as a means to reduce the overfitting tendency of Decision Trees by inserting randomness in the construction of each individual tree (FRIEDMAN; HASTIE; TIBSHIRANI, 2001). The canonical algorithm for building Random Forests was proposed in a paper by Breiman (BREIMAN, 2001), and is composed of two main steps:

- **Tree bagging:** in this step the Bootstrap Aggregating (Bagging) technique is employed, which consists in random sampling the training set with replacement, then training the trees with each resulting subset. After the trees are trained, the decision is made by either taking the average the predictions of all the individual trees (for regression) or by computing the majority voting (for classification problems). The bagging method, also proposed by Breiman, reduces the variance of the model with the subsampling and averaging steps.
- **Randomized Decision Tree Training:** in order for a collection of Decision Trees to be called a Random Forest, at least some degree of randomness must be part of the process. In this algorithm, this is introduced in the features selection step that

takes place in the split decision of each node. Traditionally, the algorithm ranks all the available features and selects the best one using a metric like IGR. In Random Forests though, only a random subset of these features is available. This is done to reduce the correlation among the trees, i.e., if one or a few features are very strong predictors for the target label, they will be selected by most of the trees in the forest, causing them to become correlated (therefore redundant).

Random Forests have been used in a variety of applications, including for arbitrary downscaling of video sequences (VAN et al., 2015). They require more training time compared to Decision Trees, but are still easier to train compared to Neural Networks and SVMs, as they do not involve several variables that must be optimized.

### 3.3 Evaluating Models

The performance of binary classification models can be assessed with different metrics, most of which can be formulated using known concepts in machine learning: True Positives (TP), which stand for the number of correctly classified Positive values; and True Negatives (TN), representing the Negative values. In addition, the False Positives (FP) and False Negatives (FN) are interpreted as the additive inverse of TP and TN.

Three common metrics that will be used in this thesis, True Positive Rate (TPR), True Negative Rate (TRN), and accuracy (ACC), are formulated below:

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN} \quad (3.8)$$

$$TNR = \frac{TN}{N} = \frac{TN}{TN + FP} \quad (3.9)$$

$$ACC = \frac{TP + TN}{(TP + FP + TN + FN)} \quad (3.10)$$

Other metrics like Logarithmic Loss and Area Under ROC Curve (AUC) can be used, but accuracy is usually a good starting point to assess different models. In addition, TPR is useful in particular cases where True Positives are more important than True Negatives.

### 3.4 Tools and Frameworks

With the huge success of machine learning applications, several tools, libraries, and frameworks are constantly being developed. These tools comprise every task involved in the process, from data analysis and preprocessing to model training and testing. The following tools were identified as the most prominent ones among several possibilities:

- **R:** R is a special language designed specifically for data machine learning and statistical learning implementations (TEAM, 2014). It provides a series of databases, learning algorithms for classification and regression, feature analysis methods, etc. The R community is also very active, providing support via forums and custom libraries for specific purposes.
- **Weka:** this software is a collection of machine learning algorithms for data mining tasks (WITTEN et al., 2016). The algorithms can either be applied directly to a data set or called from a Java code using the source codes provided. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes.
- **Scikit-Learn:** this is a set of packages available in Python that greatly assist every step required for model training (PEDREGOSA et al., 2011). The Scikit-Learn libraries implement virtually all traditional learning algorithms, a plethora of data processing and analysis methods, and they also contain many famous data sets. Combined with NumPy for fast vector operations processing and PyPlot for graphical features, Scikit-Learn is one of the most complete and powerful tool sets available.
- **TensorFlow:** this is an open source software library created by the Google Brain Team for the purposes of conducting machine learning and deep neural networks research (ABADI et al., 2016). The computations are modeled using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows one to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API.
- **LibSVM:** this C++ library contains specific functions to train and test models using Support Vector Machines (CHANG; LIN, 2011). Classification and regression models are supported. This library also implements different kernel functions, and

an interesting feature that provides the probability of a decision to be correct. Some helpful scripts in Python can also be obtained from the library's website, providing feature selection and data subsampling routines.

- **C4.5/C5.0:** these are also C++ libraries, but for Decision Tree learners (QUINLAN, 2006). Model training, testing, and even ensemble methods (using AdaBoost) are implemented.

From this list, the Scikit-Learn was used for data analysis and preliminary model training. However, the HM software is implemented in C++, so the specific LibSVM and Decision Trees libraries (C5.0) were used to modify the HM encoder.

A final note on this topic is that the need for better and more efficient machine learning environments are also encouraging the circuits and systems industry. Google's Tensor Processing Unit (TPU) is one of the main examples. The TPUs are application-specific processors developed to improve the performance and power efficiency of TensorFlow operations (JOUSSI, 2016). These units are especially interesting, because they reduce the precision (bit width) of operations in order to maximize the IOPS (input/output operations per second) per Watt.

## 4 CHALLENGES AND RELATED WORKS

This chapter begins with a complexity analysis of HEVC encoders, which will be used to identify the challenges connected with the topic of this thesis. Following, a literary study of the main contributions regarding complexity reduction and Machine Learning in visual applications will be presented, with special focus of HEVC encoding and transcoding.

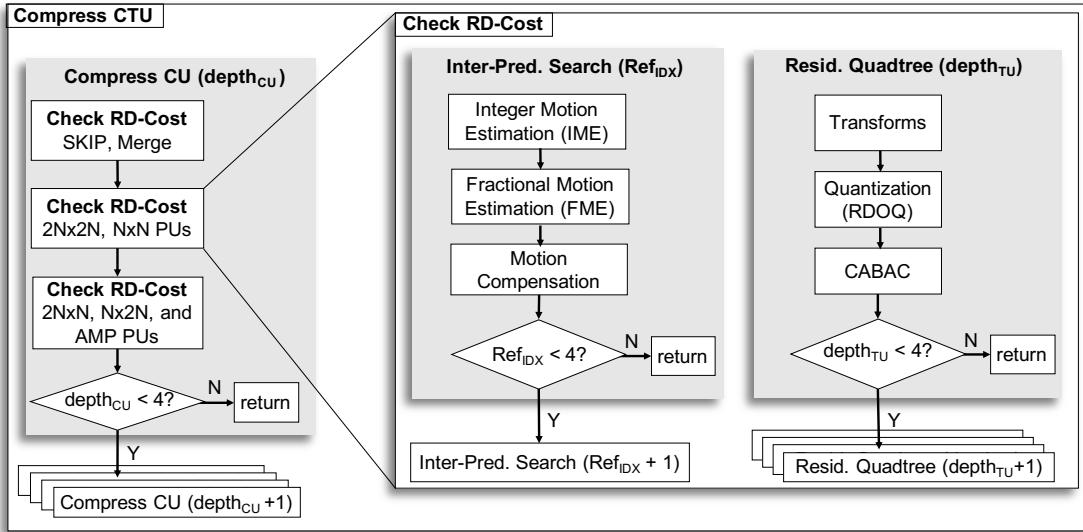
### 4.1 HEVC Encoding Complexity Analysis

The H.265 norm (ITU-T, 2013) dictates how to decode HEVC-compliant bit-streams, explaining how each decoded word must be interpreted. Therefore, there is not much ground to innovate in HEVC decoders, apart from parallel implementations, CPU-specific optimizations, etc. The encoder, however, can be implemented in many different ways, with a single restriction of generating streams whose syntax is compliant with the HEVC norm. While this opens a large space for exploration, it also makes it difficult to compare different solutions under fair circumstances. Therefore, experts associated with the JCT-VC group developed the HEVC Model (HM) software to serve as a reference for research with this standard. The HM was not designed to be efficient in terms of complexity, but in return it contains all the tools supported by HEVC, thus it was adopted by many researches of both industrial and academic branches.

The chart in Figure 4.1 gives an overview of the tasks involved in the compression of a single CTU using the HM implementation. The round arrows indicate loops, and the associated conditions represent the default configuration. Note that the *Compress CU* and *Residual Quadtree* loops are actually replicated four times at each iteration to form the quadtree structures.

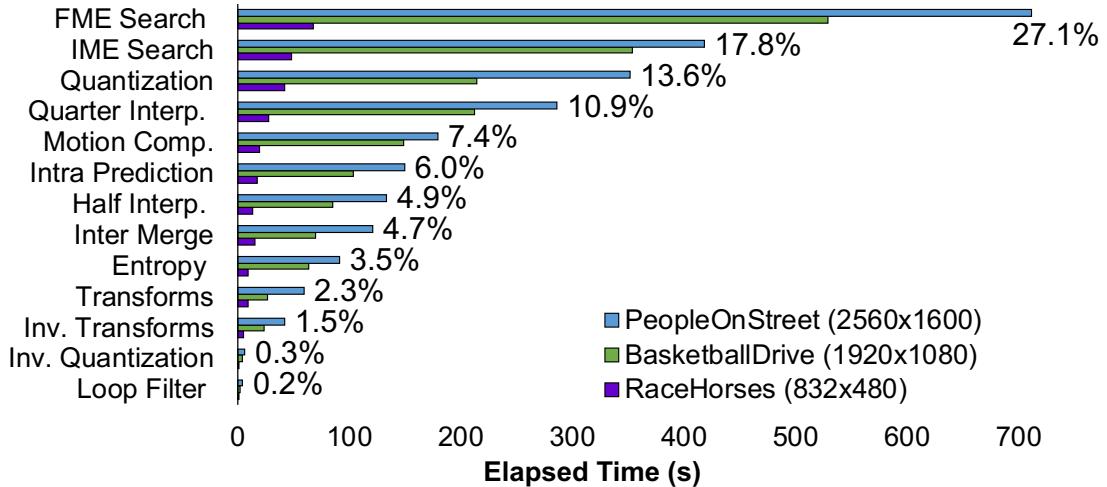
Note in Figure 4.1 that the *Inter-Prediction Search* and the *Residual Quadtree* loops that compose the *Check RD-Cost* process are nested inside the *Compress CU* method, which forms the CTU quadtree. This composition gives us a hint of which encoding tools are responsible for most of the encoding execution time, as most of these tasks involve several computations to be performed. For instance, the Integer Motion Estimation (IME) search involves computing Sum of Absolute Differences (SAD) of each candidate, whereas the Fractional Motion Estimation (FME) search is composed of a fractional interpolation step followed by a block-matching search using the Sum of Absolute Trans-

Figure 4.1: Main tasks required to encode a single CTU of an HEVC encoder.



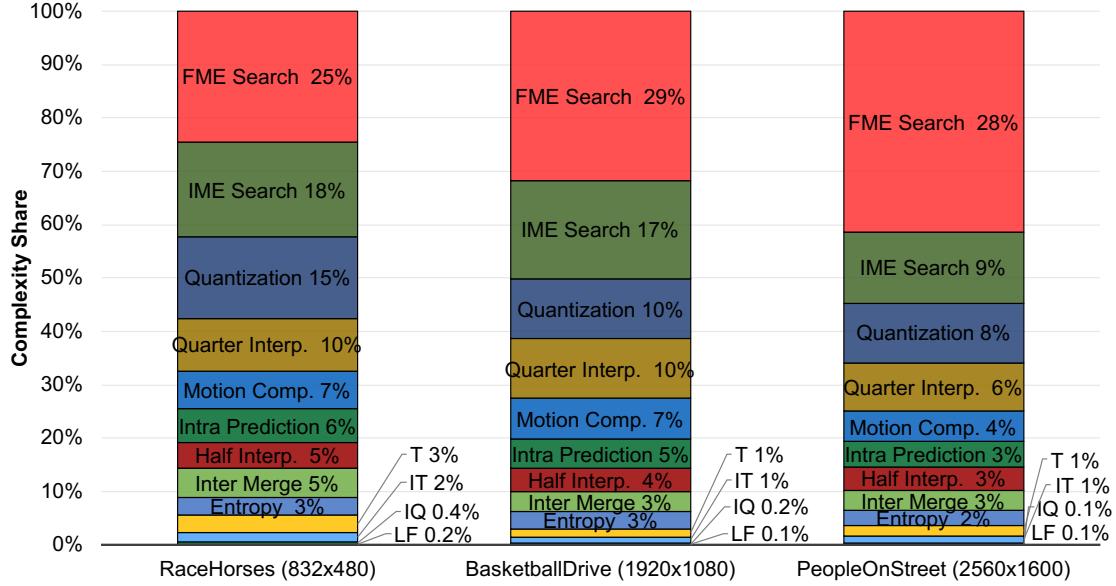
formed Differences (SATD) as metric. To quantify how each step is involved in the encoding complexity, the HM was instrumented using the GNU Profiler (GProf) tool set, and it was monitored while encoding 32 frames of three sequences with a QP of 32. The results are presented in Figure 4.2 in terms of absolute elapsed time, and also in the form of a complexity share breakdown in Figure 4.3.

Figure 4.2: Processing time and average complexity share of several encoding components for three video resolutions (sorted by average time share).



As Figures 4.2 and 4.3 show, the FME is the most computation-intensive step in the encoding process, consuming on average 43% of the encoding time (combining the FME Search and the Half/Quarter Interpolation steps). Other steps related to inter-prediction have shown to be quite consuming as well, such as the IME and the Motion Compensation. It can be also observed in Figure 4.2. that Quantization is almost six times more complex than the Transforms, most likely due to Rate-Distortion-Optimized

Figure 4.3: Complexity share breakdown of encoding components for three video resolutions (sorted by average time share).



Quantization (RDOQ). This algorithm is highly control-flow and contains inter-coefficient dependencies in its mode decision.

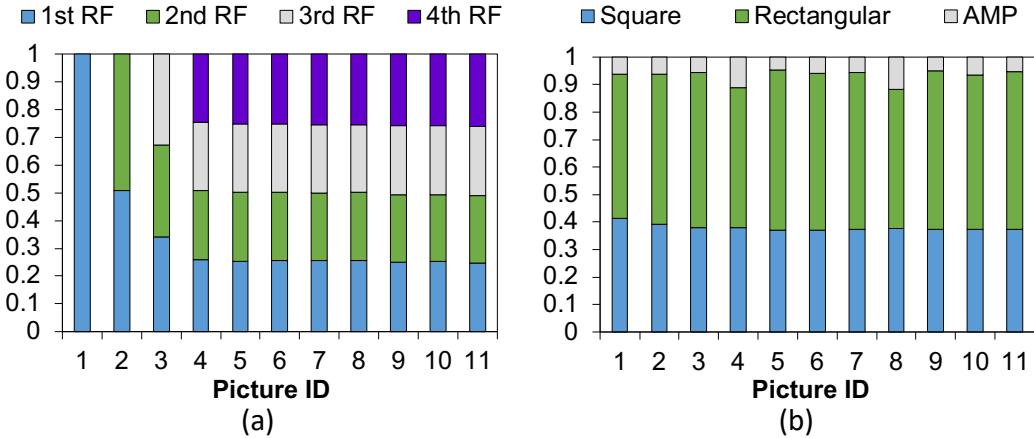
The inter-prediction, and the T/Q/IT/IQ loop add up to 72.7% (combining the FME+IME Searches, the Half and Quarter Interpolations, Motion Compensation, and Inter Merge steps) and 17.7% (combining the Transforms, Quantization and their inverse counterparts) of the total time respectively. In light of these facts, a detailed analysis was carried out to discover how scalable the inter-prediction complexity is and, more importantly, how this scalability can be extended even further.

During inter-prediction, the pixel block is sub-partitioned into Prediction Units (PU). There are eight possible PUs for each CU in the quadtree: two square Symmetric Motion Partitions ( $2N \times 2N$ ,  $N \times N$  SMPs); two rectangular SMPs ( $2N \times N$ ,  $N \times 2N$ ); and four asymmetric ones (AMP). From these, only the one with best rate-distortion values is effectively used.

As depicted in Figure 4.1, the FME and IME searches are called for each reference frame (RF), which by default is equal to four. Figure 4.4. shows the inter-prediction time spent in each RF and also in each PU group.

Figure 4.4(a) indicates that the search takes about the same time in each RF. This is interesting for this work, as it means that a 25% reduction is expected each time the number of RFs is reduced by one. In addition, Figure 4.4(b) shows that the rectangular SMPs ( $2N \times N$ ,  $N \times 2N$ ) take 55% from the ME time, whereas the AMPs represent a minor 7% slice due to a fast mode-decision algorithm that skips AMP evaluation in many cases.

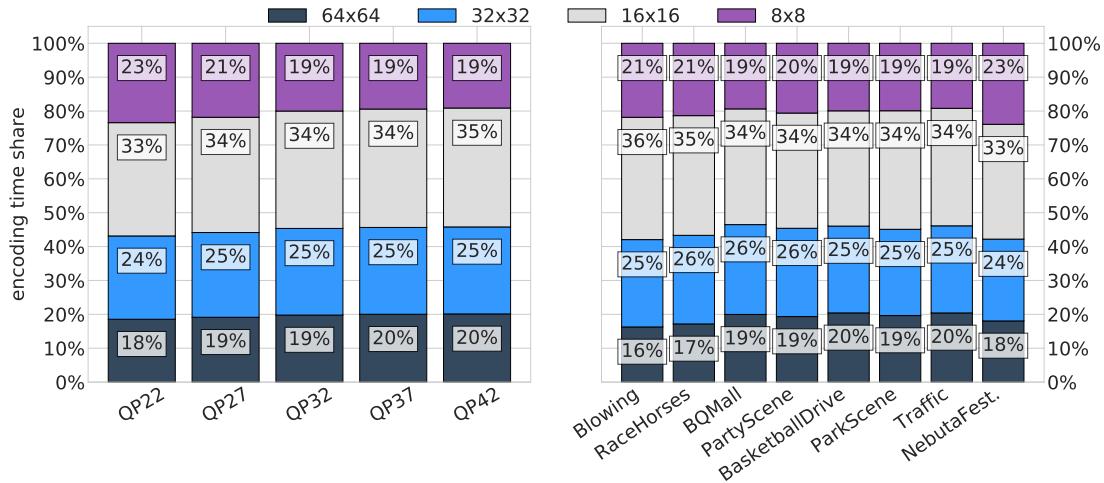
Figure 4.4: Motion Estimation complexity share of in each reference frame (a) and in each Prediction Unit group (b) – sequence: BQMall ( $832 \times 480$ ).



Surprisingly, AMP can be completely disabled in the current HM versions, but this is not possible for the time-consuming SMPs.

The encoding time spent within each CTU quadtree depth (or for each CU size) is also an important indicator, as it allows us to estimate how much time can be saved using CU early termination techniques. The results depicted in Figure 4.5 show this distribution in terms of QP (a), using the average of all tested sequences, and in terms of sequence, using the average of all QP values (b).

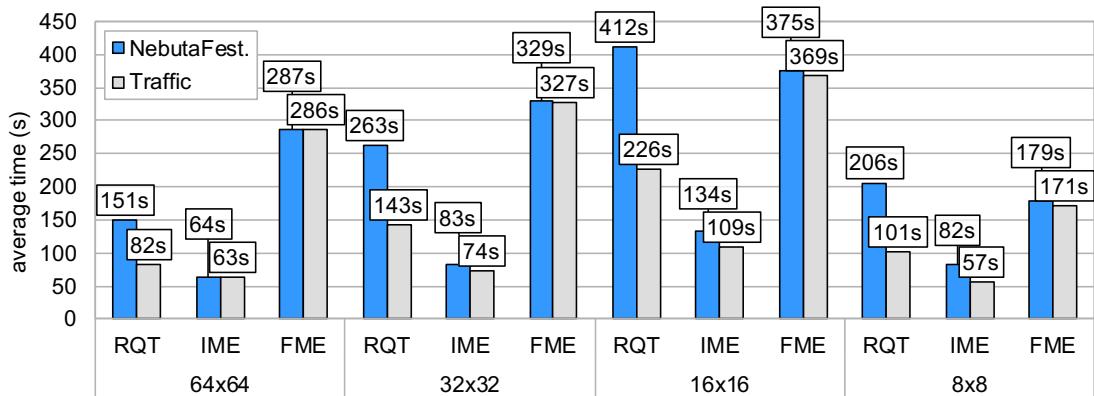
Figure 4.5: Encoding time share spent to process each CU size. The results were averaged for each QP (a) and for each sequence (b).



From this figure we deduce that significant encoding time is spent on smaller CUs. For all tested cases, regardless of QP or sequence resolution, more than 50% of the encoding time is spent on 16x16 and 8x8 CUs. If 32x32 CUs are also considered, the complexity share increases to more than 80%. These results show the potential of early CU termination methods and justifies the great amount of solutions that have been proposed

on this topic. On the left side of Figure 4.5, it is noticeable that the QP does not affect this distribution significantly, although lower QP values seem to increase the complexity share of 8x8 CUs a little. The sequence-wise bars in Figure 4.5 show a similar behavior, where more time is spent on smaller resolutions. An exception is the *NebutaFestival* sequence, which has a 2560x1600 spatial resolution and seems to spend more time on 8x8 CUs compared to its counterpart of the same resolution (Traffic). It is difficult to figure out where this discrepancy occurs though, so Figure 4.6 presents a more detailed profiling of these two sequences, showing the average encoding time (QP-wise) spent on the RQT, FME and IME processes for all CU sizes.

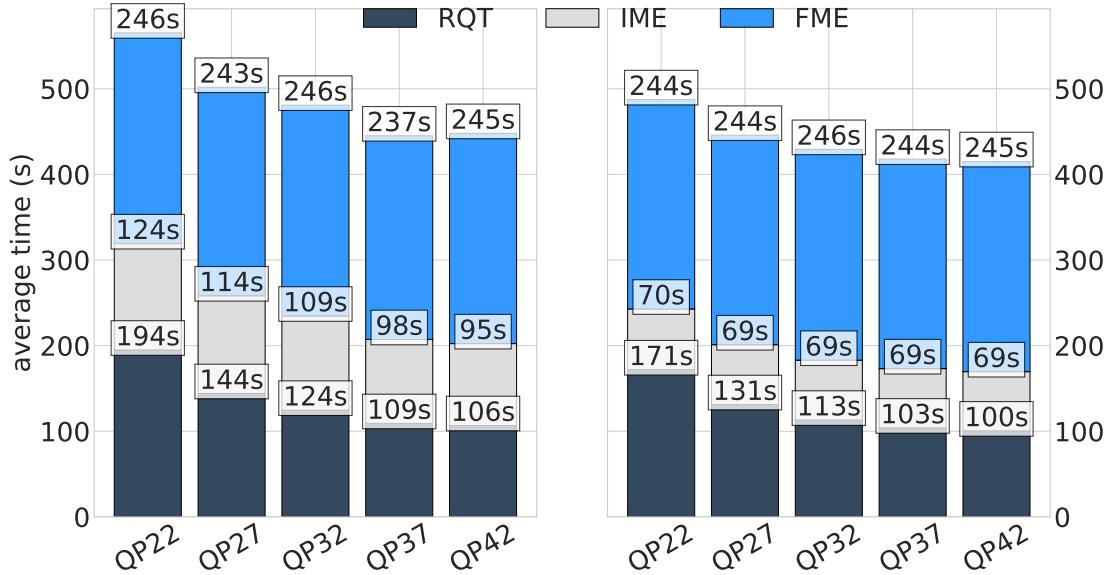
Figure 4.6: Encoding time spent on the main HEVC tools for two sequences with 2560×1600 resolution.



Although both sequences have the same spatial resolution, Nebuta takes almost twice the time in the RQT processing compared to Traffic. The main explanation for this comes from a fast method implemented in recent HM versions that inserts a termination criterion in the RQT subdivision process. Basically, the subdivision stops when the absolute sum of transformed coefficients is below a threshold value, which is computed based on the QP. Therefore, scenes that are less complex in motion and texture information achieve the termination condition more easily. In fact, this also explains why higher QP values reduce the encoding time for the same sequence. Figure 4.7 shows that the RQT processing and the IME take less time as we increase the QP (thus decreasing the number of significant coefficients).

Note that the FME takes practically the same amount of time, whereas the RQT time decreases by more than 45% for BasketballDrive and 41% for ParkScene when QP changes from 22 to 42. In the framework of this thesis, where fast decisions were envisioned, this means that eliminating the RQT processing has a smaller potential compared to a CTU early termination method, especially with higher QPs. On the other hand, it

Figure 4.7: Average encoding time spent on the RQT, IME, and FME processes for the BasketballDrive and ParkScene sequences (1920×1080).



also gives us a clue regarding the importance of the residual information in the encoder decisions.

## 4.2 Related Works

Machine Learning techniques have been applied in several studies related to Video and Image Processing. The following section will highlight important contributions in different visual applications. Section 4.2.2 focuses on complexity reduction solutions for video encoders. Section 4.2.3 presents some relevant video-coding complexity reduction studies that make use of Machine Learning techniques. The references related to complexity management in video encoders are presented in Section 4.2.4. Finally, Section 4.2.5 presents important contributions regarding HEVC transcoding.

### 4.2.1 Machine Learning for Visual Applications

Some works found in the literature apply Machine Learning techniques to solve problems related to Video Coding, such as (XU et al., 2017), (WANG; ZHU; YIN, 2016), and (WU et al., 2016).

(XU et al., 2017) shows that compressed domain features can be effectively used

for salience detection using a wide eye-tracking database. This work points out that regions with higher splitting depth, higher bit allocation and/or larger motion vectors information are often the regions that attract human attention, proving that these features can be used for attention modeling.

(WANG; ZHU; YIN, 2016) presents a video anomaly detection mechanism using Extreme Learning Machines (ELM, a particular case of Neural Networks). As features, the authors use texture information from Local Gradient Pattern (LGP) descriptors combined with Optical Flow descriptors for motion information.

A crowd movement detection mechanism based on Support Vector Machine learners is presented in (WU et al., 2016). The authors designed a retrieval system that is capable of detecting hand-drawn motion patterns in video scenes of crowded areas in order to facilitate indexing and retrieval applications. A series of motion-based calculations are used as features for classification.

#### **4.2.2 Complexity Reduction in Video Coding**

This section presents a collection of works found in the literature that propose solutions for HEVC complexity issues based on heuristics faster than the ones already defined in the standard. The works are classified according to the coding stage affected.

##### *4.2.2.1 Intra Prediction*

(JIANG; MA; CHEN, 2012) presents a fast mode decision algorithm for the intra-prediction process based on applying a Sobel filter on the picture to identify regions with more details. The authors claim that, based on optical flow theories, the gradient direction of a pixel represents its maximum variation, whereas the perpendicular direction indicates the minimum. Thus, before calling the intra-prediction method, the gradients of each pixel are calculated, identifying the dominant gradient direction of the PU being encoded. Lastly, the intra-mode directions evaluated are the ones perpendicular to the dominant gradient. The results show a 20% time savings with a 0.74% increase in BD-BR using the All-Intra configuration.

In the work of (KIM et al., 2011), the authors propose a heuristic that limits how deep the quadtree goes on intra CTUs. This kind of technique is usually referred to as Early Termination. Two methods are proposed: the first one evaluates the best mode of

the CU immediately above (in terms of depth) in the quadtree. If the best mode from the above CU is the same as the one for the current CU and if the current CU size is the same as the current TU size, the quadtree partitioning is interrupted. The second method considers a mode decision heuristic that was actually adopted by HEVC, called Rough Mode Decision. RMD consists in evaluating a smaller set of the 35 available directional modes. Based on the best result from the RMD, a refinement is applied, evaluating only the modes similar to the best one from RMD. By combining both methods, the results point a 23% time reduction, with a 0.9% BD-BR increase using the All-Intra configuration. The RMD was actually implemented in the HEVC reference, showing the efficacy of this solution.

Other works related to intra-prediction were found. To cite a few: (ZHAO et al., 2011), (YAN et al., 2012), (SHEN; ZHANG; AN, 2013), and (SHI et al., 2013). However, since this dissertation focuses mostly on inter-prediction, they will not be discussed here.

#### *4.2.2.2 Inter Prediction*

The authors in (LENG et al., 2011) employ a frame-level fast mode decision, which is based on counting the occurrence of CTU depths. If the occurrence of a given depth  $d$  is much smaller than that of  $d + 1$ , then  $d$  is not evaluated for all CTUs of the next frame. Secondly, a CTU-level decision evaluates the neighboring CTUs to decide the depth interval that will be evaluated in the current CTU. The average results correspond to a 45% reduction in the ME execution time, whereas bitrate is increased by 0.01% and PSNR is decreased by 0.066dB. However, the authors did not present results in accordance with the CTC, so it is difficult to define the quality of the results.

(CHOI; PARK; JANG, 2011) propose a CTU Early Termination algorithm based on the occurrence of the SKIP mode. The algorithm is rather simple: if SKIP mode is selected as best mode in the current depth, then the following depths are not evaluated. The authors present statistics showing that if SKIP is selected as best mode for depth  $d$ , the next depth has a 5% chance of having a better mode. Encoding time is reduced by 42%, with a 0.6% BD-BR increase.

The solution presented in (CASSA; NACCARI; PEREIRA, 2012) is composed of two fast mode decisions. The first one, named Top Skip, consists in skipping the larger CU sizes. To support this decision, the authors claim that there's a low probability of finding the best mode in these CUs, especially in high-resolution pictures. Therefore, the Top Skip decision starts from the largest CU found on the previously encoded CTU. The

second decision tries to remove the lower CUs based on an Early Termination algorithm, which defines a threshold value based on the standard deviation of the residual samples. The results show a 40.3% encoding time reduction, with a 1.91% bitrate increase and 0.007dB BD-PSNR decrease.

The following works also target inter-prediction fast mode decisions: (KIM et al., 2012) and (XIONG et al., 2014). The inter-prediction is the most computation exhaustive process in the encoding loop, therefore many researchers focus on trying to reduce its complexity. The references cited here are only some examples from many.

#### *4.2.2.3 Other HEVC Encoding Steps*

In (TENG; HANG; CHEN, 2011), the authors present a fast decision for the TU mode decision. The proposed solution reverses the evaluation order of the possible TU mode, prioritizing smaller TU sizes. The authors also define some Early Termination conditions: when the PU transformed coefficients after the transforms are all zero, then quadtree partitioning is interrupted. The divulged results show a 49% average time reduction in the TU mode decision time (defined as RQT), with a 0.31% BD-BR increase and a 0.013 BD-PSNR decrease. The time reduction in the overall HEVC encoder was not divulged.

Many other works that target a specific HEVC component can be found. (MIYAZAWA et al., 2012) presents an algorithm that disables the Adaptive Loop Filter (ALF) when certain conditions are met. (JOO; CHOI; LEE, 2013) proposes a fast algorithm for SAO encoding based on the best intra mode.

### **4.2.3 Machine Learning for Video Coding Complexity Reduction**

Most references that focus on complexity reduction using Machine Learning classifiers deal with the RDO mode decision, due to the enormous optimization space of this method. In (CORRÊA et al., 2015), the authors present fast decision models for CU, PU and TU mode decisions using Decision Trees as learners. Several coding domain features, like splitting depth and rate-distortion cost, are used as features to build each Decision Tree. The authors also present a feature analysis, showing that coding domain features are very useful for mode decision classification. The results show that a 65% complexity reduction can be achieved with minor losses of 1.36% in coding efficiency

(using Bjøntegaard Delta Bitrate – BD-BR (BJONTEGAARD, 2001)).

Momcilovic et al. (MOMCILOVIC et al., 2015) present a run-time learning system based on Neural Networks. The authors extract coding features online and use them to train Neural Networks asynchronously on a separate thread. When the model is trained, a signal is sent to the encoder thread, and the model is then applied for fast CU decisions. The results show that 47.5% complexity reduction is achieved on average with a 1.17% BD-BR penalty.

A SVM-based fast CU decision is proposed in (ZHANG et al., 2015a). In this work, the authors designed a hierarchical classification framework for different block sizes. The features used for model training include coding features, like SKIP flag, motion vector, and neighboring depth. Online and offline training modes are discussed, showing that it is possible to build adaptive models that fit the scene characteristics. This solution achieves an average complexity reduction of 51.45% with 1.98% BD-BR loss.

The authors in (SHEN; YU, 2013a) also present a CU splitting early termination algorithm to reduce encoding complexity. CU splitting is modeled as a binary classification problem, on which a SVM is applied. The authors apply a weighted SVM strategy to reduce the rate-distortion loss that occurs during misclassifications. The presented results show that the proposed algorithm achieves about 44.7% complexity reduction on average with 1.35% BD-rate increase under the Random Access configuration.

Another SVM approach for CU and PU partitioning is proposed by (ZHU et al., 2017). This work employs binary classification model for CU early termination, whereas the PU partitions are decided with a multi-class SVM. The models are trained offline, online, or in a mixed mode that combines both offline and online versions. Using the mixed online/offline mode, a complexity reduction of 65.6% is achieved with a 3.67% BD-BR increase under the Random Access configuration.

A Bayesian Decision Rule using online-trained models is proposed by (KIM; PARK, 2016). The authors propose the use of joint online and offline learning methods to produce a more robust classifier. The online models were trained with the minimum error Bayesian decision rule with selective input training pictures. The offline model is responsible for modeling the loss function based on the minimum risk Bayesian decision rule. The proposed method reduces the computational complexity of HEVC encoder by 34.9% on average with a 0.71% BD-BR loss using the Random Access configuration.

#### 4.2.4 Complexity Management in Video Coding

This section describes the most relevant works on complexity management found in the literature, as well as one that targets thermal control. These solutions make use of the complexity scalability that is provided by many HEVC tools due to their high configurability.

The main challenge consists in how to efficiently scale complexity up and down in order to maintain quality and compression requirements. (MIRTAR; DEY; RAGHUNATHAN, 2012) presents a solution based on dynamic thermal control for an H.264/AVC encoder. The goal of this work was to reduce coding complexity before the CPU reaches temperature levels considered too high. The authors initially defined different thermal complexity profiles, with varying frequencies and voltages. The second step consisted on running the H.264/AVC encoder for each profile in order to create offline complexity estimations for all sequences. These results were stored in what they called a Thermal Policy Characterization Table (TPC). The authors do not detail how the TPC used in the presented solution was built, but apparently, a single sequence was used as training set. To circumvent the offline training issues (e.g., videos that stray from the training set), the authors present an online tuning equation in order to scale the estimations up or down. The results point a 2.4 dB average PSNR drop and a bitrate 4% higher. These results were extracted for three sequences only, compromising the confidence of the analysis.

The work presented in (HUIJBERS; ÖZÇELEBI; BRIL, 2011) proposes a complexity-scalable Motion Estimation algorithm. This method limits the number of candidates evaluated in the ME process, based on the amount of computation available for each Macroblock. Computation budgeting is decided upon the pre-quantized residue generated for the collocated Macroblock in the previous frame: if the residue is considered too high, more computation is granted to the current Macroblock; otherwise, less candidates will be evaluated for it. This routine is called before each Macroblock is processed to compensate cases when the resources are over/under-used, leaving concerns regarding the computation overhead introduced by this solution. The results from this work are only presented as charts, with no quantified values to allow comparisons.

In (TAN et al., 2009), a solution for H.264/AVC encoders that initially selects SKIP mode for all Macroblocks of a wave front (Macroblocks without interdependence) is proposed. After this initial step, the algorithm refines each Macroblock until the available resources are used up. This can be interpreted as a bottom-up budgeting strategy.

The budgeting is calculated by using a single parameter that represents the percentage of available resources. The results in this work are also presented as charts.

In (CORRÊA et al., 2012), a complexity control solution is presented. The authors initially show that the maximum depth of collocated CTUs remains the same through a considerable number of consecutive frames. Based on that conclusion, they designed an algorithm that classifies frames as unconstrained (the regular encoding is performed), and constrained, in which a fast mode decision is applied to limit the maximum quadtree depth (Early Termination). The main contribution in this work is the fact that the authors developed a model that dynamically classifies frames as constrained or unconstrained by using online-based estimations. For a 60% target complexity (40% time savings), the results divulged show a 6.29% BD-BR increase and an actual encoding time savings of 38%.

In (GRELLETT et al., 2017), a complexity control scheme based on parameter presets to reach specific targets was designed. Each preset controls the configuration of several encoding algorithms in order to achieve higher reductions. The presets are assigned to each CTU depending on the estimated computations. The authors also implement a dynamic frame-level control system using PID controllers to increase the accuracy of the system, achieving complexity savings from 10% up to 90%.

A similar strategy is applied in (CORRÊA et al., 2016), in which the authors apply a Pareto-based analysis to generate the best presets for each target savings, optimizing the rate-distortion tradeoff. Dynamic control mechanisms are not considered in this work, as the presets are assigned statically, but the authors are still able to achieve complexity savings near 90%.

In (DENG; XU; LI, 2016), a hierarchical complexity control that also applies CTU and frame-level techniques is proposed, reaching complexity reductions from 10% up to 90%. In this work, a statistical analysis was conducted to assign the maximum depth for each CTU.

#### **4.2.5 Complexity Reduction in Video Transcoding**

The research on HEVC transcoding can be categorized into two groups. The first one comprises solutions that aim at reducing the HEVC transcoding time given an input bitstream encoded in a different standard, such as H.264/AVC. This can be referred to as heterogeneous transcoding. The second group is composed of methods to reduce

transcoding time using a reference HEVC bitstream as input, which is called HEVC bivariate transcoding or homogeneous transcoding. Important contributions from both groups can be found in the literature, and the following sections will discuss the ones that were most relevant to this work.

#### *4.2.5.1 H.264 to HEVC Transcoding*

H.264/AVC encoders have steadily conquered the domestic and professional markets over the last ten years. Therefore, there is a lot of legacy content encoded with this standard, so H.264/AVC to HEVC transcoding is necessary to enable a gradual migration to the most recent technology. Also, H.264 to HEVC transcoders allow streaming services to take advantage of the superior RD performance provided by the latter.

The authors in (PEIXOTO; IZQUIERDO, 2012) propose a fast mode decision for HEVC using the motion vector information from H.264/AVC bitstreams. The algorithm computes the variance of all the MVs that are inside the current CU area, and compare this result with two thresholds ( $T_{low}$ ,  $T_{high}$ ). When there are MVs from different reference frames, this metric is not computed. Based on the computed variance, some decisions can be skipped.

Two strategies are employed to speed up ME as well: MV reuse and refinement (which basically considers the H264/AVC in the search plus some other important cases when refinement is enabled); and MV scaling, which computes the variance even for MVs that are not from the same reference frame, using a distance-based scaling factor.

Different combinations of threshold values, as well as enabling or not the MV refinement and scaling techniques, are tested. Therefore, the authors called their solution complexity-scalable.

Compared to a trivial cascaded transcoder, the authors achieve a minimum speedup of 2x (with 2.92% BD-BD loss) up to 3.6x (with 9.1% BD-BR loss). The results were encoded under VBR conditions, so different QPs were used to achieve bitrate variations.

The authors in (DÍAZ-HONRUBIA et al., 2016) propose an Adaptive Fast Quadtree Level Decision (AFQLD) approach, which exploits the information gathered in the H.264/AVC decoder in order to assist decisions on CU splitting in HEVC using a statistical Naïve–Bayes (NB) classifier to avoid an exhaustive RDO search. Adaptive refers to the fact that the algorithm can dynamically be adapted to the content of each sequence. In an offline data mining process, all the knowledge needed is extracted from the H.264/AVC decoding statistics by means of machine learning (ML) techniques, and is then converted into

mathematical models that can be executed in the online transcoding process.

For decisions at level 0 and 1, they build one model for each level of residual energy (basically different delta QPs in a GOP), adding up to 8 models.

A fast H.264/AVC to HEVC transcoder using Data Mining is proposed by (CORRÊA et al., 2015). Several features from decoded H.264/AVC streams were gathered, including sum of coefficients, Macroblock partitioning information, and number of SKIP Macroblocks within the CU area. Afterwards, the HEVC encoder was used in the same sequences to extract the CU partitioning information, which was used as labels of the data set. A data analysis is provided, showing that there is a great correlation between the number of SKIP Macroblocks and the CU partitioning, as well as with the sum of coefficients. The extracted features were used to train Decision Trees with the C4.5 algorithm, which were then implemented in the HEVC Model code to apply fast CU partitioning decisions. Separate trees were trained for each CU size, except for 8x8 CUs, since they are already the smallest size, thus cannot be partitioned.

Experimental results show that, compared with a cascaded transcoder, a 44% transcoding time reduction is achieved with a 1.67% BD-BR loss. The results were obtained under VBR conditions, using different QPs to achieve varying bitrates.

#### *4.2.5.2 HEVC Bitrate Transcoding*

Generating multiple representations on the fly is not a trivial task even for large companies like Youtube and Netflix, as the computing resources are finite, so fast transcoding solutions are valuable assets to mitigate this problem. In addition, even fast encoders like x265 and VP9 struggle to achieve real-time throughput for Full HD sequences. Lastly, all of these problems will be intensified as Ultra HD resolutions (such as 4k and 8k<sup>1</sup>) are becoming more popular. The following references bring contributions in this direction.

The work of Schroeder et al. (2016) presents a fast transcoding method for adaptive HEVC streaming based on using the encoding information from a high-quality reference to constrain the rate-distortion-optimization of lower-quality encodings. Several reuse strategies are discussed, namely prediction mode, intra angle, motion vector, and CU partitioning. The authors show that there is a high correlation between the decisions of the reference stream and those of the dependent streams.

Most savings are achieved with the CU partitioning reuse, specifically 33.6% on average with a BD-BR increment of 0.53%. The authors also showed that reusing the

---

<sup>1</sup>Full High Definition, 4k, and 8k respectively stand for  $1920 \times 1080$ ,  $3840 \times 2160$ , and  $7680 \times 4320$  pixels

motion vector from the reference encoding actually increases the compression efficiency by 0.12% on average, while time is reduced by 6.2%. However, it is stated that the surrounding area of the reused MV must be searched, because there is usually a small offset. Combining all methods, an average time savings of 38% are achieved with a BD-BR increase of 0.96%.

Wang et. al (WANG et al., 2016) also propose a Single Input/Multiple Output system that speeds up the CU partitioning on subsequent encodings (called Low Quality – LQ) based on input from reference (High Quality – HQ) bitstreams. In this work, however, both HQ and a LQ bitstreams are used as input. The CU partitioning from the HQ bitstream is used to define Lower Bounds in the partitioning decision of the lower-quality representations, while the LQ partitioning is used to infer the Upper Bounds. These bounds are mapped directly from the HQ and LQ streams. Experimental results show that an average savings of 51.91% were achieved using three QP values (27, 32, and 37), with a PSNR drop of 0.7 dB on average. Bitrate results were not divulged.

Decision Trees are used in the fast decision method of (VAN et al., 2013) for HEVC transrating is proposed. The authors extract information from a decoded bitstream in higher quality and use it as input for the tree-based classifier. One model for each CU size (64x64 down to 16x16) was built using the J48 algorithm.

To improve rate-distortion performance, the accuracy of a prediction is also computed and compared with two thresholds in the decision process. If the accuracy is larger than or equal to  $\text{Th}_2$ , the split-flag accuracy is classified as high. If the accuracy is smaller than  $\text{Th}_1$ , the split-flag is classified as a low accuracy. Otherwise, the accuracy is classified as medium. From this analysis, which modes will be processed are detailed in Table 4.1.

Table 4.1: CU evaluation table of the method proposed by (VAN et al., 2013)

Predicted Split	ACC	Check CU <sub>d</sub>	Check CU <sub>d+1</sub>	ET
0	High	Y	N	Y
	Medium	Y	Y	Y
	Low	Y	Y	N
1	High	N	Y	N
	Medium	Y	Y	N
	Low	Y	Y	Y

The authors in (VAN et al., 2013) implemented their solution in the reference HM encoder 7 and encoded classes B to F sequences in accordance with the CTC. Compared to a traditional cascaded transcoder, the proposed method achieved an average reduction

of 64.03% with a BD-bitrate increment of 1.76%.

Fast transcoding methods for CU and PU decision are presented in (VAN et al., 2016). The first presented method, called Top-to-Bottom (T2B), starts with the CU at the root of the CTU quadtree and processes the four sub-CUs only if the co-located CU in the reference stream was split. Otherwise, the subdivision is halted. The second method, called Bottom-to-Top (B2T) follows the reverse path, starting at the leaves of CTU quadtree. First, all the leaves are processed. Then, at each iteration, the T2B algorithm checks if the current CU was split in the reference bitstream. If it was not split, it is processed, and if it was, the split flags of its four sub-CUs are evaluated. If they are all zero, it means that there is a chance that the current larger CU is a better option, so it is still evaluated. If at least one of the four sub-CUs was split, the current CU is skipped.

Fast PU decision adaptations were also implemented in each of these methods, using the split flag information to decide whether to compute all PUs (when the split flag is equal to one), or only the 2Nx2N and the best PU of the co-located CU (when the split flag is zero).

Learning-based classifiers were also designed for the top-down and bottom-up strategies, using a set of five features: split flag, delta depth between the current CU depth and the max depth of the input CUs, sum of depths, number of PUs, and the Coded Block Flag (CBF) of the CUs in the reference stream. With this data, Decision Trees were trained using the C4.5 algorithm, one for each CU size and  $\Delta QP$  combination. The authors worked with three  $\Delta QPs$ , adding up to nine classifiers.

From the proposed schemes, the fastest approach is able to reduce the complexity by 82% with an average BD-BR increase ranging from 1.04% to 2.77%, using  $\Delta QPs$  equal from 2 to 6 in their simulations and VBR conditions. CBR encodings were also performed, in which the same method achieves savings around 81%, with BD-BR increases ranging from 2.47% to 3.01%. One important remark of this work is that the best results were not obtained from the learning-based methods, proving that a simple fast heuristic is sometimes more efficient than a complex classifier to solve a problem.

### 4.3 Final Remarks

Although significant ground has already been covered for fast HEVC encoding methods, there is still much room for innovation. Recent works revealed the potential of learning-based methods to achieve complexity reduction of HEVC encoders, but the

methodology can be improved by taking into account the compression efficiency in the training process. The methods of (CORRÊA et al., 2015) prove that combining fast CU, PU and RQT partitioning decision methods can improve the complexity reduction with tolerable compression penalty, but the complexity analysis presented in the first section of this chapter indicated that the Motion Estimation searches also need to be addressed. The fast encoding methods proposed in this thesis aim at filling this gap by using fast Motion Estimation decisions, while increasing the efficiency of the fast CU, PU and RQT methods compared to competing solutions.

The transcoding solutions demonstrated that using a reference HQ bitstream can be extremely useful to guide the decisions of subsequent encodings of the same sequence. This is an important discovery for adaptive streaming applications, and will be explored in this thesis. It was also shown that limiting the Upper and Lower Bounds of the CTU quadtree during the CU partitioning decision can provide significant reduction in complexity, but the corresponding solution requires the production of an extra reference bitstream to accomplish this. It is preferable to use a single reference to reduce the execution pipeline of the proposed method, specially if real-time transcoding is required. In addition, the works of (SCHROEDER et al., 2016) and (VAN et al., 2016) let us conclude that fast heuristics can be also efficient for fast transcoding, and the latter reference even shows that they outperform learning-based approaches. However, the set of features used in (VAN et al., 2016) is limited, which might explain the reduced performance of their classifiers. The transcoding methods proposed in this thesis will make use of the positive aspects of these works while trying to mitigate the drawbacks of each case.



## 5 ANALYSIS OF HEVC ENCODING AND BITSTREAM INFORMATION

In its essence, Machine Learning algorithms are capable of building predictive models based on observed measurements (or features), which can or not be associated with an output value (or label). Therefore, the relevance of such measurements is a key component for a successful implementation. In many cases, the relationship between each feature and the modeled task is known, so an expert can easily identify the ones that are useful and the ones that can be discarded without affecting model performance. For instance, when building a predictive model for face detection, it is expected that pixel-based features and edge detection measurements will be helpful in the classification and should not be disregarded.

There are also cases when a feature does not have any apparent relation with the modeled task, but it is identified as relevant information by the training algorithm. Detecting these hidden relationships is an important motivation of using Machine Learning techniques, as these discoveries might help us to better understand complex problems.

It is possible to design new features as well, which is sometimes referred to as Feature Engineering. For instance, linear combinations of two or more features, scaling, and even data transformation methods can be used to design novel features that could prove to be more useful than the original ones.

Regardless of the method used to extract features, it is important to assess how they relate with the output variables to discard irrelevant or noisy ones and to discover hidden relationships. Some applications also have complexity or storage constraints, so a feature analysis can assist on how to reduce the feature space to alleviate these issues.

In the following sections, we will discuss which information can be extracted from HEVC encoders and decoders, and then assess the importance of this information in the classification problems addressed in this thesis. Initially, the setup and methods used in this analysis are detailed. Afterwards, the encoding features will be assessed with the encoding partitioning decisions, whereas the decoding ones will be evaluated with the transcoding modes.

The sequences listed in Table 5.1 were used in all the analyses presented in this chapter. This is a subset of the sequences defined by the JCT-VC in their Common Test Conditions document (BOSSEN, 2012).

We used a wide range of sequences with varying spatial/temporal resolutions and motion characteristics to increase the quality of our analysis. A detailed description of

Table 5.1: Video sequences used in the analysis

<b>class</b>	<b>sequence</b>	<b>resolution</b>	<b>fps</b>	<b>bit depth</b>
A	Traffic	2560x1600	30	8
	NebutaFestival	2560x1600	60	10
B	BasketballDrive	1920x1080	50	8
	ParkScene	1920x1080	24	8
C	BQMall	832x480	60	8
	PartyScene	832x480	50	8
D	BlowingBubbles	416x240	50	8
	RaceHorses	416x240	30	8
E	Vidyo1	1280x720	60	8
F	SlideShow	1280x720	20	8

each sequence is presented in Annex A.

## 5.1 Feature Analysis for the HEVC Encoding Decisions

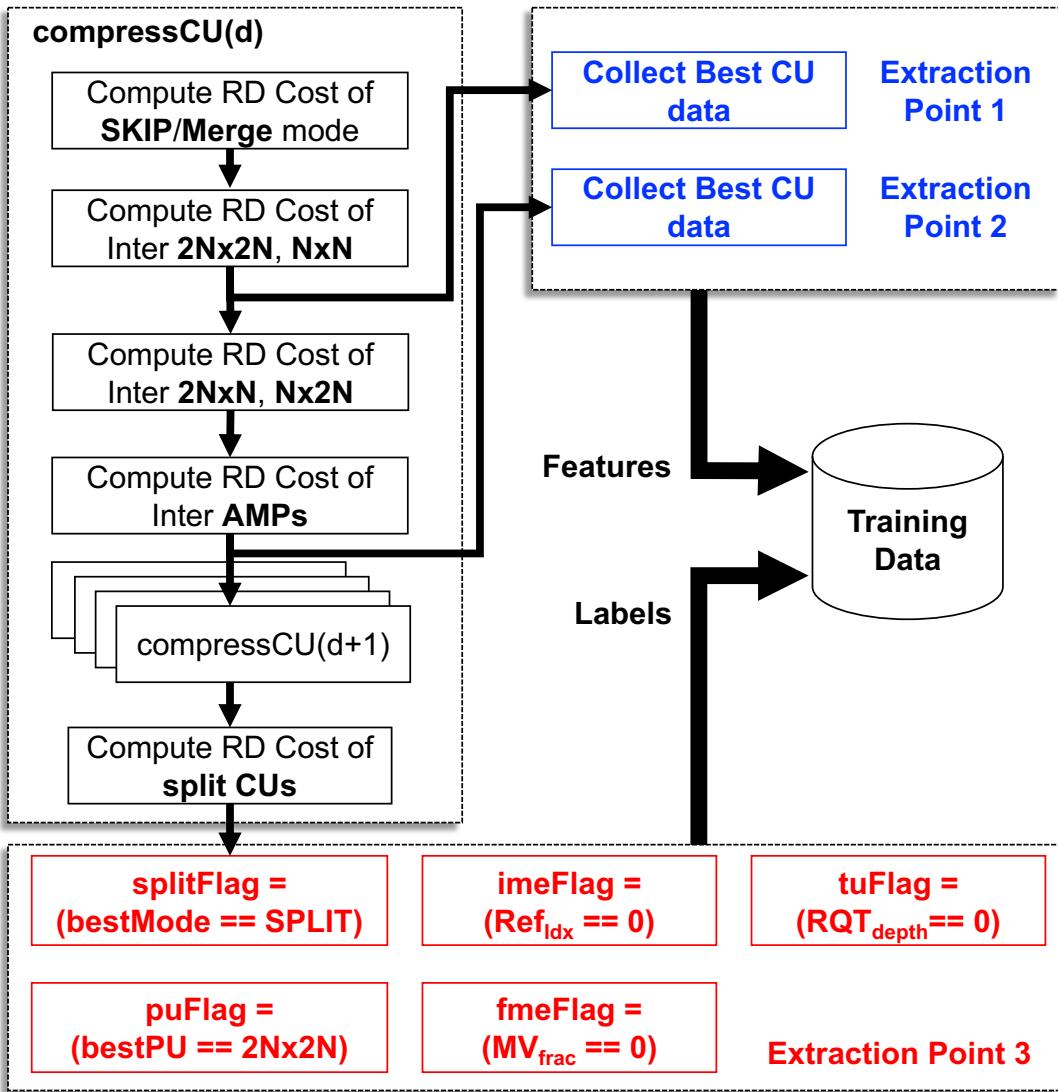
Our first set of experiments aimed at discovering which information can be used to efficiently predict the best partitioning of the HEVC structures, namely the CU, PU, and TU partitions. Therefore, we collected several encoding-domain variables, such as rate-distortion costs, collocated depth and so on. The flowchart in Figure 5.1 shows the CU/PU/TU mode decision algorithm implemented in the HM 16.8 (BOSSEN; FLYNN; SÜHRING, 2013) and the points at which data was extracted.

Three Extraction Points were defined. The first two are responsible for extracting the encoding information. In the framework of Machine Learning, this corresponds to the features that will be used to train our classifiers. The last point corresponds to the extraction of the final decisions, which are equivalent to the labels of each classifier.

At Extraction Point 1, the information is collected after the SKIP/Merge and 2Nx2N inter-prediction are computed. This is useful for fast decisions that will affect the current depth of the CTU quadtree, because up to six more PUs are going to be computed, including the RQT evaluation. It is also possible to extract data before this point, but important information from the inter-prediction would be unavailable.

The CU data was extracted at a later point, after all PUs were evaluated (Extraction Point 2), because the decision to split or not a CU will only affect the lower depths of the CTU quadtree. It is important to note that the features extracted at point 1 can still be

Figure 5.1: Data collection scheme for the HEVC encoding decisions.



used for the partitioning decision, but the information is more reliable at point 2, because the actual best mode of the current CU is available.

Finally, the final CU, PU and TU decisions are collected at Extraction Point 3. The CU split label has two values: when the rate-distortion cost of the four sub-CUs in the next depth is less than the cost of the current depth, it is assigned to 1 (*True*), otherwise it is 0 (*False*). The PU labels were initially assigned as the PU partitions themselves, so it has eight possible values. Finally, the TU labels are the RQT depth, which can assume values from 0 up to 3.

A set of 54 features (see B) was extracted for this analysis and for training the fast encoding decision classifiers. They can be categorized into the following groups:

- **Coding Flags:** this group contains information that is either used or produced in the encoding process, and which can be directly accessed from the encoder software

without extra computations. To exemplify: prediction mode, PU partition, RQT depth, CTU depth of the temporal co-located CUs etc.

- **Coding metrics:** these are the metrics used in the rate-distortion computations, such as number of encoded bits, distortion, rate-distortion cost, SSE, and number of non-zero coefficients.
- **Motion cues:** contains the values related with the Motion Estimation process, including magnitudes of the integer and fractional portions of the ME vector, FME precision (full pel, half pel or quarter pel), and reference frame index.

The following sections of this chapter will only mention the most noteworthy ones on each discussion, but a complete list of these features can be found in Annex B. Most features are extracted directly from the encoding variables, but some were designed specifically to be used in the training of classifiers: the ratio between the rate-distortion costs of two modes (CostRatio), and the normalized difference, also between two costs (NormCostDiff). The following equations detail how these derived features are computed:

$$CostRatio_{m1,m2} = \frac{RDCost_{m1}}{RDCost_{m2}} \quad (5.1)$$

$$NormCostDiff_{m1,m2} = \frac{|RDCost_{m1} - RDCost_{m2}|}{RDCost_{m2}} \quad (5.2)$$

$$\Delta CtxDepth_{CTU} = CtxDepth_{CTU} - depth_{curr} \quad (5.3)$$

$$CtxDepth_{CTU} = \frac{\sum^N AvgDepth(N)}{N} \quad (5.4)$$

$$AvgDepth = \frac{\sum^M DepthCU(M)}{M} \quad (5.5)$$

In (5.1), the Ratio between the RD costs obtained from modes  $m1$  and  $m2$  is computed, and its normalized version is computed in (5.2). In (5.3), the average depth of the neighboring CTUs is first computed with (5.4), using the depth of its  $M$  constituent CUs, as in (5.5). Then, the average of these averages is computed and subtracted from the depth of the CU being currently encoded, yielding the  $\Delta CtxDepth_{CTU}$  value. The neighboring CTUs include four spatial neighbors and up to two temporal neighbors (one from each reference list), so up to 6 neighbors are used in the calculation of  $\Delta CtxDepth_{CTU}$ .

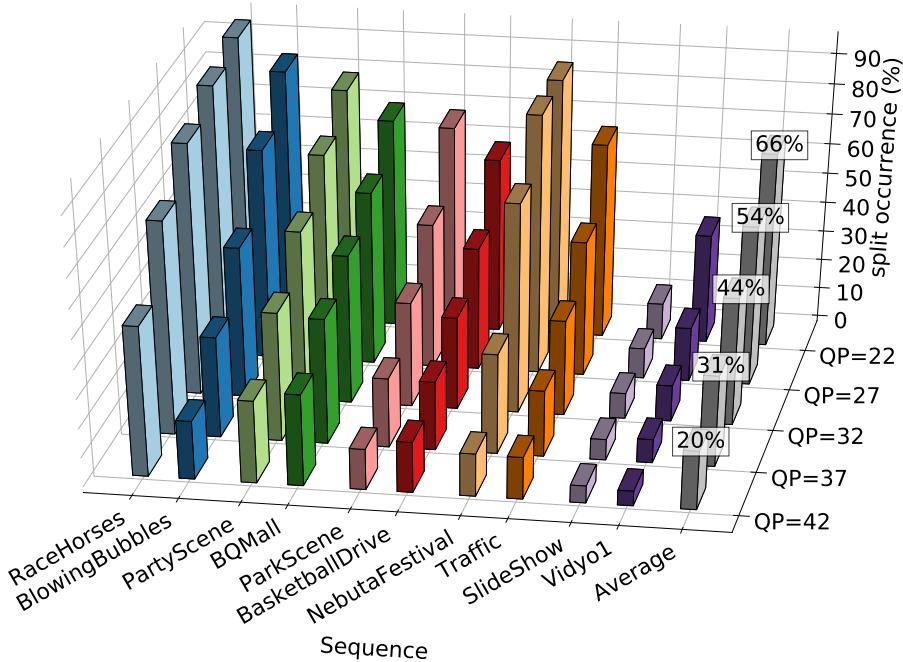
In addition to these, two other features ( $CtxDepth_{CU}$  and  $\Delta CtxDepth_{CU}$ ) have similar formulations, but they use the average depth of neighboring CUs instead of the entire CTU.

In every analysis of this thesis, the input data were randomly sub-sampled until the distribution of the class attribute is balanced, i.e., until the number of examples of each class was the same. This is important to ensure the analysis is not biased towards the decision of the majority class.

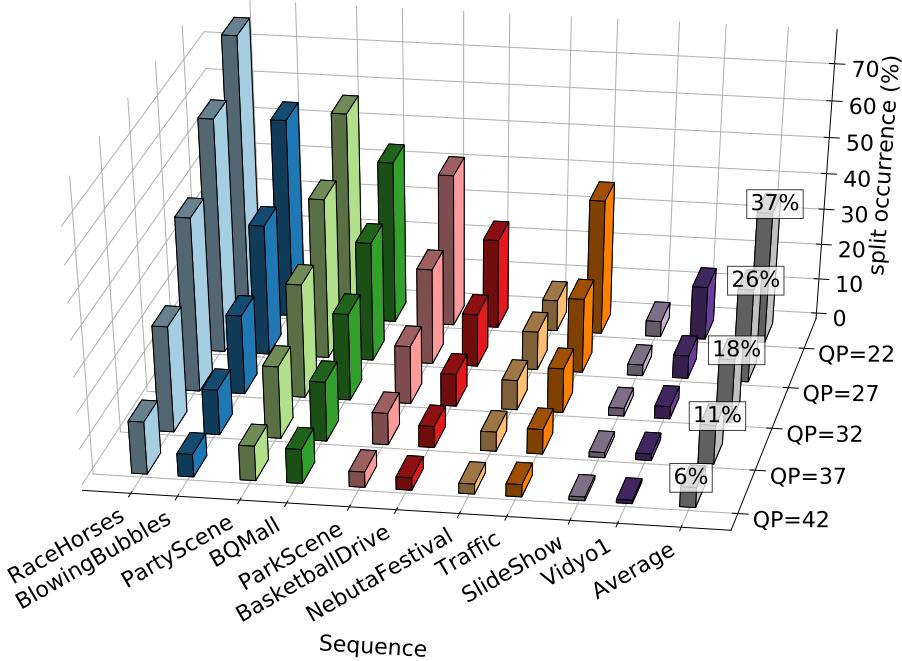
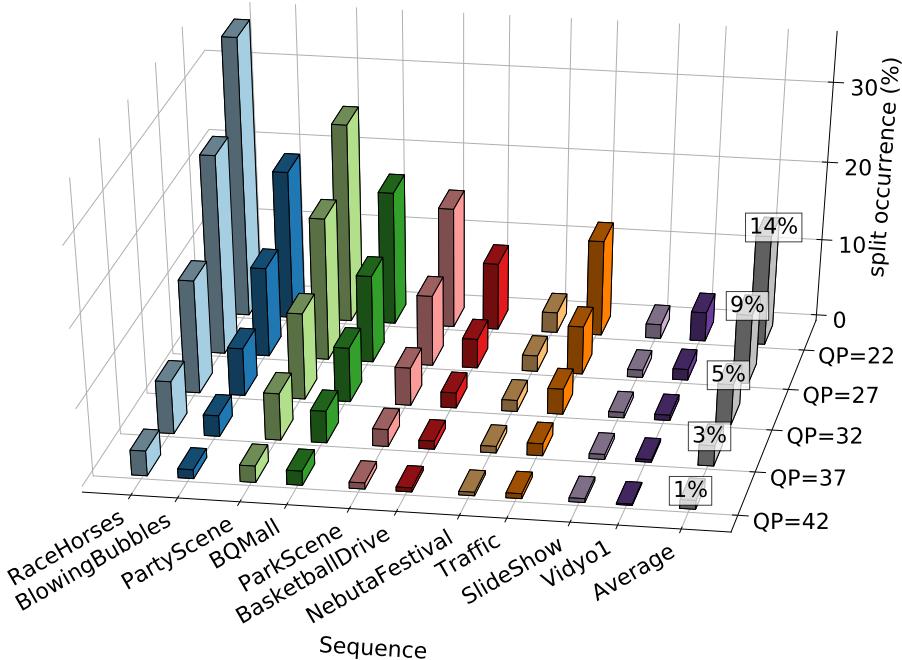
### 5.1.1 CU Partitioning

The CU partitioning decision has a lot potential for saving encoding time, because the PU and TU decisions are nested inside each node of the CTU quadtree. Therefore, this was the first and most scrutinized part of this research. Figures 5.2, 5.3, and 5.4 show the distribution of the CU partitioning for each sequence and QP value. Each figure represents a particular CU size, from 64x64 down to 16x16. The 8x8 CUs are not accounted for, because that is already the smallest size supported by HEVC. The CUs that were further partitioned in the RDO mode decision are marked as *Split*, and *Not Split* otherwise.

Figure 5.2: Occurrence of CU splits on  $64 \times 64$  CUs.



The plots of these figures lead to important conclusions about the CU partitioning. The first one is that there is a correlation between the QP and the amount of split CUs. Regardless of its size, the odds of a CU being split decrease significantly as QP increases.

Figure 5.3: Occurrence of CU splits on  $32 \times 32$  CUs.Figure 5.4: Occurrence of CU splits on  $16 \times 16$  CUs.

On average, 64.9% of the  $64 \times 64$  CUs are split with a QP of 22, and this goes down to 19% when QP is set to 42. The differences are larger for  $32 \times 32$  and  $16 \times 16$  CUs, namely 35.9% down to 5.3% ( $32 \times 32$ ), and 14.6% down to 1.2% ( $16 \times 16$ ). This type of behavior is related with the fact that larger CUs tend to increase the compression efficiency, because less header information (motion vector delta, reference index, etc) is sent to the bitstream.

The second observation is that smaller CUs are much less likely to be split than larger ones. For instance, with a QP of 32, a  $64 \times 64$  CU is 2.4 times more likely to be

split than a 32x32 CU, and 7.42 times more than a 16x16 CU. This is also expected for the same reasons mentioned in the first observation, i.e., larger CUs are more efficient in terms of compression.

A final consideration from Figure 5.2 is that the CU partitioning is not tightly related to the spatial resolution of sequences, but it seems to be more affected by specific scene characteristics. This can be clearly observed when comparing the *RaceHorses* and the *SlideShow* results. The former sequence contains high movement and complex textures in the background, whereas the latter is mostly still, with many homogeneous regions. Therefore, features that are related with these parameters, e.g., motion vectors, are possibly efficient for CU partitioning prediction.

The following paragraphs will discuss the importance of the extracted features to the CU partitioning decision problem, using different metrics: (1) Information Gain Ratio (IGR); (2) F-Score; and (3) Mean Random Forest (MRF) score. The motivation to use IGR is that this metric is used in the C4.5 decision tree building algorithm (QUINLAN, 1986). It is a very efficient metric with improved efficiency compared to the Information Gain (IG) used in the seminal ID3 implementation (QUINLAN, 1986). The F-Score will also be measured, because it is reportedly efficient with SVMs classifiers (CHEN; LIN, 2006). The F-score measures the separation of two sets of real numbers. Given a set of training vectors  $x_k$ ,  $k = 1, \dots, m$ , and the number of positive ( $P$ ) and negative ( $N$ ) values, the F-score of feature  $i$  is calculated as follows:

$$F_{score}(i) = \frac{(\bar{x}_i^P - \bar{x}_i)^2 + (\bar{x}_i^N - \bar{x}_i)^2}{\frac{1}{P-1} \sum_{k=1}^P (x_{k,i}^P - \bar{x}_i^P)^2 + \frac{1}{N-1} \sum_{k=1}^N (x_{k,i}^N - \bar{x}_i^N)^2} \quad (5.6)$$

In (5.6),  $\bar{x}_i$ ,  $\bar{x}_i^P$ ,  $\bar{x}_i^N$  are respectively the average of the  $i$ th feature of the whole, positive, and negative data sets,  $x_{k,i}^P$  is the value of feature  $i$  in the  $k$ th positive instance, and  $x_{k,i}^N$  is the value of feature  $i$  in the  $k$ th negative instance.

Finally, Random Forests have recently been used to measure the efficiency of a set of features, providing the best selection for the final model (VAN et al., 2015). A nice advantage of the MRF score is that it actually build Random Forests using a random subset of the input features at each iteration, so it is capable of detecting the inter-dependencies among them, whereas the other metrics consider each feature separately.

We will start with a detailed IGR analysis in order to answer important questions regarding the quality of our data and to find out how we can handle all this information to maximize the efficiency of the trained classifiers. Ideally, all the information can be used to build a single model for all CU sizes and QP values, but that is not necessarily the case,

because some features might be more useful in a particular case.

The chart in Figure 5.5 shows the top ten features in terms of IGR, using three different approaches: using the samples from all QP values and CU sizes in a single model (Figure 5.5(a)); separating the samples by the QP used to encode the sequences, adding up to five training sets, one for each QP (Figure 5.5(b)); and grouping the samples by CU size, producing three separate models (Figure 5.5(c)).

In Figure 5.5(a), it is possible to see that the most important features for the CU partitioning are related to the rate-distortion values (like  $\text{Bits}_{\text{Best}}$  and  $\text{NormDiff}_{\text{Best,MSM}}$ ) and the partitioning of neighboring CUs (such as  $\Delta \text{CtxDepth}_{\text{CU}}$ ). On the other hand, the least relevant features (which were not included in the chart for visual purposes) are the ones obtained from ME, like motion vector magnitudes. This indicates that the motion complexity is not particularly related with the partitioning complexity.

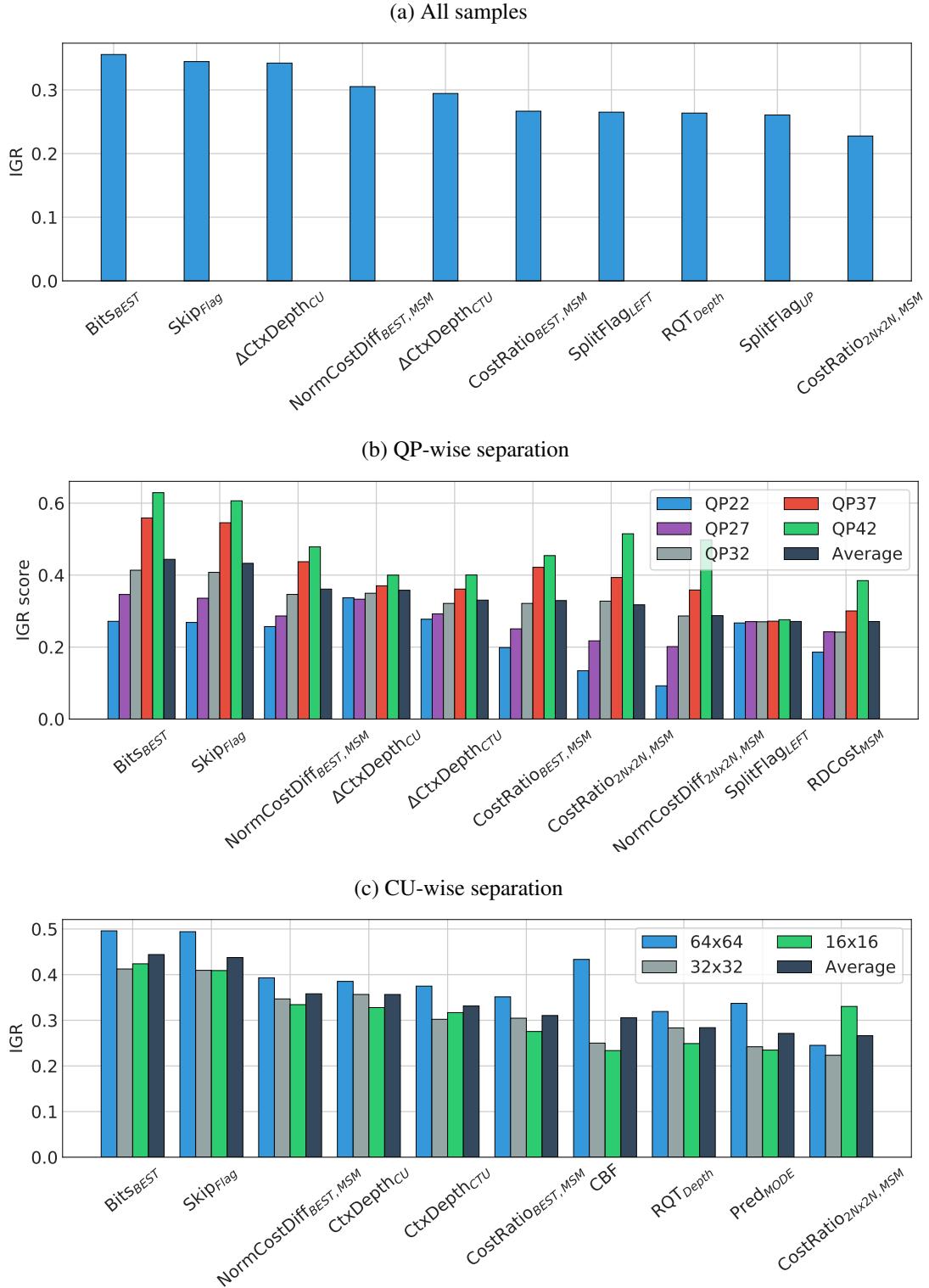
A second important conclusion is that the compound features created with the specific purpose of increasing the quality of our data set ( $\text{NormDiff}_{\text{Best,MSM}}$ ,  $\text{CtxDepth}_{\text{CU}}$  etc) fulfilled their purpose, showing that spending some effort designing new features can be rewarding.

Comparing the three plots of Figure 5.5, we observed that the most relevant features are almost the same regardless of how data is separated, but their scores vary. Separating the data by the QP or CU size increases the IGR of the extracted features, indicating that better classifiers will be trained if we group our input data accordingly. Particularly in Figure 5.5(b), it is clear that the information becomes more relevant as we increase the QP, which indicates that the task of predicting the best CU partitioning becomes easier with larger QPs. The CU-wise scores in Figure 5.5(c) present and analogous behavior, where the larger CUs features tend to achieve higher scores compared to smaller ones.

On average, the CU-wise and QP-wise results presented very similar results: the best feature achieved an average IGR score of 0.444 on both cases, whereas the 10th best feature achieved scores of 0.266 (CU-wise) and 0.271 (QP wise). The average difference between the scores of these two groups of results is -0.003, showing that both approaches are likely to perform similarly when training the classifiers.

In this work, we opted for a CU-wise separation of the training data for a simple reason: if we adopted a QP-wise separation, the training data would have to be produced for several values between the valid range (0 to 51). On the other hand, the CU sizes are standardized and will remain in the 64x64-8x8 range regardless of the encoding configuration. Therefore, the analyses presented from this point on will not consider the other

Figure 5.5: Information gain Ratio of the input features considering three types of data separation: using all samples (a); separating by QP size (b); and separating by CU size (c).



approaches.

Figures 5.6 and 5.7 respectively show the F-Score and the NRF score of the ten best features using each metric.

Figure 5.6: F-Score of the top ten features using this criterion for CU sizes  $64 \times 64$ ,  $32 \times 32$ , and  $16 \times 16$ .

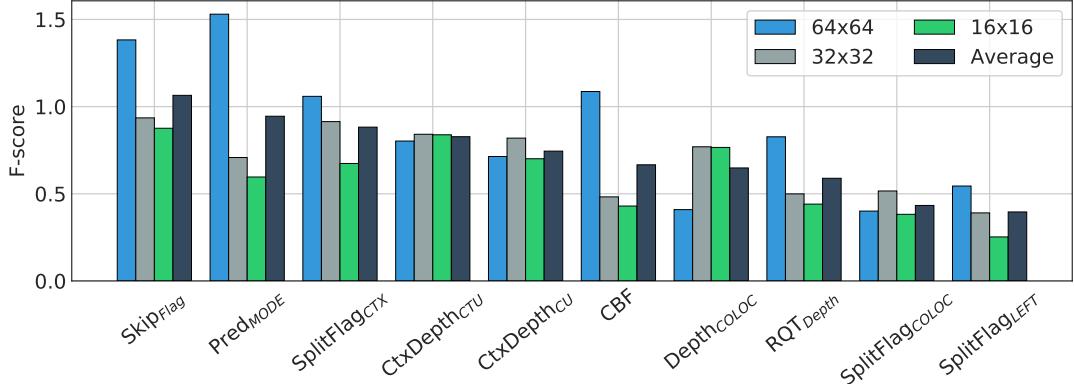
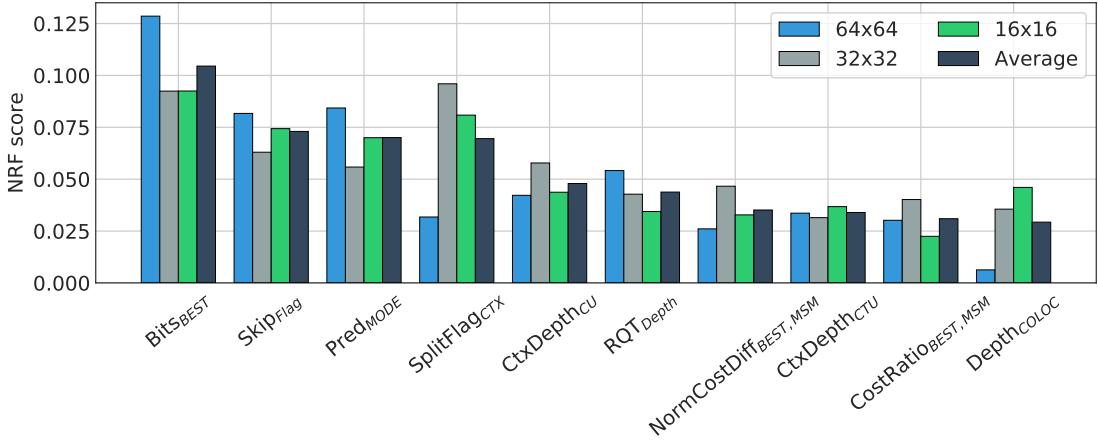


Figure 5.7: NRF score and standard deviation of the top ten features using this criterion for CU sizes  $64 \times 64$ ,  $32 \times 32$ , and  $16 \times 16$ .



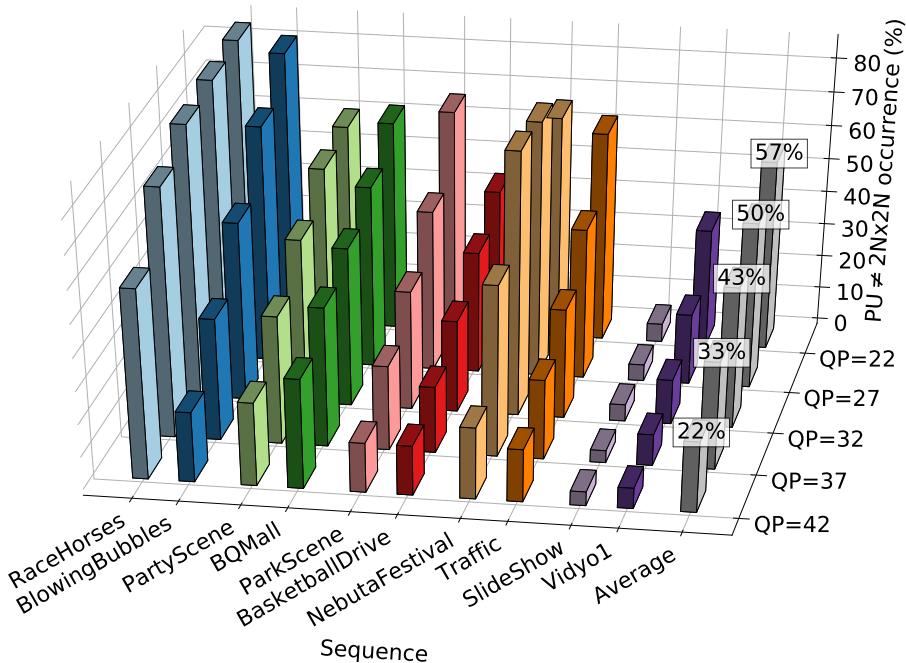
Note that the F-Score seems to favor discrete features compared to the IGR score. For instance, the Bits<sub>Best</sub> was the best feature in both IGR and NRF rankings, but it is not even among the ten best scores with this metric. In fact, relevant features in terms of IGR and NRF like NormDiff<sub>Best, MSM</sub> and Ratio<sub>Best, MSM</sub> are among the least relevant ones in the F-Score results.

The NRF results are similar to the ones obtained in the IGR analysis. This is expected, as Random Forests are actually a group of Decision Trees with some degree of randomness in the feature selection step of the training algorithm. As these results did not provide new insights in this analysis, the NRF scores will be omitted in the following sections of this chapter.

### 5.1.2 PU Partitioning

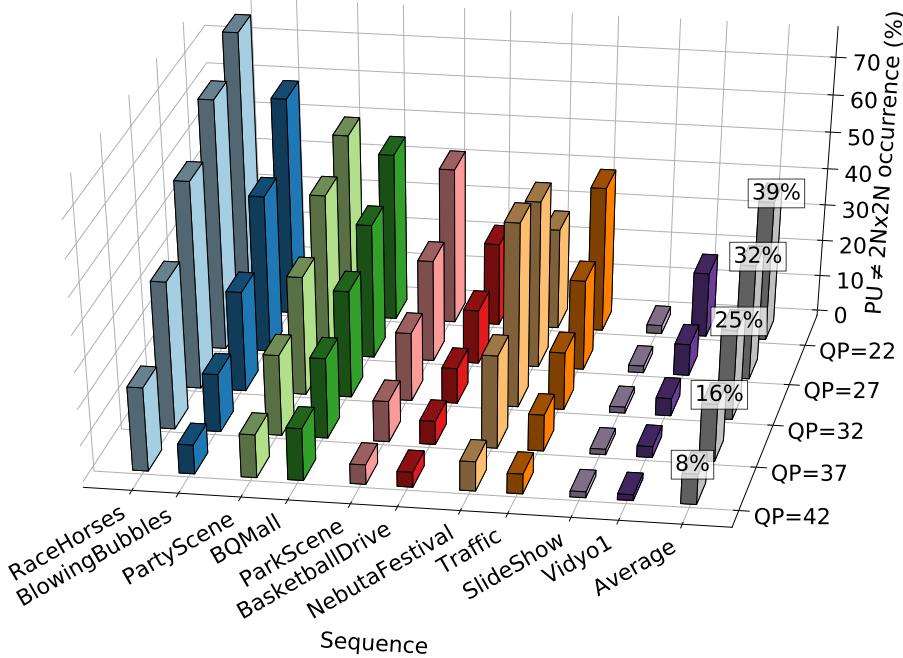
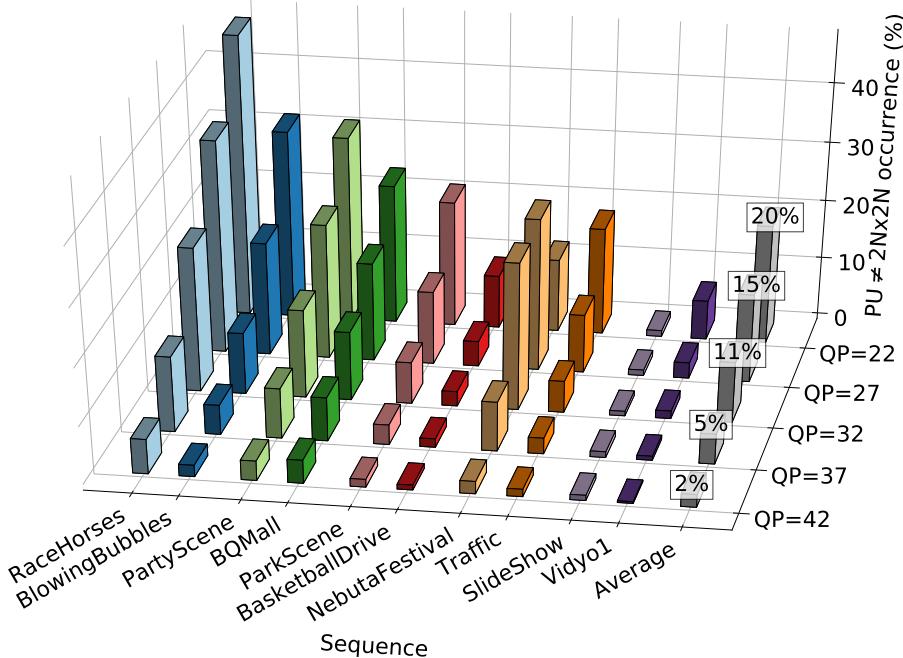
Since there are up to seven possible PU partitionings for each CU, predicting the exact one is very hard, so it might be interesting to relax this problem. In our initial analyses, we realized that most PUs are encoded as  $2Nx2N$ , most likely because these parts require less overhead with header information. Therefore, our first analysis presented in Figures 5.8, 5.9, 5.10, and 5.11 shows the statistics of a PU being encoded as any part different from  $2Nx2N$  for CU sizes of  $64\times64$ ,  $32\times32$ ,  $16\times16$ , and  $8\times8$  respectively.

Figure 5.8: Occurrence of non- $2N\times2N$  PUs on  $64\times64$  CUs.



The charts show that  $2Nx2N$  are clearly predominant. Note that scene complexity also takes a crucial part in the PU partition decision: *RaceHorses* is once again the sequence with the highest occurrence of non- $2Nx2N$  PUs, achieving a peak occurrence of 85% on  $64\times64$  CUs with a QP of 22. In contrast, the *SlideShow* results presented a highest occurrence of 5.4% in the same circumstances. The effects of QP and CU size are also noticeable: for  $64\times64$  CUs, the average occurrence of non- $2Nx2N$  PUs ranges from 57% down to 22% as the QP is increased from 22 up to 42; and when QP is set to 32, the occurrence goes from 43% on  $64\times64$  CUs down to 3% on  $8\times8$  CUs. Therefore, our labels for PU partitioning were defined as *True*, when the best PU of a CU is encoded as  $2Nx2N$ , and *False* otherwise.

The IGR and F-Score of the top ten features for the PU partitioning decision are presented in Figures 5.12 and 5.13 respectively.

Figure 5.9: Occurrence of non- $2N \times 2N$  PUs on  $32 \times 32$  CUs.Figure 5.10: Occurrence of non- $2N \times 2N$  PUs on  $16 \times 16$  CUs.

In Figure 5.12, the features with highest information for this problem are very similar to the ones obtained in the CU partitioning analysis. The main difference is that we only have  $2N \times 2N$  and SKIP/Merge results in this case, so features like  $\text{NormDiff}_{\text{Best},\text{MSM}}$  are replaced by  $\text{NormDiff}_{2N \times 2N,\text{MSM}}$ . It is also important to highlight that the PU partitioning seems to be easier to predict on  $8 \times 8$  CUs, as the features in this case achieved a higher overall score. This probably comes from the fact the  $8 \times 8$  CUs cannot be encoded as AMP partitions in the HM software, reducing the input variability.

Figure 5.11: Occurrence of non- $2N \times 2N$  PUs on  $8 \times 8$  CUs.

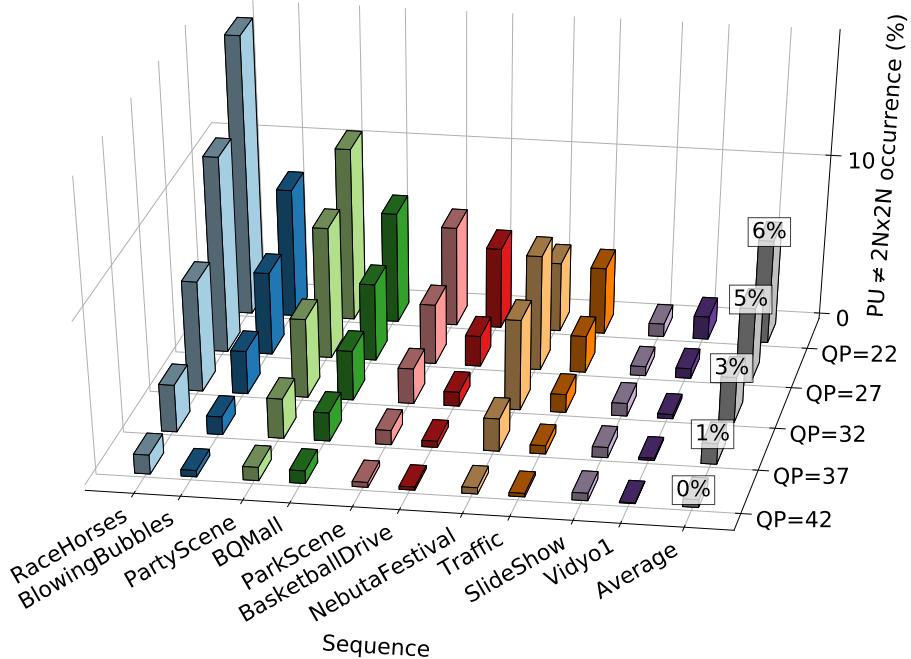


Figure 5.12: IGR score of the top ten features for the PU partitioning problem (CU sizes  $64 \times 64$ ,  $32 \times 32$ ,  $16 \times 16$ , and  $8 \times 8$ ).

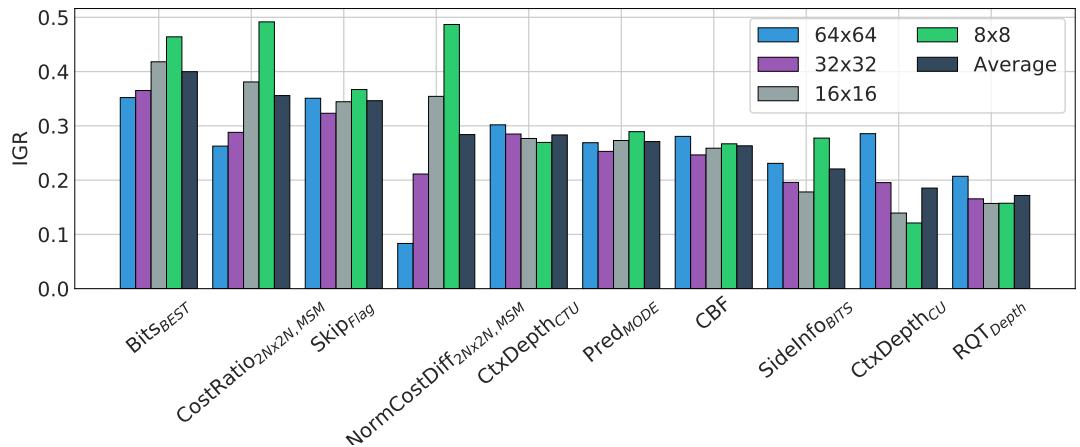
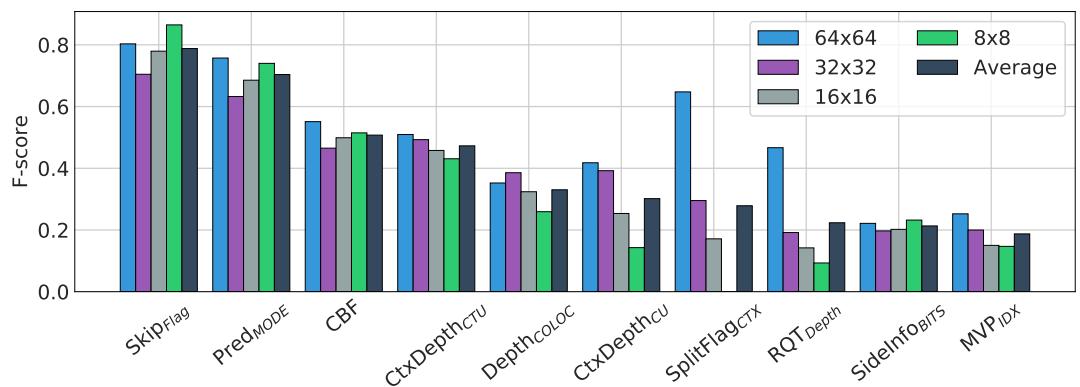


Figure 5.13: F-score of the top ten features for the PU partitioning problem (CU sizes  $64 \times 64$ ,  $32 \times 32$ ,  $16 \times 16$ , and  $8 \times 8$ ).

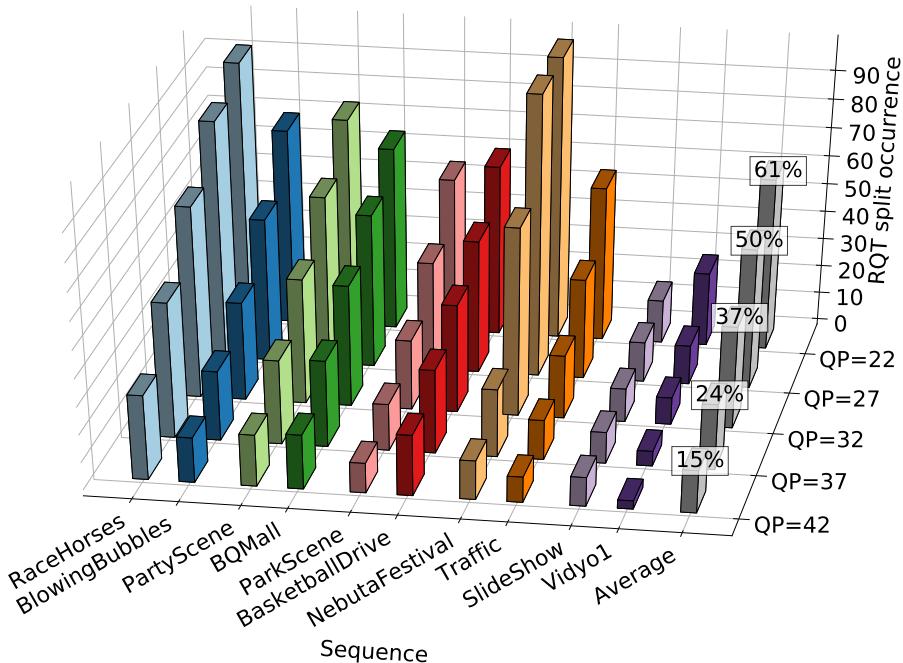


As expected, the F-scores depicted in Figure 5.12 favored discrete attributes, resulting in a very different set. Note also that the  $\text{SplitFlag}_{C_{tx}}$  attribute is useless on 8x8 CUs, as it has a constant value of 0 for this case (see the formula in Annex B to understand this). However, this feature is useful for other CU sizes, so it is still relevant.

### 5.1.3 RQT Partitioning

In the HEVC Model implementation, the RQT partitioning decision is performed for each PU. The root of the RQT corresponds to the largest TU, which is the same size as the current CU (except for 64x64 CUs, in which case the largest TU size is 32x32). In most cases, the best RQT partitioning is at the root of the RQT, because larger transforms tend to increase the compression efficiency of the input residual blocks. With that in mind, we decided to label the RQT samples as: *True* when the best partitioning is below the RQT root, and *False*, when it is at the root node (no partitioning). The charts presented in Figures 5.14 to 5.17 depict the occurrence of RQT splits for all tested sequences and QP values.

Figure 5.14: Occurrence of RQT splits on 64×64 CUs.



The behavior observed in these charts is similar to that of the CU and PU partitionings. A special remark is that the QP has a greater impact in this case: on 64x64 CUs, the highest average occurrence of RQT splits is 61.4% when QP is set to 22, and this is reduced to 15.4% with a QP of 42, yielding a reduction factor of 3.9×. In the case of

Figure 5.15: Occurrence of RQT splits on  $32 \times 32$  CUs.

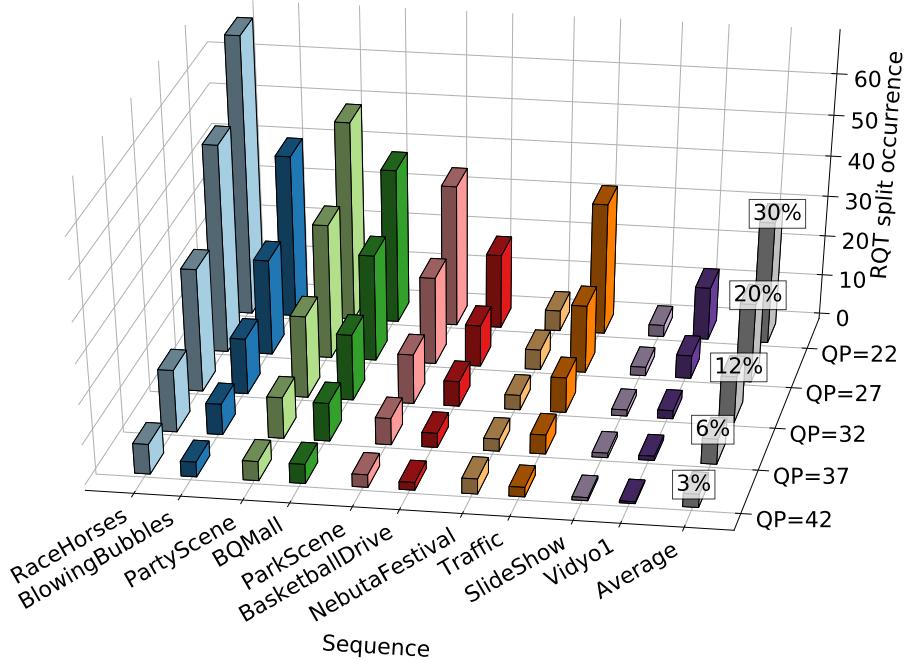
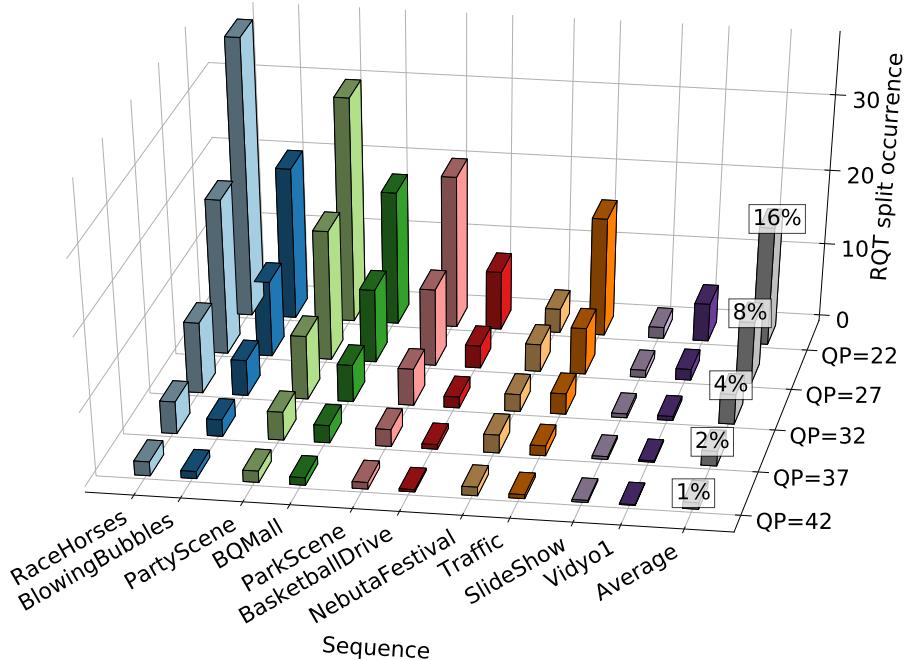


Figure 5.16: Occurrence of RQT splits on  $16 \times 16$  CUs.



8x8 CUs this disparity is much higher though: 8.6% at QP=22 and 0.2% at QP=42, thus a reduction factor of 43×.

Once again we show the IGR and F-score the features in Figures 5.18 and 5.19, this time for the RQT partitioning.

Here we can observe high correlation between the CBF flag and RQT partitioning, specially in F-score results. The reasoning of this lies on the semantics of the CBF flag, which is *False* when the residue block has no significant coefficients, meaning the

Figure 5.17: Occurrence of RQT splits on  $8 \times 8$  CUs.

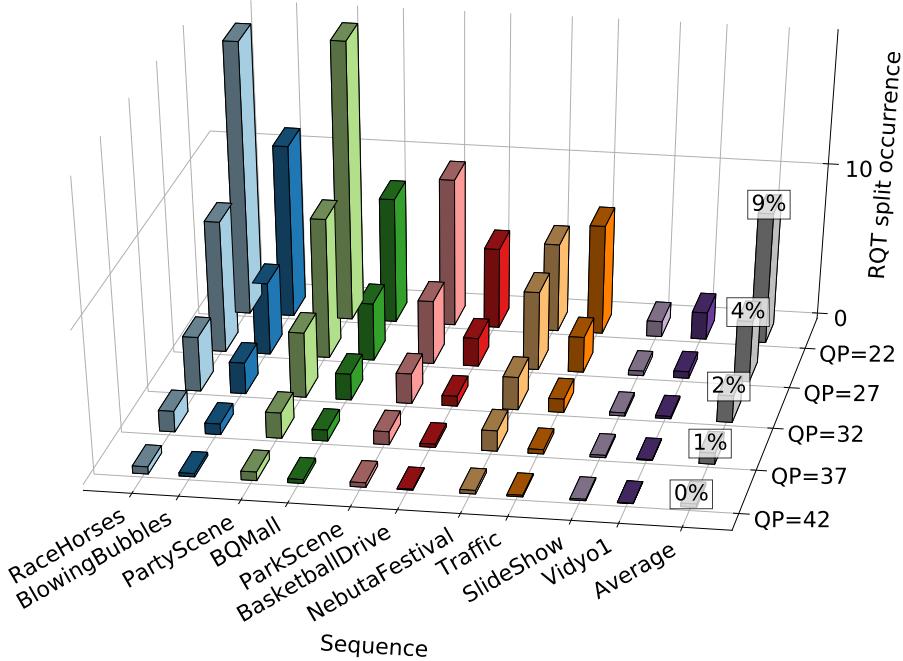


Figure 5.18: IGR score of the top ten features for the RQT partitioning problem (CU sizes  $64 \times 64$ ,  $32 \times 32$ ,  $16 \times 16$ , and  $8 \times 8$ ).

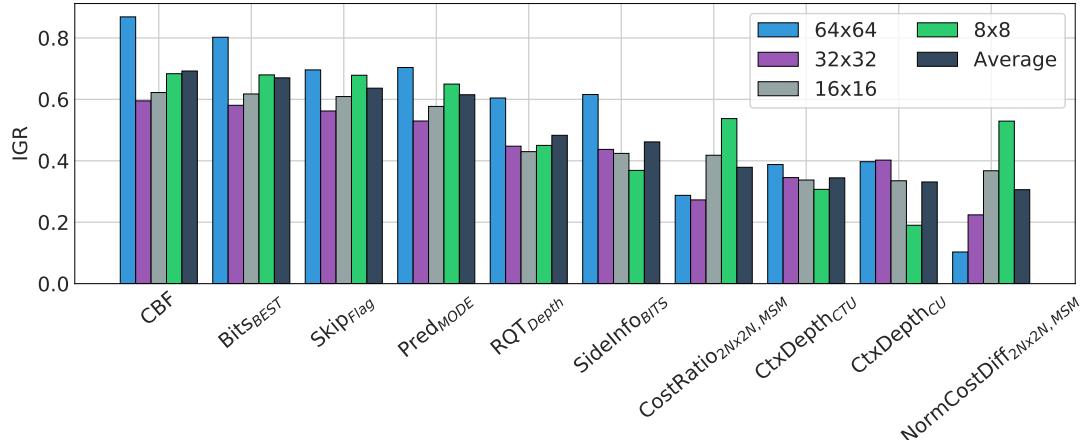
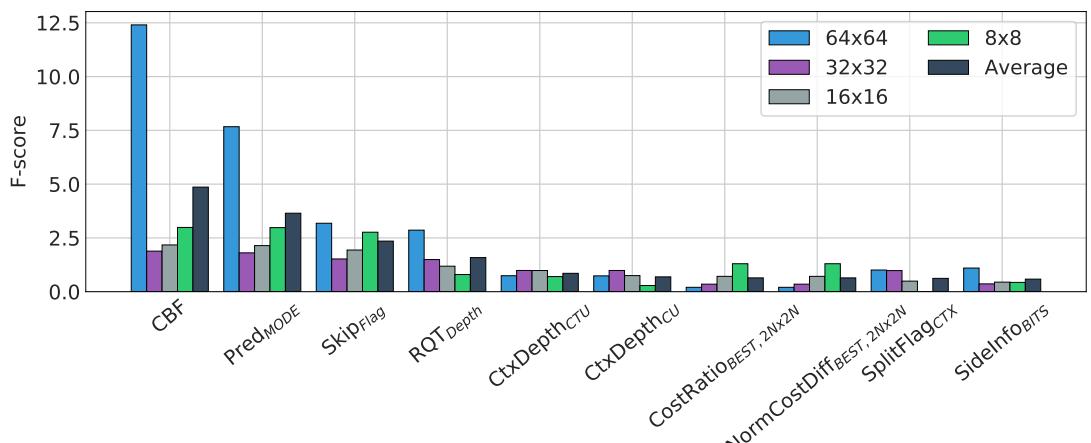


Figure 5.19: F-score of the top ten features for the RQT partitioning problem (CU sizes  $64 \times 64$ ,  $32 \times 32$ ,  $16 \times 16$ , and  $8 \times 8$ ).



transforms will either be as large as possible or even skipped, if such mode is enabled.

### 5.1.4 Reference Frame Index

In order to decrease the IME complexity, the first solution that comes to mind is to skip the search algorithm altogether with the assistance of a motion vector predictor. Since the range of motion vectors depends on an input parameter (namely the search range), it would not be possible to define a classification model for this task. Therefore, to predict the motion vector using ML, one would have to rely on regression models. A second approach was adopted in (CORRÊA et al., 2017), in which the authors skip some steps of the TZ Search algorithm to reduce the IME time. In this work, we decided to follow a different approach: reducing the number of reference frames that are searched in this process. In the default configuration of the HM software, up to four reference frames are searched, so 75% of the IME time is saved if one can predict which reference frame is the most suited one and disregard the others. However, the number of reference frames is also arbitrary, so we further relaxed this problem into two groups: when the reference frame is the first one in the list of references (thus with a reference index of zero), and when it is any of the other references, with an index higher than zero. This approach is interesting, because it works with any number of references. Figures 5.20 to 5.23 depicts the occurrences of the latter for each CU size.

Figure 5.20: Occurrence of non-zero reference frame indices on  $64 \times 64$  CUs.

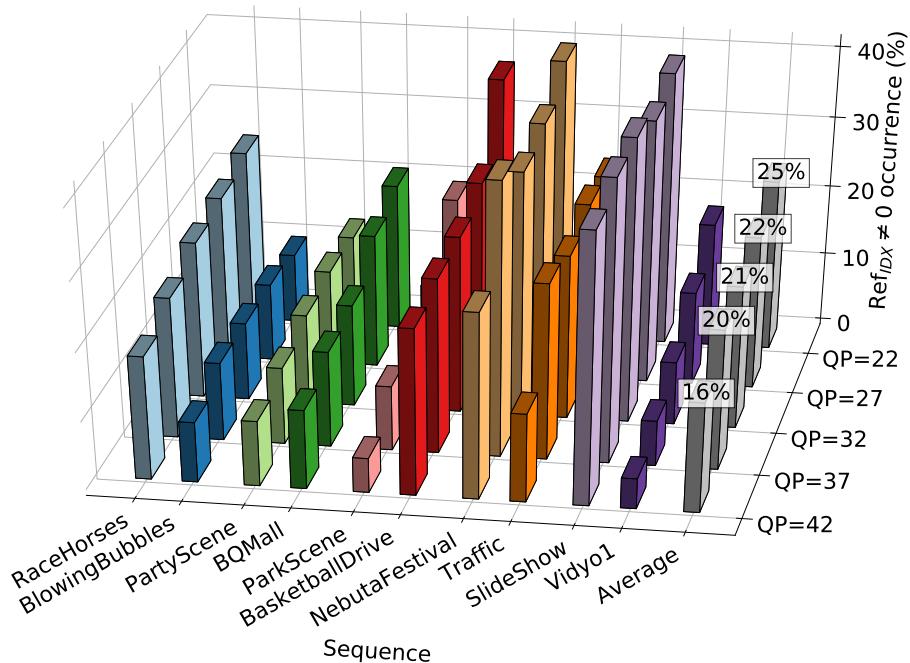


Figure 5.21: Occurrence of non-zero reference frame indices on  $32 \times 32$  CUs.

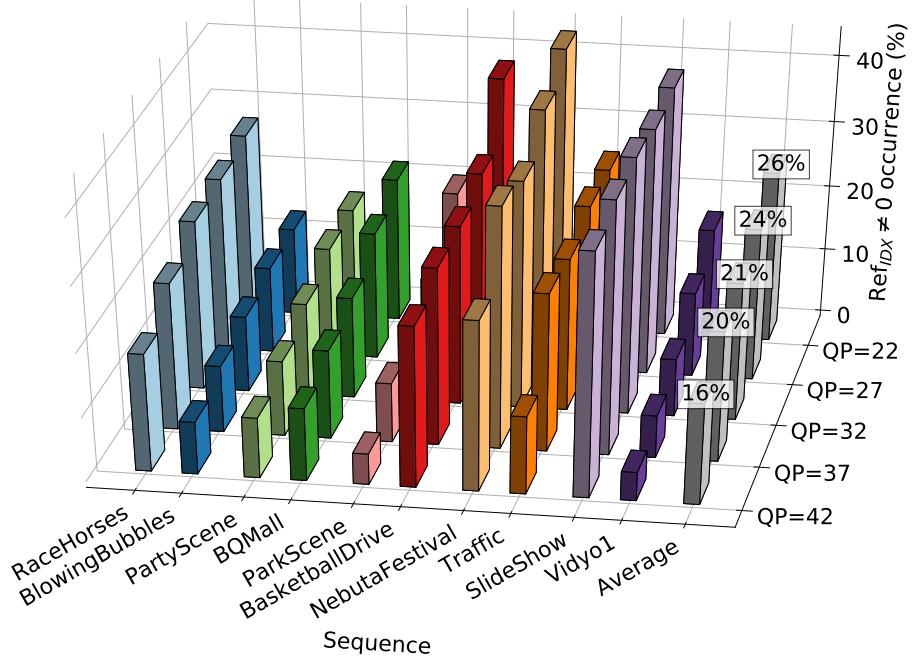
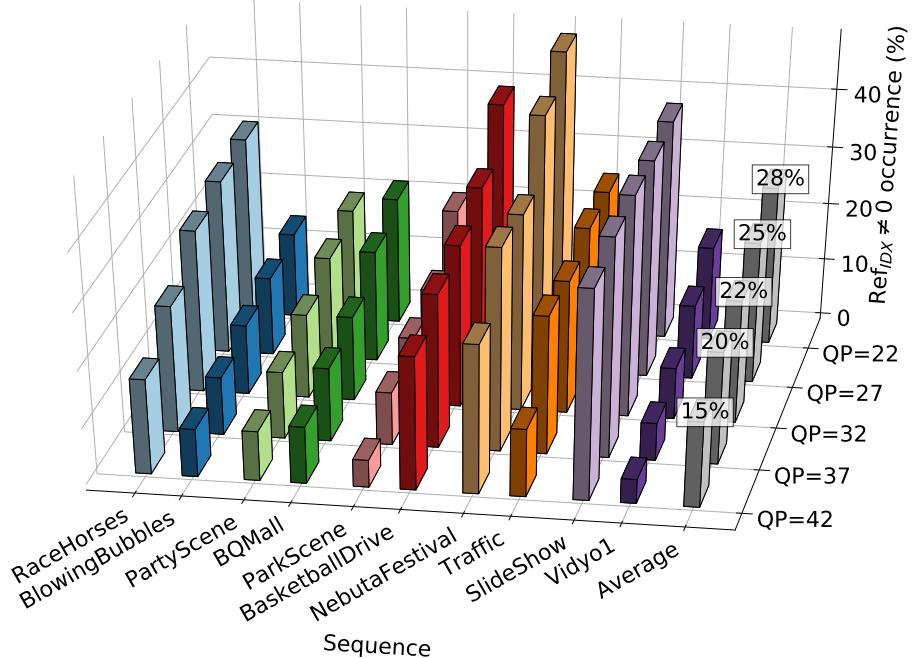
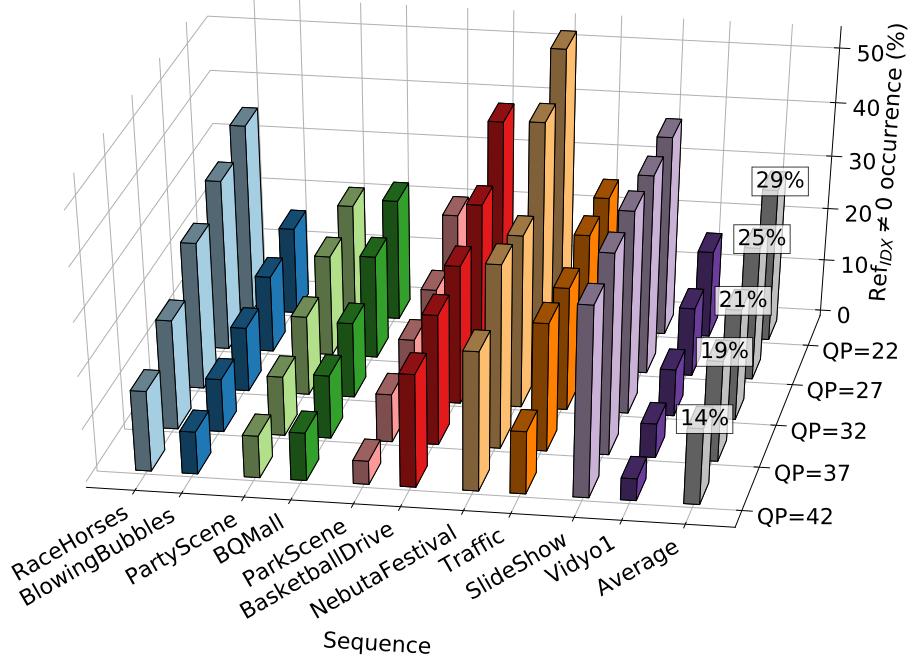


Figure 5.22: Occurrence of non-zero reference frame indices on  $16 \times 16$  CUs.



It is clear that the on most cases, the best reference index is the first one of each list. The main explanation for this is that the first frame in the search is the closest temporal neighbor of the current frame, so it is expected that it is more correlated in general. It is important to note that the effects of QP and CU size are much less apparent here. When QP is increased, there is a small reduction of non-zero reference frame indices, but some sequences (i.e., *SlideShow* and *BlowingBubbles*) are not affected at all. In addition, CU size is almost irrelevant in this case, although a slight increase of non-zero reference

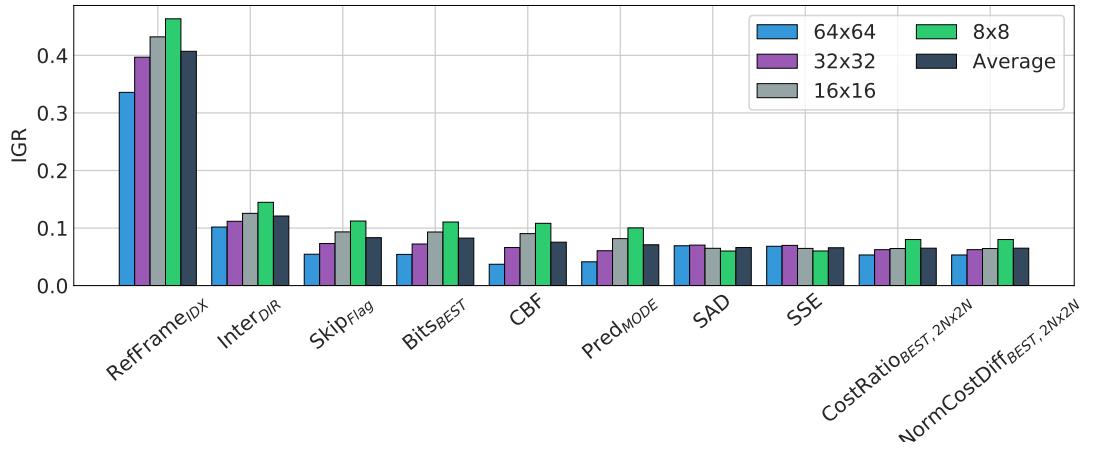
Figure 5.23: Occurrence of non-zero reference frame indices on  $8 \times 8$  CUs.



indices can be observed on smaller CUs.

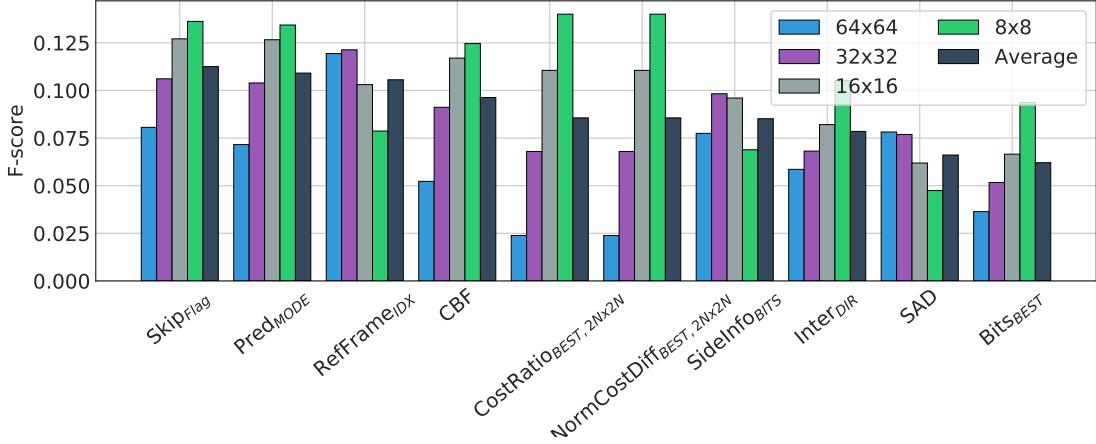
The IGR and F-score of the features for the Reference Frame Index problem are presented in Figures 5.24 and 5.25.

Figure 5.24: IGR score of the top ten features for the ME early termination problem (CU sizes  $64 \times 64$ ,  $32 \times 32$ ,  $16 \times 16$ , and  $8 \times 8$ ).



The scores obtained with our features are considerably lower for this problem. The IGR analysis shows us that only a few of the attributes are notably relevant. The reference index from the  $2Nx2N$  PU and the inter-prediction direction were the only ones to achieve an IGR score higher than 0.1. Low scores are also observed in the F-score analysis, although the discrepancy seems to be smaller. Despite their lower importance, we can see that attributes related to the distortion computation (SAD, SSE) are more relevant for this problem, which is intuitive. This indicates that better features should

Figure 5.25: F-score of the top ten features for the ME early termination problem (CU sizes  $64 \times 64$ ,  $32 \times 32$ ,  $16 \times 16$ , and  $8 \times 8$ ).



be introduced to increase the relevance of our data, most probably ones related with the distortion computation.

### 5.1.5 Fractional Motion Vector

The last analysis of this section aims at evaluating how often the FME is effectively used in the encoding process. To accomplish, the fractional part of the motion vectors was extracted, and the samples were categorized into two groups: when the fractional part is equal to zero (thus FME was not useful), and when it was higher than zero. The occurrence of non-zero fractional MVs is illustrated in Figures 5.26 to 5.29.

Figure 5.26: Occurrence of motion vectors with fractional offsets on  $64 \times 64$  CUs.

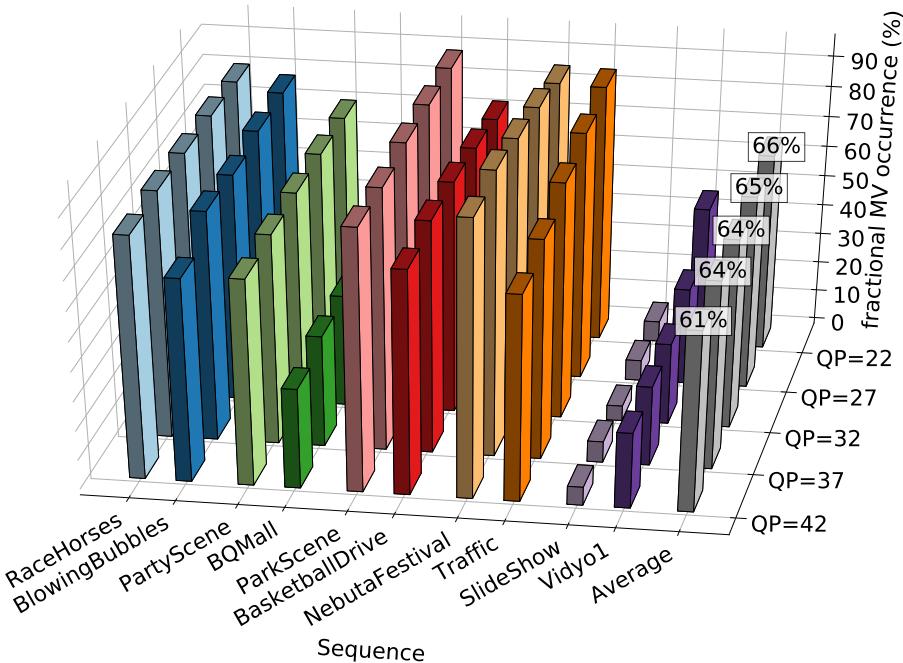


Figure 5.27: Occurrence of motion vectors with fractional offsets on  $32 \times 32$  CUs.

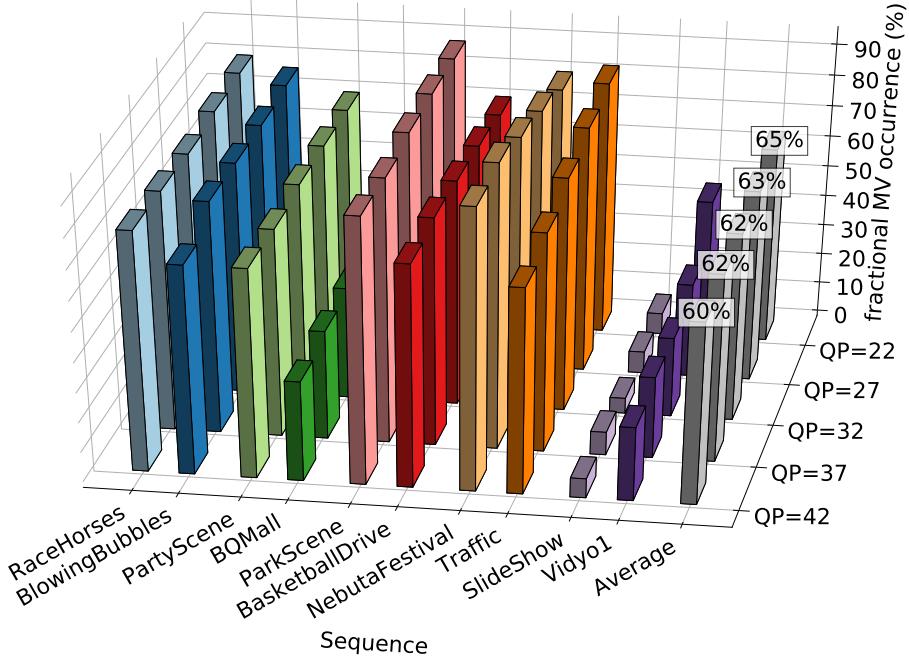
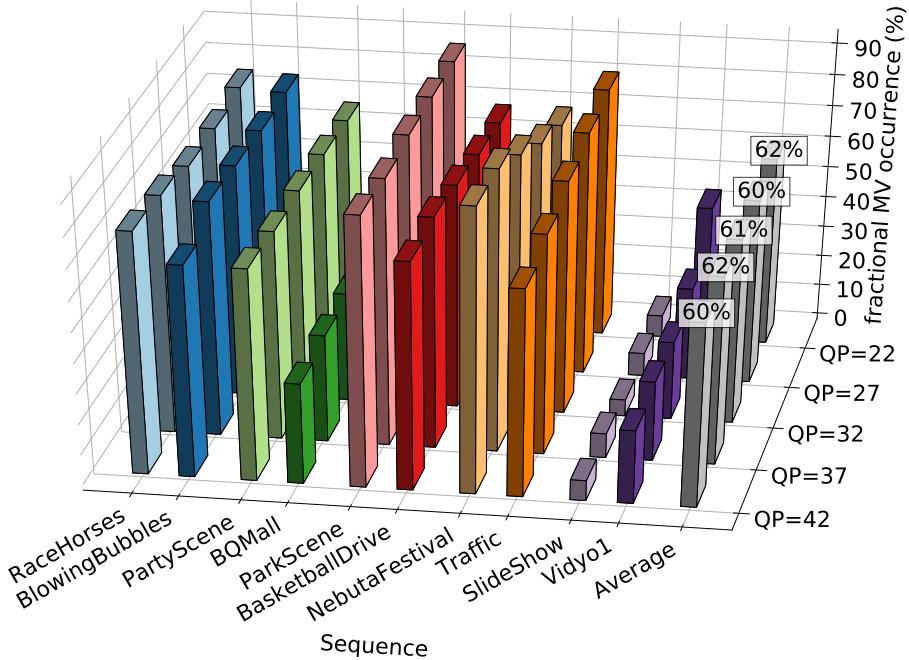


Figure 5.28: Occurrence of motion vectors with fractional offsets on  $16 \times 16$  CUs.



Note that the FME statistics are similar to the ones observed in Reference Index results. QP and CU size do not play a significant role, and scene complexity seems to be the real motive behind the occurrences. Furthermore, most the vectors in our data set contained a fractional part, showing that the time savings potential is inferior compared to the other resources considered in this study.

Finally, consider the charts in Figures 5.30 and 5.31 depicting the top ten features in terms of IGR and F-score respectively.

Figure 5.29: Occurrence of motion vectors with fractional offsets on  $8 \times 8$  CUs.

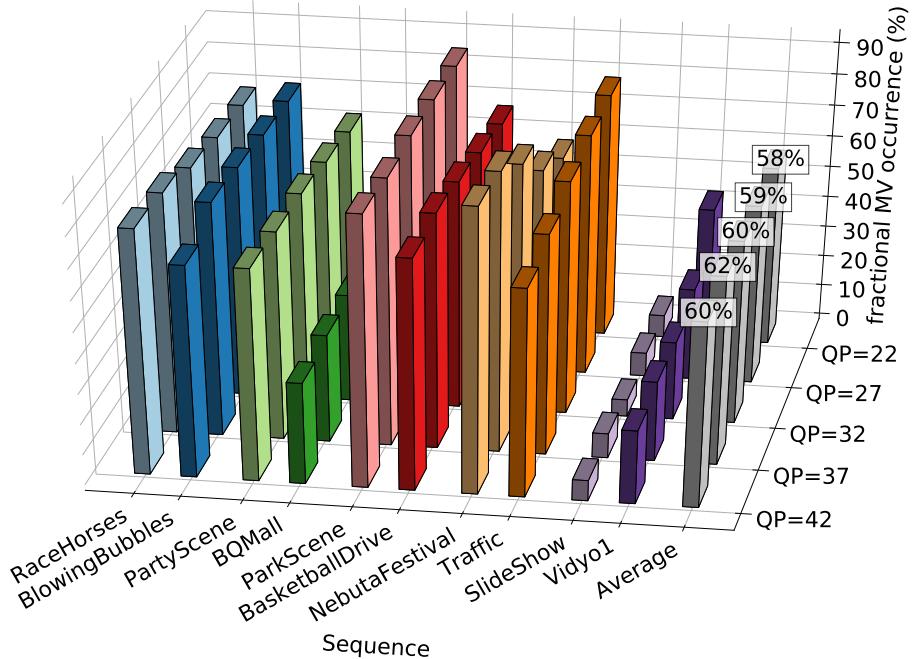


Figure 5.30: IGR score of the top ten features for the FME skip problem (CU sizes  $64 \times 64$ ,  $32 \times 32$ ,  $16 \times 16$ , and  $8 \times 8$ ).

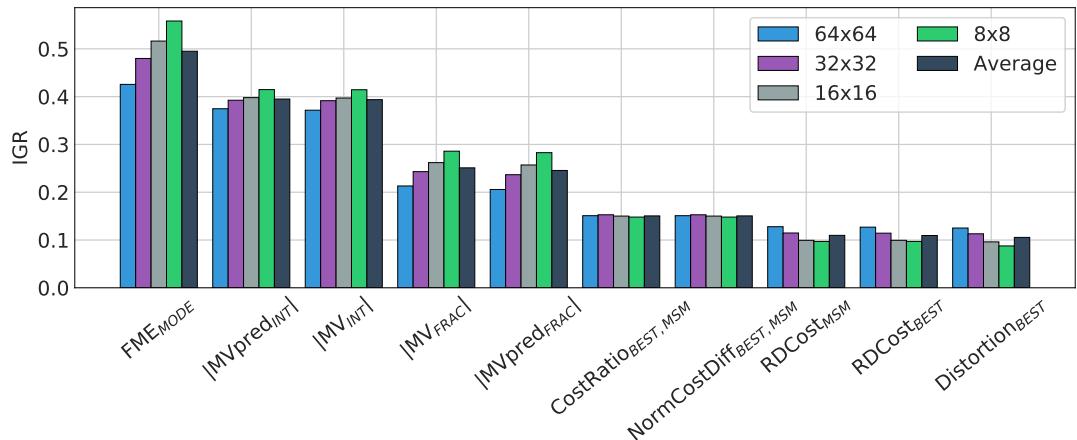
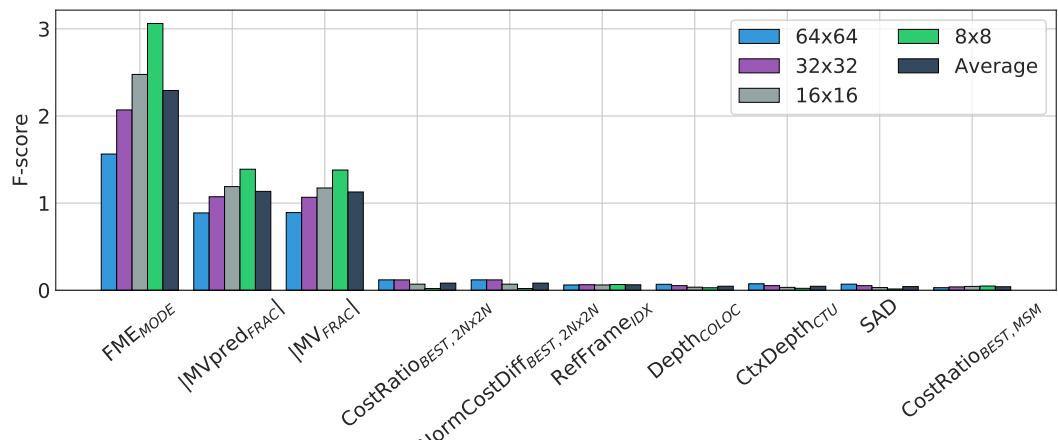


Figure 5.31: F-score of the top ten features for the FME skip problem (CU sizes  $64 \times 64$ ,  $32 \times 32$ ,  $16 \times 16$ , and  $8 \times 8$ ).



While some features presented a fairly high IGR, the results are still lower than the ones observed in the CU, PU and RQT charts. The F-score results are even more contrasting, indicating that at most three of the 54 features are relevant. This shows that the expected performance of the models trained with these features is limited. Nevertheless, this is still an important finding, showing that new features are still to be discovered to predict the occurrence of a fractional offset during Motion Estimation.

## 5.2 Feature Analysis of the HEVC bitstream information for transcoding decisions

The second group of experiments performed during this thesis had the objective of figuring out how the information obtained from a source bitstream could help predicting the decisions of subsequent encoding tasks. The findings of this analysis will be used to elaborate fast decision methods for transcoding-based systems.

Before describing the data collection procedure, it is important to clarify the data preparation. As mentioned in Chapter 1, we envisioned an adaptive streaming scenario, in which a reference, high-quality (HQ) bitstream is used to produce lower quality (LQ) representations that meet each client's processing and transmission budgets. The first thing to decide was a suitable target bitrate for the HQ bitstream: it should be high enough to ensure image quality, but also low enough to reduce storage constraints. The CTC document defines the QP values that must be used, but it does not mention suitable target bitrate values when encoding under ABR or CBR conditions.

Therefore, we adopted the following methodology in our ABR analysis:

1. Encode each sequence with a low QP (we used 22) and get its achieved bitrate ( $BR_{QP22}$ ).
2. Re-encode the sequence using  $BR_{QP22}$  as target bitrate. This will be our high-quality representation with  $BR_{HQ}$ .
3. Generate low-quality versions using reduced bitrate factors of  $BR_{QP22}$  as target bitrate (0.9 \*  $BR_{HQ}$ , 0.8 \*  $BR_{HQ}$ , and so on).

This time, decoding-domain variables were extracted, many of which are also available during encoding: motion vector magnitude, PU partition, CU depth, etc. There are two main differences between the encoding and decoding-domain variables: (i) the former may contain temporary rate-distortion values from the RDO evaluation, whereas the latter only contains the ones actually sent to the bitstream; (ii) the former has access

to pre-quantized residues and other uncompressed data, as opposed to the latter. These differences limit the scope of features that can be extracted from the decoder, but this is not necessarily a major drawback if relevant ones can still be found.

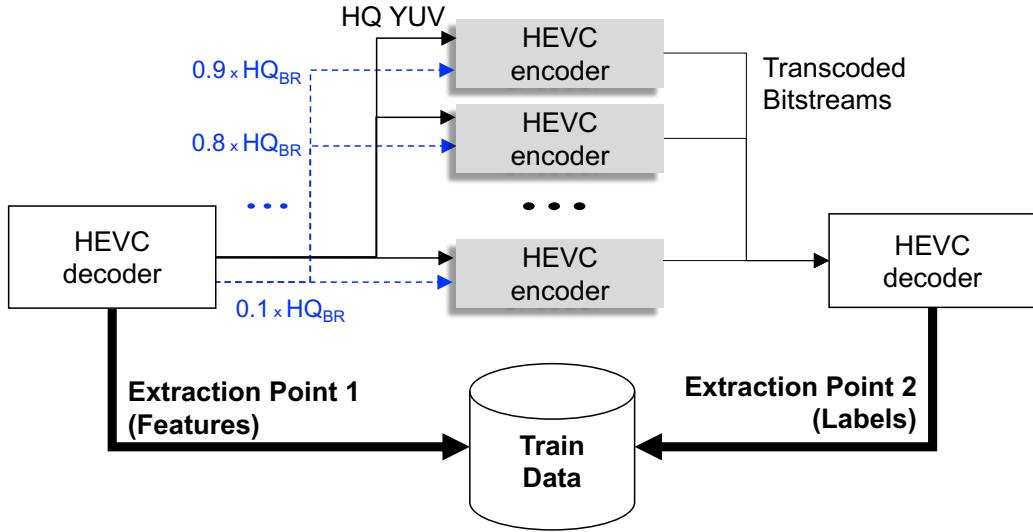
Table 5.2 shows the bitrate used to encode the HQ bitstreams of the training sequences and the assigned bitrates for all the bitrate factors used in this work. The x265 (MULTICOREWARE, 2017) fast HEVC encoder was used using the "placebo" preset in every encoding. The number of frames was fixed to 150.

Table 5.2: Target bitrates (in kbps) used to encode the HQ bitstreams ( $HQ_{BR}$ ) as well as for every bitrate factor used in this work

Factor	Traffic	BsktDrive	ParkSc.	PartySc.	BQMall	BBubble	RaceH.
1 ( $HQ_{BR}$ )	18377	27835	10882	12484	7206	2181	1903
0.9	16539	25051	9793	11236	6485	1963	1712
0.8	14702	22268	8705	9988	5765	1745	1522
0.7	12864	19484	7617	8739	5044	1527	1332
0.6	11026	16701	6529	7491	4324	1309	1142
0.5	9189	13918	5441	6242	3603	1091	952
0.4	7351	11134	4353	4994	2883	873	761
0.3	5513	8351	3265	3746	2162	655	571
0.2	3676	5567	2177	2497	1442	437	381
0.1	1838	2784	1089	1249	721	219	191

The flowchart in Figure 5.32 shows the data collection flow adopted for the transcoding decisions.

Figure 5.32: Data collection scheme for the HEVC transcoding decisions.



In Figure 5.32, the attributes are extracted from the HQ bitstream using a modified FFmpeg decoder (FFMPEG, 2017) at Extraction Point 1. Then, the partitioning decisions are obtained from the LQ representations at Extraction Point 2, also using the FFmpeg

decoder. These will be used to design fast heuristics, and also as labels for fast decision classifiers. Our decision to use the FFmpeg decoder instead of the one implemented in the HEVC Model was motivated by the efficiency of this software in terms of processing time. The FFmpeg implementation is able to decode 2560x1600 in real time, which is an important requirement for transcoding systems that aim to generate the required on the fly in real time.

It is important to highlight that the partitionings of the HQ and LQ versions are different, so the amount of CUs in each case vary. Since our goal is to assign one label (from the LQ bitstream) for each feature vector (from the HQ one), we first had to find a way of ensuring a 1 to 1 correspondence among these data files. To accomplish this, two approaches were used:

1. **CU-grain information:** in the first approach, the information was extracted for each 8x8 CU regardless of the actual CU size. When a decoded CU is larger than 8x8, we simply repeat the data samples to represent the same area of this CU. For instance, a CU that is decoded as a 16x16 block will produce 4 identical feature vectors, one for each 8x8 block in the 16x16 area. Note that this brings redundancy to our data samples, but at the same time it solves the disparity between the data extracted from two different bitstreams.
2. **CTU-grain information:** the CTU data can also be extracted with a 1 to 1 correspondence, as the CTU size is fixed. In contrast with the first approach, where more samples are generated, here we condense the information of all CUs into a single value. The reduction operations used in this thesis were average( $\bar{x}_f$ ), max( $\bar{x}_f$ ) and min( $\bar{x}_f$ ), where  $\bar{x}_f$  is the vector with the values of feature  $f$  for all the decoded CUs in a CTU.

In this work, the CU-grain data was used in fast heuristics to reduce the transcoding time in each CU, whereas the CTU data was applied to train ML classifiers for predicting upper and lower CTU quadtree bounds. The following section will discuss the first set of experiments.

### 5.2.1 CU-grain Analysis

A set of 19 features was extracted from each decoded CU, which can be defined into the following groups:

- **Encoding Decisions:** this group contains the decisions that can be directly accessed in the decoding process. To exemplify: prediction mode, PU partition, prediction mode, CTU depth, CBF of the luminance and chrominance components etc.
- **Decoding metrics:** includes attributes that are calculated from the decoded data, such as sum of coefficients before and after dequantization, sum of the predicted block pixels etc.
- **Motion cues:** composed by the magnitude of the motion vector in both reference lists.

The following results were obtained from the average of all tested bitrate factors (from 0.1 to 0.9). Table 5.3 shows the occurrence of a CU being encoded as size  $\text{Size}_{LQ}$ , provided it was encoded as size  $\text{Size}_{HQ}$  in the HQ bitstream. Each cell is red-shaded with an intensity that corresponds to its value. In other words, high probabilities are strongly red-shaded, whereas low probabilities are practically white-filled.

Table 5.3: CU size probabilities given the size in the HQ bitstream

$\text{Size}_{HQ}$	$\text{Size}_{LQ}$	64x64	32x32	16x16	8x8
64x64	95.7%	3.4%	0.6%	0.3%	
32x32	42.5%	54.5%	2.7%	0.3%	
16x16	21.4%	26.8%	48.0%	3.8%	
8x8	11.8%	19.3%	25.5%	43.4%	

This table shows that there is a high correlation between the HQ and LQ in terms of CU partitioning. The average probability of a 64x64 CU being encoded as 64x64 on LQ representations is above 95%, so a great amount of computations can be saved if this information is provided beforehand. The chances of smaller CUs occur in LQ encodings are very low (at most 3.7%), which indicates that the HQ partitioning information can be used as a lower bound in the LQ CTU quadtree with small losses in compression efficiency. Also note that the chances of a CU being encoded with a larger size is significantly high on average. In fact, the odds of a CU encoded as 8x8 in the HQ bitstream being encoded as 8x8 in the LQ versions is smaller than the odds of it being encoded as a larger block. However, this information alone cannot be used to define upper bound in the CTU quadtree of LQ encodings, because the probabilities of the larger CUs are similar.

The occurrences of a PU being encoded as part  $P_L$ , provided it was encoded as part  $P_H$  in the HQ bitstream are listed in Table 5.4.

Like with the CU sizes, the PU decision is also very likely to select the same partition of the HQ reference, but the larger 2Nx2N partitions seem to be predominant

Table 5.4: PU partition probabilities given the partition in the HQ bitstream

$P_H \setminus P_L$	2Nx2N	2NxN	Nx2N	NxN	2NxnU	2NxnD	nLx2N	nRx2N
2Nx2N	91%	3%	3%	0%	1%	1%	1%	1%
2NxN	60%	29%	3%	0%	3%	3%	1%	1%
Nx2N	60%	3%	29%	0%	1%	1%	3%	3%
NxN	52%	3%	3%	38%	1%	1%	1%	1%
2NxnU	62%	6%	3%	0%	25%	2%	1%	1%
2NxnD	60%	7%	3%	0%	2%	26%	1%	1%
nLx2N	61%	3%	7%	0%	1%	1%	26%	2%
nRx2N	61%	3%	7%	0%	1%	1%	2%	25%

in general. The probability of a PU being encoded as 2Nx2N is above 50% in every row of the table, reaching its peak at 91% when the HQ PU is also 2Nx2N. Rectangular SMPs (2NxN and Nx2N) have a small chance of occurring when AMPs were encoded in the reference bitstream, following the same orientation (horizontal or vertical). In other words, if a PU was encoded as 2NxnU or 2NxnD, it has a chance of being encoded as 2NxN in the lower-quality representations. We can also see that the chances of an AMP being selected is very small unless the same happened in the HQ version. If we accumulate the probabilities of a PU being encoded as either 2Nx2N or as the same part from the HQ bitstream, we come up with a percentage higher than 85% for all cases. Therefore, it is also clear that many PUs can be skipped if we know the decision from the reference encoding.

Table 5.5 shows the occurrences of transforms of size  $\text{Tr}_{LQ}$  in each CU, given that the same CU in HQ bitstream used transforms of size  $\text{Tr}_{HQ}$ . As we know, more than one TU can be present in a single CU, so this size actually represents the minimum transform size, which can also be interpreted as the maximum RQT depth. The *Skip* value represents the cases in which transforms were not applied due to the lack of significant residues (generally when the residue block is made of zeroes).

Table 5.5: Minimum transform size probabilities given the minimum sizes in the HQ bitstream

$\text{Tr}_{HQ} \setminus \text{Tr}_{LQ}$	Skip	4x4	8x8	16x16	32x32
Skip	98.4%	0.3%	0.3%	0.3%	0.7%
4x4	45.0%	35.0%	9.5%	6.8%	3.7%
8x8	28.8%	7.2%	36.7%	11.3%	16.0%
16x16	40.7%	2.1%	4.3%	37.1%	15.8%
32x32	37.6%	0.2%	1.2%	2.3%	58.8%

The size of the transforms used in the LQ encodings are also highly correlated with

the ones from the HQ version. Specially when a transform is skipped in the HQ stream, the chances of it being skipped in the LQ versions is above 98%. We also observe a similar trend observed in the CU and PU analysis, where larger blocks and transform skips are more likely to occur as bitrate is reduced. This is expected, because the rate control algorithms use the QP to guide the bitrate towards its target, increasing this parameter when the target bitrate gets smaller. As we know, higher QPs reduce the significance of the residue samples, which ultimately favors the use of large transforms. Note however that there is a small chance of smaller transform blocks occur on LQ representations, which was not observed in the previous analyses. This means that if we limit the RQT depth based on the transform size of the HQ bitstream, there is a higher chance of making wrong decisions. However, this does not necessarily mean that compression efficiency will be penalized, as will be discussed in Chapter 6.

Table 5.6 shows the prediction mode probabilities, where each value corresponds to the probability of a CU being predicted as  $\text{Mode}_{LQ}$  given that it was predicted as  $\text{Mode}_{HQ}$  in the reference stream, where Inter represents the inter-prediction mode with motion vectors and reference indices, SKIP stands for either SKIP or Merge mode, in which the motion parameters are inherited from neighboring PUs.

Table 5.6: Prediction mode probabilities given the minimum sizes in the HQ bitstream

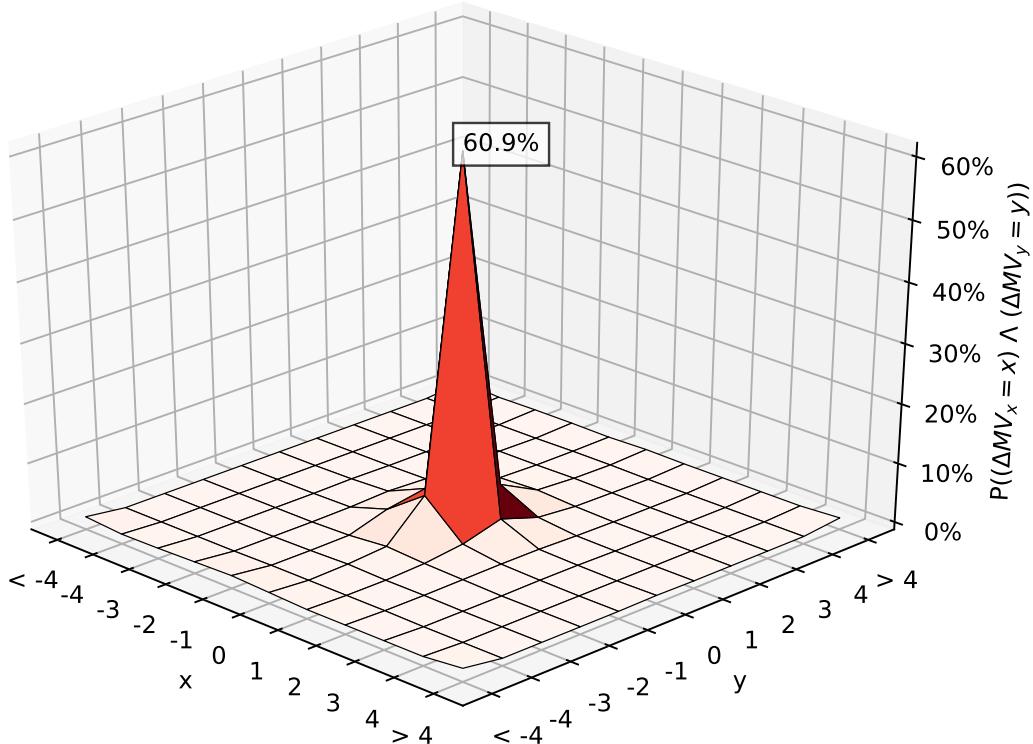
$\text{Mode}_{HQ}$	$\text{Mode}_{LQ}$	Inter	Intra	Skip
Inter		59.7%	1.3%	39.0%
Intra		25.4%	68.0%	6.6%
Skip		11.3%	0.1%	88.7%

When CUs are encoded with inter prediction in the HQ bitstream, there is high possibility that they will be encoded as SKIP mode in the lower representations. When the HQ mode is SKIP, this probability increases significantly, but the chances of inter-prediction are not negligible, so it might be interesting not to discard the inter-prediction. Finally, we can conclude that if the HQ mode is inter or SKIP, there is very low probability of it being intra-coded in the LQ encodings, so at least this mode can be safely disregarded.

The last analysis of this section studies how motion vectors from the HQ reference can be used to guide the Motion Estimation search on subsequent LQ representations. The first idea that comes to mind in this case is that the motion vectors are likely to be the same and can be directly mapped from the HQ to skip the ME entirely. To test this hypothesis, we extracted both horizontal and vertical coordinates of the motion vectors from each pair

and computed the deltas between them ( $\Delta MV(x)$  and  $\Delta MV(y)$ ). The chart in Figure 5.33 shows the occurrences for some values of delta. Note that a (0,0) delta means LQ Motion Estimation produced the same MV of the HQ encoding.

Figure 5.33: Average distribution of motion vector deltas between the HQ bitstream and the LQ representations.

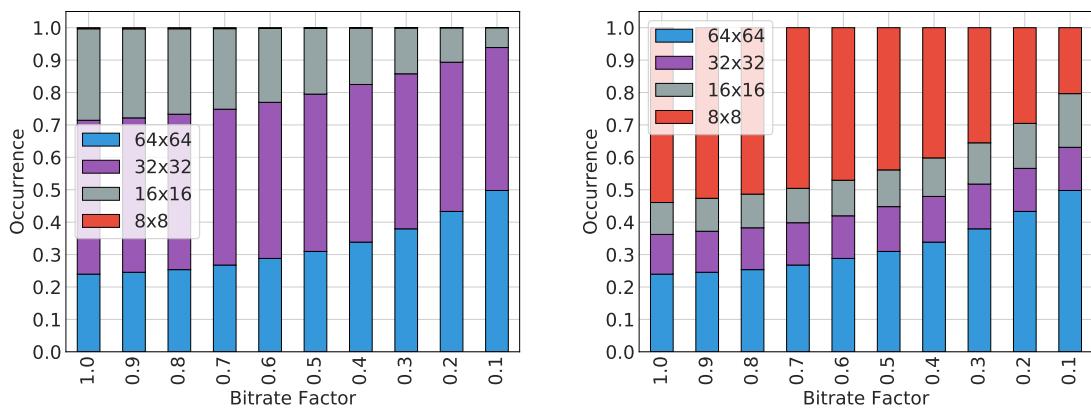


We can see with this chart that usually the motion vector produced in the ME of lower bit rates is very similar to the ones produced in the HQ encodings. This information can be used either to skip the ME process or to guide the search of the subsequent encodings, accelerating the convergence of the search algorithm. Note also that either the MV delta is very near the (0,0) region or very far (more than four pixels). The chart is pretty much flat on 0% for deltas equal to  $\pm 4$ ,  $\pm 3$ , and  $\pm 2$ .

### 5.2.2 CTU-grain Analysis

As stated in the previous section, limiting the CTU quadtree with upper and lower bounds with direct mapping or statistical-based rules is a promising solution to reduce transcoding complexity. However, a literary study on this topic showed that Machine Learning can also be used to implement efficient classifiers for this task. Most of them

work on a CU-grain decision, meaning that one classifier is built for each CU size (like it is done with the fast encoding decisions in this thesis). However, we decided that it might be interesting to try a different approach with CTU-level decisions. Our motivation came from the results presented in Figure 5.34, showing the minimum and maximum CU size of CTUs for different bitrate factors (a bitrate factor of 0.9 means that the target bitrate used was  $0.9 * HQ_{BR}$ , and a bitrate factor of 1 represents the reference encoding with  $HQ_{BR}$  as target).



As expected, reducing the target bitrate has a significant impact in the CTU partitioning. In the maximum CU size chart, it is possible to see that the 64x64 and 32x32 CUs are clearly used the most, the latter being the most common of the two. While the 64x64 occurrence starts at 24% in the reference bitstream, increasing to almost 50% at the lowest bitrate, the 32x32 occurrence has an almost constant behavior between 44% and 47%. If we combine the 32x32 and 16x16 results, the probability of making a correct guess with these values ranges from 50% (lowest bitrate) to almost 76% (highest bitrate). This means that in many cases it is possible to skip the 64x64 evaluation, specially at higher bitrates. The 8x8 CUs are very rarely the maximum size, occurring at most 0.34% in the HQ encodings. This is also expected, because a maximum size of 8x8 means that the entire CTU was encoded as 8x8 CUs, which is very unlikely given the advantages of encoding larger blocks.

In Figure 5.34(b), it is clear that the minimum CU size is predominantly 8x8 at higher bitrates, starting at almost 54% in the reference stream, but this value constantly diminishes for lower bitrates, reaching a minimum of 20% with a 0.1 bitrate factor. Therefore, the 8x8 CUs can be skipped 80% of the times without compromising the compress-

sion efficiency in the lowest bitrate. Surprisingly, the intermediate 32x32 and 16x16 are not usually the minimum CU size. In fact, if we combine the 32x32 and 16x16 occurrences, the result is always inferior to those of 64x64 CUs regardless of the bitrate factor. This is actually a positive conclusion, because it means that in many cases it is possible to completely skip the CTU quadtree evaluation, and this is accentuated at lower bitrates.

Therefore, if we are able to train two classifiers that can predict the upper (maximum size) and lower (minimum size) bounds of the CTU quadtree, we will be able to take advantage of two scenarios: when encoding for high bitrates, where more partitionings are expected, the upper bounds will assist in skipping large CUs; in the case of low target bitrates, in which larger blocks are more common, the lower bounds will be responsible for skipping the evaluation of smaller CUs.

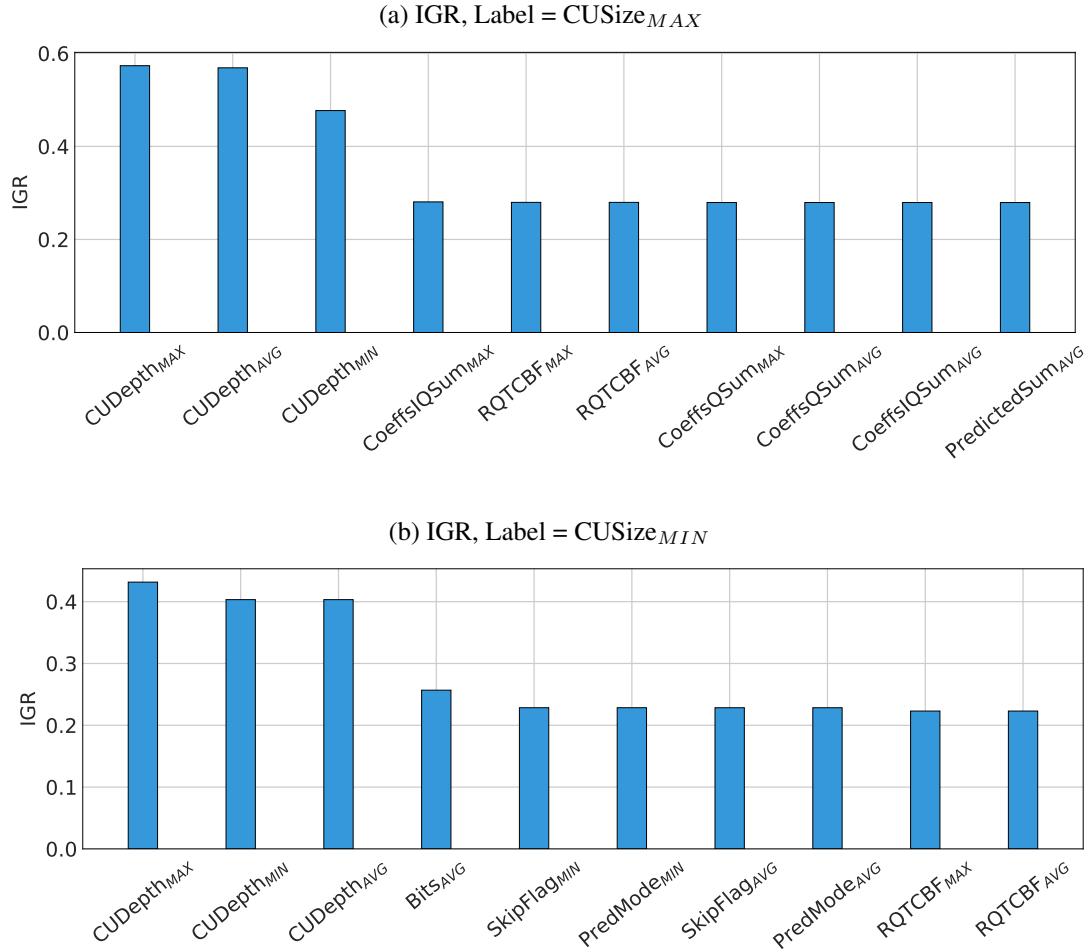
The features extracted for each CU in the previous section were reduced to a single value for the entire CTU, using the average, minimum, and maximum operations. Two different types of labels were considered in this analysis as well: the minimum and the maximum CU size. This will require the use of two data sets to train the classifiers, so the following analyses will be presented for each separate data set, one with the minimum and one with maximum CU size as label. The reasoning here is that these values can be used as the lower and upper bounds for the CTU quadtree evaluation, reducing the encoding time.

Note that, differently from the encoding models, each label has four possible values, which correspond to the supported CU sizes. However, the analysis of Figure 5.34(a) proved that there is no sense in considering 8x8 as a possible maximum CU size. Therefore, for this particular case, our labels were reduced to three: 64x64, 32x32, 16x16.

Figure 5.35 shows the IGR of the top ten features for the maximum and minimum CU size prediction problem. Since only Random Forests were used to train the transcoding classifiers in this work. The F-score analysis will be omitted.

The charts show that the features related with the CU depth of the HQ bitstream are the most relevant ones to predict the upper ( $CUSize_{MAX}$ ) and lower ( $CUSize_{MIN}$ ) bounds of the LQ encodings. In Figure 5.35(a), we can also observe that the metrics related with the sum of coefficients are also relevant. This is probably related to the fact the blocks with low coefficients are more likely to be encoded with larger CUs. Different features appear in Figure 5.35(b). For this case, the coefficients are less important and are replaced by information regarding the decisions made in the HQ encoding. The  $SkipFlag_{MIN}$  feature is equal to zero, when at least one CU in the CTU was encoded with the intra or inter

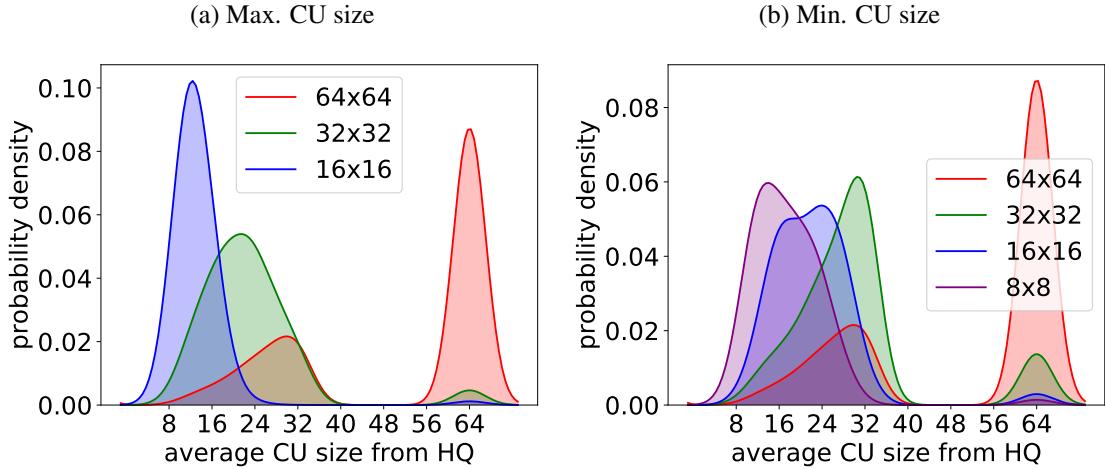
Figure 5.35: Information Gain Ratio of the CTU-grain transcoding features for prediction the maximum (a) and minimum (b) CU size in each CTU.



modes, or one, when all CUs are encoded as *Skip*. This feature was identified as relevant because, when all CUs are encoded as *Skip* in the HQ version, there is a very high chance of it being encoded as a single 64x64 block in lower bitrates.

Figure 5.36 shows the probability distribution of the classes in function of average CU size. In Figure 5.36(a) it is possible to observe that when the average CU size in HQ BS is near 64x64, there is a high possibility of finding Max. CU sizes of LQ BS with size 64x64 as well. When looking to the other classes, if the average is between 16x16 and 32x32, the frequency of CU sizes with 32x32 blocks is higher, and when the average CU size is between 8x8 and 16x16, the Max. CU size of LQ BS are in the biggest part of the cases, 8x8. So visually, a high predictive power of this feature is observed. On the other hand, in Figure 5.36(b) it is observed that when average CU size of HQ blocks are near 64x64, the Min. CU sizes of the LQ BS, will be predominantly 64x64. However, when looking for the other 3 classes, an overlap is observed, whereby the predictive power of

Figure 5.36: Probability density functions of all classes in function of the most important feature



that feature is not very explicit visually.

### 5.3 Final Remarks

The encoding analysis showed that the information extracted from this process can be of great assistance in predicting the outcome of the partitioning decisions. It was shown that, if we group our data by the related CU size, our information gain is optimized, achieving higher scores than other alternatives. This guided the decisions of the fast encoding methods discussed in the next Chapter. It was also concluded that the Motion Estimation analysis presented worse results in terms of information gain ratio, and this might be related to how we modeled the fast ME methods during this study, or even to the quality of the extracted features for these particular processes.

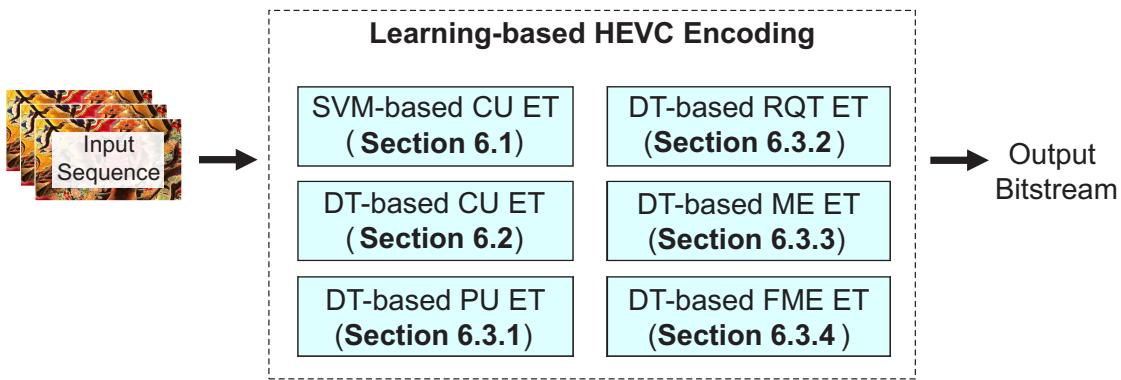
In the transcoding analysis, we observed that the partitioning decisions are highly correlated across encodings under different target bitrates. This supports the framework that will be presented in the next Chapter, in which a reference (HQ) bitstream is used to guide the decisions of dependent (LQ) encodings while eliminating those which probably will not be efficient. The ME analysis revealed that usually the MV from the HQ is very similar to the MV produced in the LQ encodings, showing that this vector can be used to either skip the entire IME or at least reduce its search range. As will be discussed in next Chapter, the latter approach was adopted, because there is a non-negligible chance of a better MV being in the vicinity.



## 6 LEARNING-BASED MODE DECISION FOR HEVC ENCODING

The analyses discussed in Chapter 5 was key to enable a good understanding of how the encoding and the bitstream information can be used to predict the decisions of HEVC encoders. From this analysis, and with the help of Machine Learning techniques, fast decision models were trained to reduce the encoding and transcoding (more specifically transrating) complexity using early termination (ET) methods. In addition, fast heuristics based on decoding information were also designed for transcoding complexity reduction. An overview of the contributions of this chapter and the sections where they are presented is shown in Figure 6.1.

Figure 6.1: Overview of the contributions presented in this thesis for encoding applications.



The following sections will elaborate each encoder-based approach developed in this thesis, starting with its first milestone, which consisted of a fast CU partitioning decision using SVMs. Then, we will show how we extended this analysis for other encoding decisions with the use of Decision Trees. The motives to migrate from SVMs to Decision Trees are also explained. Finally, the transcoding solutions will be presented in Chapter 7, showing that Machine Learning and statistical-based heuristics can be combined to provide excellent time savings with minor losses in compression.

Table 6.1 lists the sequences used to train our models, as well as those used to extract the compression and time savings results of the encoding fast decisions, where  $Tr$  and  $Te$  represent the sequences in the training and testing phases respectively.

The sequences were encoded following the common test conditions (BOSSEN, 2012), with Random Access reference structure and QP values of 22, 27, 32, 37 and 42. The first 150 frames of each sequence were used instead of the entire sequence to reduce the volume of information, but this frame count guarantees that at least two seconds of each sequence are accounted for. In addition, Intra frames were removed from our

Table 6.1: Video sequences used for training and testing in the encoding fast decision methods

<b>class</b>	<b>sequence</b>	<b>resolution</b>	<b>fps</b>	<b>frame count</b>	<b>assignment</b>
A	Traffic	2560x1600	30	150	Tr
	NebutaFestival	2560x1600	60	300	Tr
B	BasketballDrive	1920x1080	50	500	Tr
	ParkScene	1920x1080	24	240	Tr
C	BQMall	832x480	60	600	Tr
	PartyScene	832x480	50	500	Tr
D	BlowingBubbles	416x240	50	500	Tr
	RaceHorses	416x240	30	300	Tr
A	PeopleOnStreet	2560x1600	30	150	Te
	SteamLocomotiveTrain	2560x1600	60	300	Te
B	BQTerrace	1920x1080	50	500	Te
	Cactus	1920x1080	50	500	Te
	Kimono	1920x1080	24	240	Te
C	RaceHorsesC	832x480	30	300	Te
	BasketballDrill	832x480	50	500	Te
D	BasketballPass	416x240	50	500	Te
	BQSquare	416x240	60	600	Te
E	Johnny	1280x720	60	600	Te
	Kristen&Sara	1280x720	60	600	Te
F	ChinaSpeed	1024x768	30	500	Te
	SlideEditing	1280x720	30	300	Te

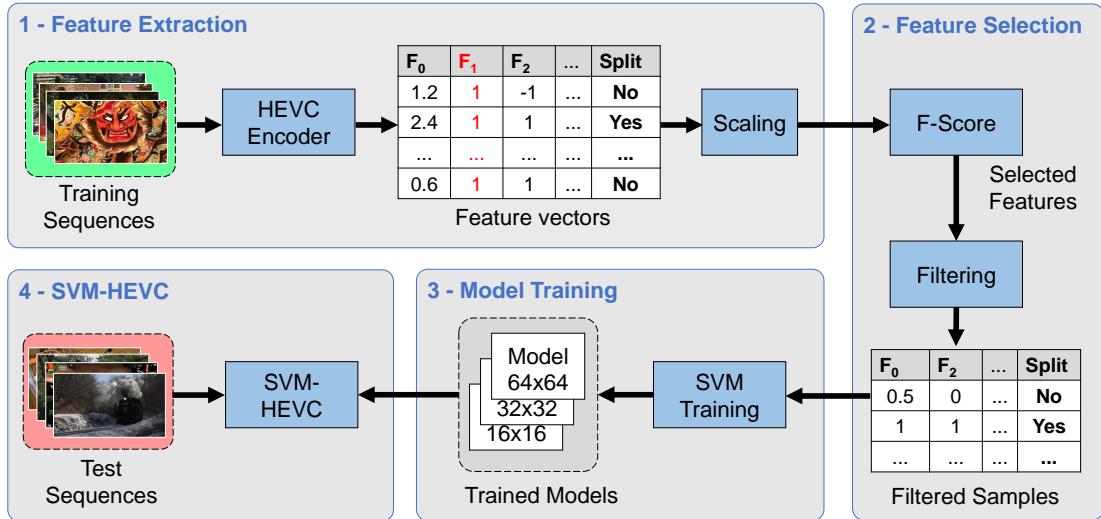
sampling because they do not contain motion information, which is part of our encoding feature set.

## 6.1 Fast HEVC CU partitioning Decision with Support Vector Machines

Figure 6.2 depicts the framework designed in this work to implement the SVM-based Mode Decision in HEVC encoders.

In Figure 6.2, coding features are computed during HEVC encoding along with information about the CU partitioning. The scaling step is required to train the SVM classifiers, as they do not perform well when features have different value ranges. Afterwards, features are analyzed based on their F-Score, and the less useful features are ignored in the next steps. The selected feature samples are used as input to the SVM training procedure.

Figure 6.2: Framework used in this work for training CU partitioning decision SVM classifiers.



In this work we used the LibSVM implementation to train our classifiers (CHANG; LIN, 2011). The final step consists in assessing the coding efficiency of the trained classifiers using a modified encoder, which incorporates the SVM functions to decide if further CU splitting is necessary.

As stated in Chapter 3, SVMs are difficult to train due to their multi-parameter optimization. To keep the computation requirements at manageable levels, we used at most 10 thousand random samples, decreasing the training time significantly. For testing, 50 thousand feature vectors were used. The training set was equally distributed, i.e., it contains the same number of examples for split and unsplit CUs, because SVMs are less efficient with unbalanced data.

### 6.1.1 SVM Algorithm Analysis

In this work, one classifier for each CU size ( $64 \times 64$ ,  $32 \times 32$  and  $16 \times 16$ ) was trained. The  $8 \times 8$  CUs do not require a partitioning decision classifier, as they are already the smallest size supported by HEVC. The setup used for SVM training is described in Table 6.2. The training and test sets are from Table 6.1. Aside from the standard QPs defined in the CTC document, an extra QP of 42 was used to increase the input variability. Note that each classifier was trained with samples from every resolution and QP value defined in this setup in order to increase the generalization of our method. The limits for testing the C and G parameters were assigned as an extended range, using as reference the default values used in the Grid Search implementation available in the LibSVM toolset

(CHANG; LIN, 2011).

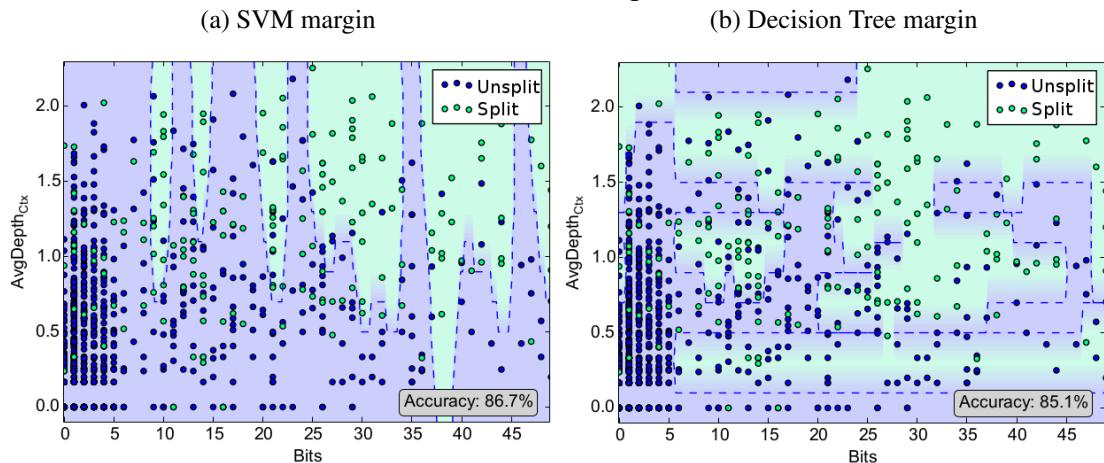
Table 6.2: Setup used for SVM training.

LibSVM version	3.22
Encoder (E)	HEVC Model 16.8
QPs	22, 27, 32, 27 and 42
Reference Structure	Random Access
Kernel	Radial Basis Function (RBF)
Slack Parameter (C)	From $2^{-8}$ to $2^8$
Gamma (G)	From $2^{-8}$ to $2^8$
Training size (K)	From 100 to 2500
Features (F)	From 1 to 28

We first compared SVM and Decision Tree classifiers in order to assess whether SVMs are better suited for solving the partitioning problem under analysis. Figure 6.3 depicts the decision margin for CU splitting decision using an SVM with the Gaussian (RBF) kernel (a) and a Decision Tree (b). For visual purposes, this is a simplified instance of our analysis, using only two features with high F-score and a reduced sample count (2000 samples). It is possible to observe that the SVM classifier provided a 1.6% better accuracy, and its decision surface is simpler than the one using a Decision Tree. In Machine Learning, simpler decision margins are preferred over complex ones, because it reduces the chance of overfitting.

The LibSVM implementation supports different kernel functions, but the Gaussian Radial Basis Function (RBF) kernel performs well on most cases. In this study, we experimented the linear kernel as well, but training was considerably slower, and the accuracy was very similar to that of RBF (usually within a 3% error margin).

Figure 6.3: Decision margin for the 64x64 CU partition decision using (a) SVM with an RBF kernel and (b) Decision Tree classifiers (simplified instance).



Examining the plots in Figure 6.3, along with results from previous analyses using SVMs, the following observations were made:

**Observation 1:** the SVM decision boundary is capable of grouping most instances correctly, but there are still many misclassifications. Increasing the number of features is a typical solution to improve accuracy, but it can also lead to overfitting when many irrelevant features are used.

**Observation 2:** experiments with SVM classifiers also showed that the number of support vectors is related to the number of input samples. Since all the support vectors are used to compute the decision function, the decision complexity can become a hindrance and lead to negligible time savings. In other words, it is pointless to use SVM decision to skip some CU computations if it takes more time than evaluating every CU partition. Therefore, our training procedure also aimed at minimizing the number of training samples.

**Observation 3:** in the framework of CU partitioning prediction, there are two types of misclassifications: (1) when a CU is not split and the classifier decides to split it further; and (2) when a CU must be split and the classifier decides otherwise. The former is a misclassification that incurs in more computations, because smaller CUs will be evaluated and the encoder will still choose to use a larger partition, but rate-distortion remains the same because the correct partition is still considered in the process. The second type, however, introduces rate-distortion penalties, because the correct partitioning will not be considered.

From the observations stated in the previous section, we designed a procedure to train SVMs considering not only the decision accuracy, but also the impact of classifiers on rate-distortion efficiency. This method is described by Algorithm 2. SVM training consists of optimizing the  $C$  parameter along with the kernel hyper-parameters. The RBF kernel has a single gamma ( $G$ ) parameter to optimize. Note that we repeated this procedure for each CU size:  $64 \times 64$ ,  $32 \times 32$  and  $16 \times 16$ .

The algorithm starts with an empty set ( $\vec{h}$ ) that will contain the temporary classifiers. Note that a scaling step is applied on the input samples (line 3) prior to their use in the training. Following the algorithm, on lines 4 to 13 we have the outer loop, which corresponds to a grid search over the number of samples ( $k$ ) and features ( $f$ ). The  $k$  examples are randomly sub-sampled from the original training set. On line 7, a filtering of samples is applied, in which only the top  $f$  features are kept using the F-Score metric as ranker. Then we perform a nested grid search (line 8), which optimizes the  $C$  and  $G$

---

**Algorithm 2** SVM training procedure to optimize the C, G, K and F variables
 

---

**Input:** **examples:** set with N training samples  
**E:** HEVC encoder with SVM-based CU decision  
 $\vec{K}, \vec{F}, \vec{C}, \vec{G}$ : vectors with the number of samples, features, slack parameters, and RBF gammas respectively

```

1:  $\vec{h} \leftarrow \emptyset$                                  $\triangleright$  vector of SVM classifiers
2:  $models \leftarrow \emptyset$ 
3:  $data_{scaled} \leftarrow \text{scale}(\text{examples}, \text{range} = [0,1])$ 
4: for each  $k \in K$  do
5:    $data_{crop} \leftarrow \text{randomSample}(data_{scaled}, k)$ 
6:   for each  $f \in F$  do
7:      $data_{filtered} \leftarrow \text{filter}(data_{crop}, f)$ 
8:      $\vec{h}[k, f] \leftarrow \text{gridSearch}(data_{filtered}, \vec{C}, \vec{G})$ 
9:     accuracy  $\leftarrow \text{SVMPredict}(\vec{h}[k, f], data_{scaled})$ 
10:    BDTSRatio  $\leftarrow \text{runEncoder}(E, \vec{h}[k, f])$ 
11:     $models[k, f] \leftarrow [\vec{h}[k, f], BDTS_{\text{Ratio}}]$ 
12:   end for
13: end for
14: bestModel  $\leftarrow \text{MAX}(models, \text{key} = BDTS_{\text{Ratio}})$ 

```

---

parameters on the filtered samples.

The best model selected after all these computations is the one with the best trade-off between rate-distortion and complexity, which is represented by the BDTS<sub>Ratio</sub> value, defined in Equation 2.3.

Since we need to maximize savings and minimize compression penalty, smaller BDTS<sub>Ratio</sub> values are better. To reduce training time, we computed this ratio using rate-distortion results from encoding a single sequence with low resolution (BasketballPass).

The necessity of employing BDTS<sub>Ratio</sub> in the training process was confirmed after several tests, because some classifiers with equivalent accuracy presented different rate-distortion results when used for HEVC mode decision, as explained in Observation 3.

The works that use SVMs with RBF kernels commonly perform a grid search to find the best  $C$  and  $G$  values. This is necessary because there is no direct relationship between  $C$ ,  $G$  and training accuracy, so it is very difficult to design heuristic methods to speedup this process. In our analysis, we observed that a second grid search to optimize the number of samples ( $k$ ) and features ( $f$ ) was also required in order to obtain good performance with a subset of the train data.

The final classifiers for each CU size (trained with the setup described in Table 6.2) and their respective values for  $C$ ,  $G$ ,  $k$  and  $f$  are listed in Table 6.3.

Table 6.3: Characteristics of the SVM classifiers obtained using Algorithm 2

CU size	<b>64x64</b>	<b>32x32</b>	<b>16x16</b>
Slack Param. (C)	$2^5$	$2^7$	$2^7$
RBF Gamma (G)	$2^3$	$2^{-3}$	$2^{-7}$
Number of samples (K)	900	700	1000
Number of features (F)	9	21	18
BDTS <sub>Ratio</sub> *	0.29	-0.086	-0.026
Train Accuracy	90%	81%	75.6%

\* Using the BasketballPass sequence only

Note that prediction is a harder problem on smaller CU blocks. For 32x32 and 16x16 classifiers, more features had to be used in order to achieve better results. Also note that the number of training samples was very similar on every CU classifier, indicating that using more features is more important than larger training sizes. The fact that no more than one thousand samples was required to build our classifiers is very significant, as it means that no more than one thousand support vectors will be used in the decision function.

To ascertain the efficiency and robustness of our classifiers, we compared the SVM-based HEVC encoder with the reference implementation under Variable Bitrate (VBR) and Constant Bitrate (CBR) conditions. These results are displayed in Tables 6.4 and 6.5. For the CBR tests, the initial QP was set to zero (default value in the HM configuration), and the average bitrates achieved with each QP value in the VBR tests were used as target bitrate. In addition, our VBR analysis was performed using Random Access (RA), Low Delay B (LB) and Low Delay P (LP) coding configurations. The Time savings (TS) between the reference HM encoder (Ref) and our proposed encoder (Test) were computed using the following equation:

$$TS = \frac{T_{Ref} - T_{Test}}{T_{Ref}} \quad (6.1)$$

As shown in Tables 6.4 and 6.5, the proposed encoder is capable of significantly reducing the encoder complexity with acceptable losses in BD-BR. The best results were achieved under VBR conditions, especially with the RA configuration. For the LB and LP configurations, a slight increase in BD-BR loss is noticed with less time savings compared with the RA results. However, time savings around 40% with BD-BR increases of 0.62% (LB) and 0.6% BD-BR (LP) are still good compromises between rate-distortion and complexity. Under CBR conditions, the highest BDTS<sub>Ratio</sub> increase was observed,

Table 6.4: BD-BR, Time Savings and BDTS<sub>Ratio</sub> relative to HM 16.8 using SVMs under Variable Bitrate and Constant Bitrate conditions (VBR QPs = 22, 27, 32, 37, CBR Targets = Bitrate<sub>QP22</sub>, Bitrate<sub>QP27</sub>, Bitrate<sub>QP32</sub>, Bitrate<sub>QP37</sub>)

<b>Sequence</b>	Random Access, <b>VBR</b>			Low Delay B, <b>VBR</b>		
	<b>BD-BR</b>	<b>TS</b>	<b>BDTS<sub>R</sub></b>	<b>BD-BR</b>	<b>TS</b>	<b>BDTS<sub>R</sub></b>
PeopleOnStreet	0.65%	25%	2.6	0.79%	22%	3.58
SteamLocomotive	0.17%	55%	0.31	0.18%	50%	0.36
BQTerrace	0.46%	52%	0.88	0.50%	47%	1.07
Cactus	0.62%	46%	1.35	1.23%	40%	3.08
Kimono	0.57%	44%	1.3	0.69%	40%	1.73
BasketballDrill	0.36%	39%	0.92	0.59%	36%	1.64
RaceHorsesC	0.45%	22%	2.05	0.5%	18%	2.74
BasketballPass	0.65%	46%	1.41	0.92%	43%	2.14
BQSquare	0.39%	47%	0.83	0.16%	35%	0.47
Johnny	0.25%	65%	0.39	0.66%	63%	1.06
Kristen& Sara	0.17%	64%	0.27	0.53%	62%	0.85
ChinaSpeed	0.99%	35%	2.83	0.77%	32%	2.44
SlideEditing	0.66%	70%	0.94	0.50%	69%	0.72
<b>Average</b>	<b>0.49%</b>	<b>47%</b>	<b>1.04</b>	<b>0.62%</b>	<b>43%</b>	<b>1.44</b>

with a complexity reduction of 46% at the cost of 1.12% in BD-BR. This drop in performance could be tied to the fact that only Random Access encodings under VBR conditions were used to generate our training set, which may bring unexpected circumstances during the test runs. However, the overall results are still promising considering that they were obtained without any additional model tuning, which ultimately let us conclude that the SVM-based CU partitioning decision is effective and robust. Also note that there is no apparent correlation between the spatial/temporal resolution of a sequence and the efficiency of our method, indicating that the performance will remain when varying resolutions need to be encoded.

### 6.1.2 Decision Threshold Effect on Rate-Distortion Performance

As stated in Section 6.1.1 (third observation), reducing incorrect decisions of not splitting a CU is important to maintain coding performance.

To accomplish this, one can modify the decision function of the SVM classifier to favor splitting CU decisions. By default, the decision threshold ( $Th_{Split}$ ) used by SVMs

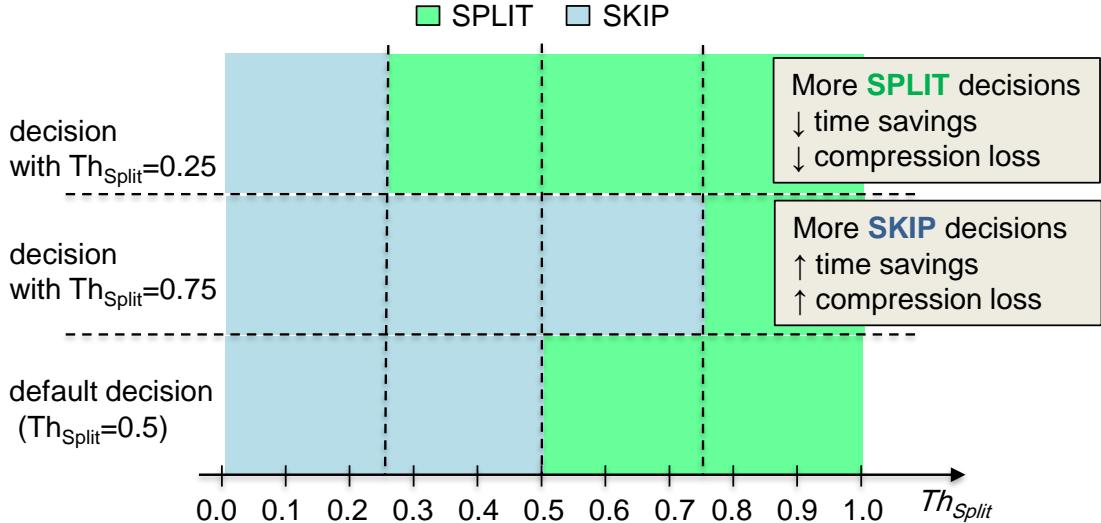
Table 6.5: BD-BR, Time Savings and BDTS<sub>Ratio</sub> relative to HM 16.8 using SVMs under Variable Bitrate and Constant Bitrate conditions (VBR QPs = 22, 27, 32, 37, CBR Targets = Bitrate<sub>QP22</sub>, Bitrate<sub>QP27</sub>, Bitrate<sub>QP32</sub>, Bitrate<sub>QP37</sub>)

<b>Sequence</b>	<b>Low Delay P, VBR</b>			<b>Random Access, CBR</b>		
	<b>BD-BR</b>	<b>TS</b>	<b>BDTS<sub>R</sub></b>	<b>BD-BR</b>	<b>TS</b>	<b>BDTS<sub>R</sub></b>
PeopleOnStreet	0.67%	21%	3.19	1.15%	26%	4.48
SteamLocomotive	0.73%	41%	1.8	0.02%	54%	0.05
BQTerrace	0.78%	42%	1.83	1.13%	52%	2.18
Cactus	0.98%	37%	2.66	1.02%	47%	2.18
Kimono	0.37%	36%	1.04	0.05%	42%	0.12
BasketballDrill	0.40%	32%	1.26	1.88%	39%	4.84
RaceHorsesC	0.52%	16%	3.21	2.15%	22%	9.61
BasketballPass	1.01%	40%	2.53	1.67%	42%	4.01
BQSquare	0.33%	29%	1.11	1.07%	41%	2.6
Johnny	0.24%	61%	0.4	1.09%	66%	1.66
Kristen& Sara	0.47%	61%	0.77	1.38%	64%	2.14
ChinaSpeed	0.80%	29%	2.78	1.81%	35%	5.24
SlideEditing	0.54%	70%	0.78	0.18%	65%	0.28
<b>Average</b>	<b>0.6%</b>	<b>40%</b>	<b>1.52</b>	<b>1.12%</b>	<b>46%</b>	<b>2.54</b>

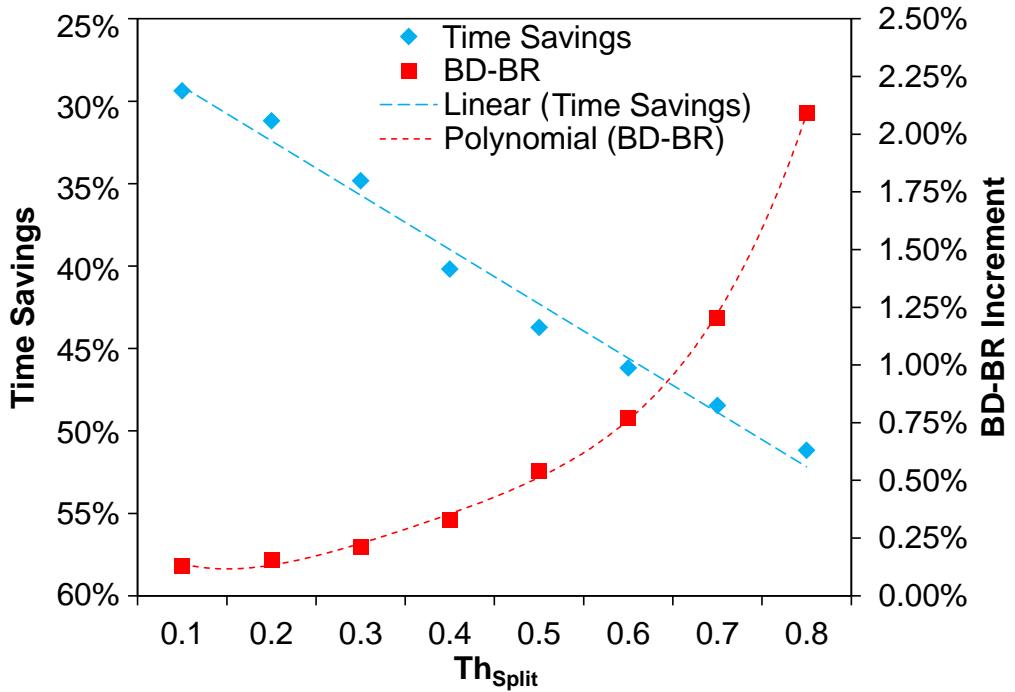
is 0.5 (since our labels are 0 and 1), and the confidence of a prediction is given by how close to 0 or 1 is the decision function output. In other words, if the decision function output is higher than 0.5, the classifier will label the current CU as split and vice-versa. Therefore, if we decrease this decision threshold to 0.25 for instance, more split decisions will be made, since the CUs that would be classified as skip with low confidence are still labeled as split. This will likely lower the complexity reduction, but it will also reduce rate-distortion losses. On the other hand, increasing  $Th_{Split}$  to 0.75 leads to more CUs that are skipped. In this case, rate-distortion is sacrificed for the sake of higher computation savings. This is depicted in Figure 6.4.

In Figure 6.4, when  $Th_{Split} = 0.25$ , more CUs will be classified as split. This will likely reduce the complexity reduction, but it will also reduce rate-distortion losses. On the other hand, increasing  $Th_{Split}$  to 0.75 leads to more CUs that are skipped. In this case, rate-distortion is sacrificed for the sake of higher computation savings. With this approach, it is possible to tune the decision threshold for applications that need to preserve quality or to increase complexity reduction. Figure 6.5 depicts different BD-BR and time savings results for each tested value of  $Th_{Split}$ .

With this approach, it is possible to adjust the decision threshold to favour encod-

Figure 6.4: Split decision for different values of  $Th_{Split}$ .

ing quality or to reduce encoding complexity. Figure 6.5 depicts different BD-BR and time savings results for each tested value of  $Th_{Split}$ .

Figure 6.5: Average Time Savings, BD-BR increment, and their linear and polynomial fits, for different values of  $Th_{Split}$ .

The plot in Figure 6.5 shows that the encoding time savings is very sensitive to the decision threshold. When this threshold is closer to zero, the savings are also small, but the BD-BR increase is also negligible. For instance, when  $Th_{Split} = 0.1$ , time is reduced by 29.3% and BD-BR is incremented merely by 0.13%. On the other hand, as the threshold gets closer to one, the time savings decrease linearly, and the BD-BR

increases exponentially. When  $Th_{Split} = 0.8$ , time savings go up to 52.2%, but the BD-BR increment also scales up to 2.1%. Therefore, in order to keep a high encoding quality, a small decision threshold should be used. Still, even when  $Th_{Split}$  is set to 0.6, the BD-BR increment is smaller than 1% (precisely 0.77%), which is negligible, and the time savings are around 46%. It is also possible to tune this parameter to achieve different complexity points. For instance, if 30% encoding time savings are sufficient, the decision threshold should be around 0.2. However, if a time reduction of 50% is required, the threshold should be in the 0.5 to 0.7 range.

### 6.1.3 Results and Discussions

Tables 6.6, 6.7 and 6.8 show the rate-distortion and complexity results achieved on each test sequence using the SVM-based partitioning decision with different values of decision threshold, namely 0.1, 0.3 and 0.7 (for  $Th_{Split} = 0.5$ , see Table 6.4). The Time Savings were computed using the formula in (6.1).

These results confirm the effect of the decision threshold on both time savings and rate-distortion performance loss. Higher values of  $Th_{Split}$  increase time savings and rate-distortion loss proportionally, though it is clear that the latter grows at a much faster pace. When  $Th_{Split}$  scales from 0.1 to 0.7, the average time savings increase by a ratio of 1.53, whereas BD-BR increment goes up by a 8.615 factor, reinforcing that smaller values of  $Th_{Split}$  should be prioritized when possible.

In addition, the complexity reduction of our approach seems to be affected by scene characteristics. For a  $Th_{Split}$  of 0.1, the savings range from 5% (RaceHorses) to 67% (SlideEditing). The content of these sequences is very different: the first one is characterized by intense motion and complex texture, whereas the latter is mostly still, with simple texture. This type of variation was expected because some sequences require much more partitioning than others.

The second analysis compares the partitioning obtained using the reference HEVC encoder and the SVM-based encoder. For this comparison, the lists of CU partitions were extracted from the reference ( $L_{ref}$ ) and test ( $L_T$ ) encoders, and the Partition Similarity

Table 6.6: Rate-Distortion-Complexity of the test sequences when  $Th_{Split} = 0.1$  (Random Access, QP=22, 27, 32, 37)

<b>Class</b>	<b>Sequence</b>	<b>BD-BR</b>	<b>TS</b>	<b>BDTS<sub>Ratio</sub></b>
A	SteamLocomotive	0.14%	43%	0.32
A	PeopleOnStreet	0.11%	11%	1
B	Kimono	0.17%	23%	0.75
B	Cactus	0.16%	33%	0.48
B	BQTerrace	0.26%	43%	0.6
C	BasketballDrill	0.03%	18%	0.17
C	RaceHorsesC	0.03%	5%	0.64
D	BasketballPass	0.16%	28%	0.58
D	BQSquare	0.13%	33%	0.41
E	Johnny	0.25%	57%	0.44
E	Kristen&Sara	-0.01%	56%	-0.02
F	ChinaSpeed	0.20%	19%	1.07
F	SlideEditing	0.05%	67%	0.07
<b>Average</b>		<b>0.13%</b>	<b>33.5%</b>	<b>0.39</b>

( $P_{Sim}$ ) metric is defined as:

$$P_{Sim}(L_{ref}, L_T) = \frac{\sum_{CU \in L_{ref}} Hit(CU, L_T) * CU_w CU_h}{\sum_{CU \in L_{ref}} CU_w CU_h} \quad (6.2)$$

In (6.2), the  $Hit(CU, L_T)$  function returns 1 when the partition is found in the test list and 0 otherwise, and  $CU_w$  and  $CU_h$  are used to compute the area that corresponds to this partition. Thus, the numerator is the area within the frame that was partitioned equally with both encoders, and the denominator represents the total frame area. The  $P_{Sim}$  metric corresponds to the percentage of CUs that are equally partitioned on both encoders, where each partition count is normalized by the corresponding CU area. Figure 6.6 shows the partitioning results of a Full HD video frame using  $Th_{Split} = 0.5$  and a QP of 32. Partitions marked in red represent the CUs that were split further in the reference encoder. They represent higher rate-distortion penalties for our SVM-based encoder. The green-marked partitions are the ones that were split with the SVM-based encoder, but not in the reference. These partitions reduce the time savings, but without rate-distortion penalty.

The partitioning of both encoders is similar, with a  $P_{Sim}$  of 93.7%. The main differences occur in regions with intense motion, and many cases correspond to further

Table 6.7: Rate-Distortion-Complexity of the test sequences when  $Th_{Split} = 0.3$  (Random Access, QP=22, 27, 32, 37)

<b>Class</b>	<b>Sequence</b>	<b>BD-BR</b>	<b>TS</b>	<b>BDTS<sub>Ratio</sub></b>
A	SteamLocomotive	0.02%	47%	0.04
A	PeopleOnStreet	0.16%	15%	1.03
B	Kimono	0.34%	31%	1.08
B	Cactus	0.25%	38%	0.65
B	BQTerrace	0.34%	47%	0.71
C	BasketballDrill	0.05%	27%	0.17
C	RaceHorsesC	0.06%	10%	0.56
D	BasketballPass	0.20%	37%	0.54
D	BQSquare	0.21%	37%	0.57
E	Johnny	-0.04%	62%	-0.06
E	Kristen&Sara	0.09%	61%	0.15
F	ChinaSpeed	0.40%	25%	1.58
F	SlideEditing	0.34%	69%	0.49
<b>Average</b>		<b>0.19%</b>	<b>38.9%</b>	<b>0.49</b>

partitioning (split) misclassifications, which are not harmful to coding efficiency (as explained in Section 6.1.2).

The rate-distortion plots for the ChinaSpeed test sequence are shown in Figure 6.7. The curves confirm that the coding penalties of using SVMs for fast mode decision are negligible. ChinaSpeed was specially selected for this analysis, as it is the one with highest average BDTS<sub>Ratio</sub> among all tested sequences, i.e., it is one of the worst cases performance-wise.

### Comparison with State of the Art

Table 6.9 compares our results with the most recent HEVC encoding complexity reduction methods found in the literature using all the temporal structures.

The results show that our solution is capable of achieving the lowest BDTS<sub>Ratio</sub> among all references, but with different time savings. To provide a fair comparison, we also compiled results that came as close as possible to the time savings of each reference by adjusting the SVM decision threshold. The sequences used to evaluate our method are the ones marked "Te" in Table 6.1. With the RA configuration, we are able to outperform all of the related works in both time savings and BD-BR using a decision threshold of 0.3

Table 6.8: Rate-Distortion-Complexity of the test sequences when  $Th_{Split} = 0.7$  (Random Access, QP=22, 27, 32, 37)

<b>Class</b>	<b>Sequence</b>	<b>BD-BR</b>	<b>TS</b>	<b>BDTS<sub>Ratio</sub></b>
A	SteamLocomotive	0.39%	60%	0.65
A	PeopleOnStreet	1.90%	31%	6.12
B	Kimono	1.00%	52%	1.93
B	Cactus	1.49%	52%	2.87
B	BQTerrace	0.68%	55%	1.24
C	BasketballDrill	1.32%	45%	2.94
C	RaceHorsesC	1.22%	29%	4.19
D	BasketballPass	1.65%	47%	3.50
D	BQSquare	0.61%	50%	1.21
E	Johnny	0.61%	66%	0.92
E	Kristen&Sara	0.63%	66%	0.96
F	ChinaSpeed	2.27%	42%	5.39
F	SlideEditing	0.74%	70%	1.05
<b>Average</b>		<b>1.12%</b>	<b>51.2%</b>	<b>2.19</b>

and outperform most of them with a 0.5 threshold. In the LB case, although we achieve a better BDTS<sub>Ratio</sub> compared to the other works, (ZHANG et al., 2015b) is able to save 2.2% more encoding time, but this difference is not very significant and could be easily dealt with finer decision threshold adjustments. Also note that (ZHANG et al., 2015b) used QP values different from the ones defined in the CTC (24, 28, 32, and 36), so this might also affect the analysis. The LP results are the hardest to outperform in terms of time savings. We were not able to outperform (MOMCILOVIC et al., 2015), but since only sequences from classes A, B and F were encoded in that work, and the authors did not include the processing time of the online training routines nor the time of the decision function in their analysis, the comparison is not fair towards our work. Despite that, our method was still able to overcome their results when the same sequences are considered. In this case, our SVM-based encoder achieves time savings equal to 53.2% at the cost of 1.48% BD-BR, which results in a 2.77 BDTS<sub>Ratio</sub> against the 2.79 achieved by their results. Finally, it was also not possible to outperform the time savings of (ZHU et al., 2017), even though our BDTS<sub>Ratio</sub> is superior for all tested decision thresholds.

These references are very recent and can be regarded as state-of-the-art solutions due to their efficiency. Therefore, this is an indicator that the computational complexity reduction method proposed is an improvement over previous works in terms of combined

Figure 6.6: Partition Similarity of the BasketballDrive sequence for  $Th_{Split} = 0.5$ .



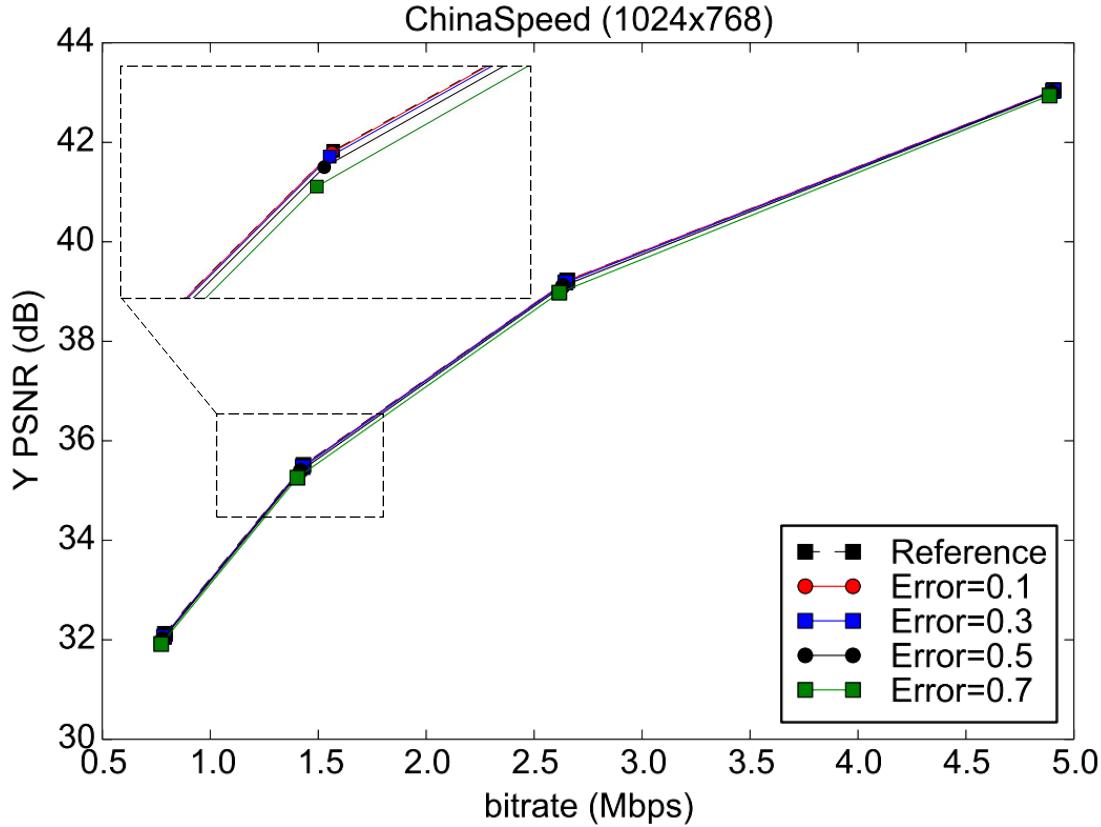
rate-distortion and complexity. Our improvements come from the use of more features to train the classifiers (such as RQT depth and motion vector prediction index), the inherent advantages of SVM over Decision Trees, and also through the use of a fine-tuned methodology that includes rate-distortion performance in the training step.

## 6.2 Fast HEVC CU partitioning Decision with Decision Trees

The SVM results showed that it is possible to overcome other references by: (i) using an appropriate feature selection coupled with a (ii) fine-tuned parameter optimization. However, it was still not clear which step (i or ii) was the main responsible to achieve these gains in performance. Therefore, the next step in this work consisted in training Decision Trees with the same set of features used in the SVM training to implement a CU Early Termination (CU ET) method.

The decision tree models were trained with data collected from the HEVC Model (HM) encoder (version 16.8) for the sequences defined in Table 6.1. The sequences were encoded with QP values of 22, 27, 32, 37 and 42, using the Random Access (RA) configuration, and the number of encoded frames per sequence was limited to 150. The features described in Chapter 5 were collected for each CU, forming feature vectors that were labeled as split, when the cumulative RD cost of the four sub-CUs was smaller than the current one, or unsplit otherwise. After all data were collected, three training sets were built, one for each CU size (64x64, 32x32, and 16x16). Then, each set was balanced to

Figure 6.7: Rate-Distortion plots for the ChinaSpeed sequence, using different values of  $Th_{Split}$ .



ensure the same distribution of split and unsplit cases using random under-sampling.

Table 6.10 shows the training results using the C5.0 algorithm (QUINLAN, 2015), including accuracy, true positive rate (TPR), and max tree depth of each classifier, using 200,000 training vectors. To ascertain the efficiency of the proposed set of features, Table 6.10 compares the training accuracy of the classifiers trained with both feature sets. Note that using more features increases the depth of the trees, but it represents a very small overhead when compared with the enormous amount of computations that are saved every time an early termination happens, as section 6.2.3 shows.

Finally, a feature usage statistics is presented in Table 6.11, where each value represents the percentage of decisions that required a test with the corresponding feature. Note that the features with the highest GR are tested on 100% of the decisions, as they were selected as root nodes of the decision trees.

Table 6.9: Comparison with related work

Cfg.	Ref.	Technique	BD-BR	TS	$BDTS_R$
RA	(CORRÊA et al., 2015)	Dec. Trees	0.28%	36.7%	0.77
	(KIM; PARK, 2016)	Bayesian Dec.	0.71%	34.9%	2.03
	(SHEN; YU, 2013b)	SVM	1.35%	44.7%	3.02
	This Work	<b>SVM, <math>Th_{Sp}=0.3</math></b>	<b>0.19%</b>	<b>38.9%</b>	<b>0.49</b>
LB		<b>SVM, <math>Th_{Sp}=0.5</math></b>	<b>0.49%</b>	<b>47%</b>	<b>1.04</b>
	(KIM; PARK, 2016)	Bayesian Dec.	0.63%	32.6%	1.93
	(SHEN; YU, 2013b)	SVM	1.66%	41.9%	3.02
	(ZHANG et al., 2015b)	SVM	1.98%	51.4%	3.84
	This Work	<b>SVM, <math>Th_{Sp}=0.5</math></b>	<b>0.62%</b>	<b>42.8%</b>	<b>1.44</b>
LP		<b>SVM, <math>Th_{Sp}=0.7</math></b>	<b>1.7%</b>	<b>49.2%</b>	<b>3.45</b>
	(KIM; PARK, 2016)	Bayesian Dec.	0.62%	32.7%	1.89
	(MOMCILOVIC et al., 2015)*	Neural Net.	1.43%	51.3%	2.79
	(ZHU et al., 2017)	SVM	2.66%	59.9%	4.44
	This Work	<b>SVM, <math>Th_{Sp}=0.5</math></b>	<b>0.6%</b>	<b>39.7%</b>	<b>1.52</b>
		<b>SVM, <math>Th_{Sp}=0.7</math></b>	<b>1.69%</b>	<b>48.3%</b>	<b>3.49</b>

\* Only sequences from classes A, B and F

Table 6.10: Train accuracy, true positive rate, and tree depth of the trained classifiers (200,000 vectors per data set)

Feature set	depth	Accuracy	TPR	Tree Depth
previous (CORRÊA et al., 2015)	d0	90.9%	89.5%	13
	d1	91.2%	89.1%	13
	d2	91.2%	88.8%	19
proposed	d0	92.7%	91.9%	20
	d1	93.6%	92.2%	24
	d2	94.2%	92.3%	21

### 6.2.1 Low-Complexity HEVC Encoder

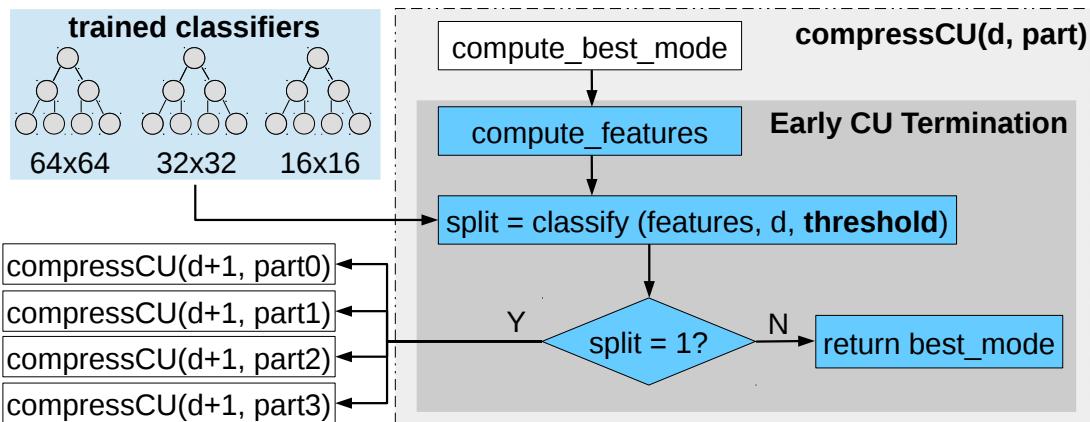
The trained classifiers are used as input to a modified HEVC encoder that implements an CU Early Termination scheme, depicted in Figure 6.8. Two methods were implemented in the HM software: (1) the feature extraction and (2) the decision function using the classifiers and the extracted features as input.

In Figure 6.8, the best mode for the current CU is computed and then the feature extraction routine is called to generate the input vector for classification. The decision function is called in the next step, using the classifier trained for the corresponding CU size, and its output is tested. If the classifier outcome is *split*, the CU partitioning eval-

Table 6.11: Average usage of the top ten features considering both data sets

Rank	Previous		Proposed	
	Feat. ID	Usage (%)	Feat. ID	Usage (%)
1st	SkipFlag	100	Bits	100
2nd	CtxDepth <sub>CTU</sub>	86	SplitFlag <sub>Left</sub>	57
3rd	MergeFlag	39	NormDiff <sub>Best,2N×2N</sub>	54
4th	NormDiff <sub>2N×2N</sub>	33	SkipFlag	38
5th	CostRatio <sub>2N×2N,MSM</sub>	25	SplitFlag <sub>Upper</sub>	37
6th	Cost <sub>MSM</sub>	22	SplitFlag <sub>UpperRight</sub>	36
7th	PU <sub>Part</sub>	19	MergeFlag	30
8th	Cost <sub>N×2N</sub>	11	Distortion	26
9th	Cost <sub>2N×N</sub>	6	RQT <sub>Depth</sub>	24
10th	Cost <sub>2N×2N</sub>	6	CtxDepth <sub>CTU</sub>	21

Figure 6.8: CU Early Termination scheme.



ation is continued. Otherwise, the splitting evaluation is terminated prematurely and the best mode found for the current CU is chosen.

### 6.2.2 Decision Tree-based Complexity-Scalable HEVC Encoder

The complexity reduction strategy presented in the previous paragraphs was employed in a complexity-scalable encoder implementation. This was achieved by including a split threshold ( $Th_{Split}$ ) parameter in the decision function, which leads to more or less split outcomes. This is an approach similar to that presented in Chapter 6.1.2. First, the original decision is computed ( $split$ ) along with its confidence factor ( $C$ ). The confidence of a decision is always computed along with it in the C5.0 algorithm. Then,  $split'$

is computed as:

$$split' = \begin{cases} 1 & \text{if } Th_{Split} < 0.5 \text{ and } C < (1 - Th_{Split}) \\ 0 & \text{if } Th_{Split} > 0.5 \text{ and } C < Th_{Split} \\ split & \text{otherwise} \end{cases}$$

When  $Th_{Split}$  is set to 0.5, the final output is exactly the same as the original classifier decision. When it is set below 0.5, any decision with a confidence below  $1 - Th_{Split}$  is reset to split, which increases the compression efficiency while decreasing the time savings. The opposite behavior occurs when the threshold is set above 0.5, favoring time savings over compression efficiency. A split threshold analysis and other comparisons are presented in the following section.

### 6.2.3 Results and Discussion

To assess the RD performance of the proposed method, 13 sequences with varying resolutions were used, following the Common Test Conditions defined by the JCT-VC group (BOSSEN, 2012). To reduce the risk of bias, the set of sequences in this analysis is complementary to the one used in the training phase. The RD and complexity results of the test sequences are presented in Table 6.12. For these experiments, the default threshold was used in the decision function (i.e.,  $Th_{Split} = 0.5$ ). Time savings (TS) were computed as per Equation (6.1), where  $T_{ref}$  is the encoding time of the original reference software (HM 16.8), and  $T_{test}$  is the encoding time with the proposed method, including the decision function calls. On average, the results in Table 6.12 show that the method is capable of reducing the HEVC encoding time in 47.8%, at the cost of a negligible compression efficiency loss of 0.24%. The rightmost column shows the ratio between RD efficiency and TS (BDTS<sub>Ratio</sub>), which is defined in Equation 2.3. On average, for every 1% in TS, a BD-rate increase of 0.0051% is required.

A comparison with related works is presented in Table 6.13. Note that *Previous* represents the work of (CORRÊA et al., 2015), which was re-implemented in HM 16.8 for comparison. The results prove that the new strategy outperforms (CORRÊA et al., 2015) for all configurations. The performance difference is smaller for the RA configuration because the authors in (CORRÊA et al., 2015) used RA data to train their models, but when other configurations (LP and LB) are considered, the difference is more significant.

Table 6.12: Rate-distortion and complexity results of the proposed method for all tested sequences using  $Th_{Split} = 0.5$ . (QP = 22, 27, 32, 37)

Class	Sequence	Random Access			Low Delay B		
		BD-BR	TS	BDTS <sub>R</sub>	BD-BR	TS	BDTS <sub>R</sub>
A	PeopleOnStreet	0.28%	23.0%	1.21	0.19%	19.4%	1.00
A	SteamLocomotive	0.32%	53.9%	0.59	0.20%	46.6%	0.43
B	Kimono	0.42%	43.7%	0.96	0.39%	35.1%	1.11
B	BQTerrace	0.47%	53.1%	0.88	0.37%	48.7%	0.75
B	Cactus	0.42%	47.8%	0.88	0.40%	41.5%	0.96
C	RaceHorsesC	0.14%	20.8%	0.68	0.14%	17.3%	0.82
C	BasketballDrill	0.17%	40.7%	0.41	0.09%	37.2%	0.24
D	BasketballPass	0.16%	43.1%	0.37	0.46%	42.0%	1.08
D	BQSquare	0.22%	45.5%	0.48	-0.02%	31.7%	-0.05
E	KristenAndSara	0.12%	69.1%	0.18	-0.03%	64.4%	-0.04
E	Johnny	0.19%	69.1%	0.27	-0.09%	64.8%	-0.13
F	SlideEditing	0.01%	73.3%	0.01	0.20%	73.0%	0.28
F	ChinaSpeed	0.44%	37.8%	1.17	0.17%	34.3%	0.50
Average		0.26%	47.8%	0.54	0.19%	42.8%	0.45

These results show that the proposed classifiers generalize better than (CORRÊA et al., 2015), since the training sets are also composed of RA data only. When compared to the remaining related works, the results outperform them in BD-BR, TS and BDTS<sub>Ratio</sub>.

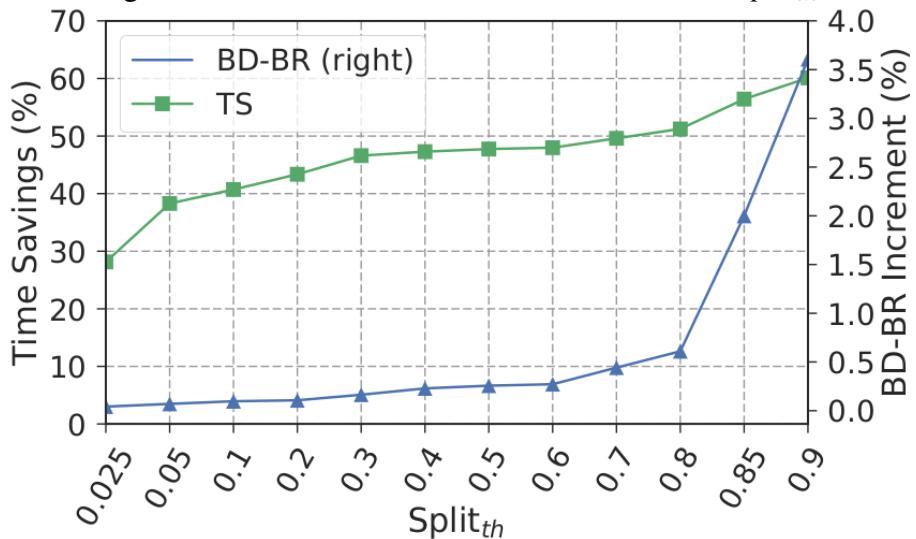
We can also conclude that the Decision Trees classifiers performed better in terms of rate-distortion and complexity compared with our previous SVM approach: with the default decision threshold, the first achieved a 0.54 BDTS<sub>Ratio</sub> against the 1.04 obtained with the latter. The decision threshold analysis also presented a better curve, meaning that Decision Trees should be used due to their simpler training procedure. That does not mean that Decision Trees are definitively better for the CU partitioning decision, because there might better ways of training SVM classifiers that might improve the final results, but at the cost of numerous, time-consuming training refinements. Therefore, the solutions for other partitioning decisions that will be presented in the next sections will focus on Decision Trees for the sake of simplicity.

The final analysis presents the effect of the  $Split_{th}$  parameter on rate-distortion-complexity efficiency. Several values for this parameter were tested, using the same sequences listed in Table 6.12. The results are summarized in Figure 6.9. The chart shows that the  $Split_{th}$  has significant impacts on both time savings and BD-BR. TS increases in a quasi-linear trend, whereas the BD-BR barely changes with a  $Split_{th}$  up to 0.8, but grows rapidly with larger values. The smallest time savings achieved are 28% with negligible 0.04% increase in BD-BR, whereas the highest time savings of 60% are achieved at the

Table 6.13: Comparison with the related work

Cfg.	Reference	BD-BR (%)	TS (%)	BDTS ( $\times 100$ )
RA	(KIM; PARK, 2016)	0.71	34.9	2.03
	(SHEN; YU, 2013b)	1.35	44.7	3.02
	Previous (CORRÊA et al., 2015)	0.32	47.2	0.67
	<b>Proposed DT classifier</b>	<b>0.24</b>	<b>47.8</b>	<b>0.51</b>
LB	(KIM; PARK, 2016)	0.63	32.6	1.93
	(SHEN; YU, 2013b)	1.66	41.9	3.02
	Previous (CORRÊA et al., 2015)	0.46	40.8	1.12
	<b>Proposed DT classifier</b>	<b>0.19</b>	<b>42.8</b>	<b>0.45</b>
LP	(KIM; PARK, 2016)	0.62	32.7	1.89
	(ZHU et al., 2017)	2.66	59.9	4.44
	Previous (CORRÊA et al., 2015)	0.39	38.1	1.03
	<b>Proposed DT classifier</b>	<b>0.22</b>	<b>41.2</b>	<b>0.53</b>

cost of 3.6% in BD-BR. However, a smaller BD-BR increase of 0.61% is observed for an average savings of 51%, which is a much better trade-off. These results ultimately let us conclude that several complexity points can be achieved with minimum RD loss using the  $Split_{th}$  threshold, which is useful for applications that require complexity scaling whereas still maintaining the encoding efficiency.

Figure 6.9: TS and BD-BR for different values of  $Split_{th}$ 

### 6.3 Learning-based HEVC Mode Decision with Decision Trees

After confirming that Decision Trees were more efficient than SVMs for the CU partitioning decision, we decided to apply the same methodology to generate classifiers of other decisions that also take part in the encoding complexity. Therefore, more Decision

Trees were trained to predict some outputs of the PU, RQT, IME and FME processes. The following paragraphs will explain how each process was optimized with the help of the trained classifiers.

### 6.3.1 PU Early Termination

From the analysis of Chapter 5, we could observe that the Prediction Unit decision spends a significant amount of time deciding on rectangular SMPs ( $2N \times N$  and  $N \times 2N$ ) and AMPs. Therefore, we designed Decision Trees to predict if the PU processing should stop at the  $2Nx2N$  evaluation or if it should continue with the other PUs.

The feature data was extracted right after the  $2Nx2N$  PU evaluation (see the Extraction Point 1 of Figure 5.1). The labels were assigned to 0, when the best PU was  $2Nx2N$ , and 1 otherwise. After the trees were trained with the C5.0 algorithm, using the same set of features from the CU ET classifiers, they were incorporated in the HM software. Algorithm 3 shows the pseudo-code of the proposed PU Early Termination (PU ET) method coupled with the CU ET algorithm discussed in the previous section. The sections formatted in red correspond to the logic that was inserted in the original implementation.

Note that the PU decision affects the features used in the CU early termination method (computed in line 17). If the PU classifiers decide not to evaluate the other PU parts (in line 4), features that rely on these evaluations (e.g.,  $PU_{part}$ ) will not have the same values that they would otherwise, because lines 6 to 11 will be skipped. Therefore, using both methods to speed up encoding time can be more harmful to compression efficiency. This could be circumvented with separate CU early termination classifiers trained with features from Extraction Point 1. This strategy is considered as a future investigation of this work. The rate-distortion-complexity results of the PU fast decision, as well as both strategies combined are presented in Table 6.14.

The results achieved with the PU ET method shows that 44.7% time savings can be achieved at the cost of 0.42% BD-BR penalty. Although this is a good compromise between rate-distortion and complexity reduction, the results are not on par with the ones achieved using the CU ET method. Therefore, it is possible to conclude that, for average time savings of 40% and below, it is preferable to disable the PU fast decision and stick with the CU early termination only. However, for higher savings, the PU fast decision should be used, increasing the average time savings to more than 61% at the cost of 1.94% in BD-BR. Note that the BD-BR penalty is of both methods combined is more

---

**Algorithm 3** CTU compression algorithm with the CU and PU Early Termination logic
 

---

**Input:** CU: current CU block

    d: current CU depth

```

1: costMSM ← skipMergePrediction(CU, d)
2: cost2N×2N ← interPredSearch(CU, d, 2N × 2N)
3:  $\vec{f} \leftarrow \text{computeFeatures}(CU, d)$                                 ▷ Extraction Point 1
4: evalPU ← decidePU( $\vec{f}$ , d)                                         ▷ Fast PU decision

5: if evalPU then
6:     cost2N×N ← interPredSearch(CU, d, 2N × N)
7:     costN×2N ← interPredSearch(CU, d, N × 2N)
8:     cost2N×nU ← interPredSearch(CU, d, 2N × nU)
9:     cost2N×nD ← interPredSearch(CU, d, 2N × nD)
10:    costnL×2N ← interPredSearch(CU, d, nL × 2N)
11:    costnR×2N ← interPredSearch(CU, d, nR × 2N)
12: end if

13: if d = 3 then
14:     costN×N ← interPredSearch(CU, d, N × N)
15: end if

16: costSplit ← +∞

17:  $\vec{f} \leftarrow \text{computeFeatures}(CU, d)$                                 ▷ Extraction Point 2
18: splitCU ← decideCU( $\vec{f}$ , d)                                         ▷ Early CU termination

19: if d < 3 and splitCU then
20:     costSplit ← compressCU(CU, d+1, part0)
21:     costSplit += compressCU(CU, d+1, part1)
22:     costSplit += compressCU(CU, d+1, part2)
23:     costSplit += compressCU(CU, d+1, part3)
24: end if

25:  $\vec{modes} \leftarrow (\text{cost}_{MSM}, \text{cost}_{2N\times 2N}, \dots, \text{cost}_{N\times N}, \text{cost}_{Split})$ 
26: bestMode ← MIN( $\vec{modes}$ , key = RDcost)
27: return bestMode
  
```

---

Table 6.14: Rate-Distortion and complexity reduction achieved using the PU fast decision method, as well as a combined CU+PU decision.

Sequence	PU			CU+PU		
	$BD_{BR}$	TS	$BDTS_{Ratio}$	$BD_{BR}$	TS	$BDTS_{Ratio}$
PeopleOnStreet	0.58	26.7	2.19	1.42	35.3	4.02
SteamLocomotiveTrain	0.47	46.6	1.00	0.85	65.0	1.31
BQTerrace	0.20	45.9	0.43	1.01	64.0	1.57
Cactus	0.48	42.1	1.13	1.85	59.2	3.12
Kimono	1.14	39.7	2.87	2.86	57.1	5.01
BasketballDrill	0.47	36.9	1.29	1.27	52.4	2.42
RaceHorsesC	0.54	21.0	2.56	1.34	30.0	4.46
BasketballPass	0.23	47.6	0.48	0.66	62.6	1.05
BQSquare	0.09	40.9	0.23	0.54	58.0	0.92
Johnny	0.24	64.0	0.37	0.70	84.4	0.83
KristenAndSara	0.06	65.0	0.10	0.57	84.3	0.67
ChinaSpeed	0.79	36.2	2.18	1.98	52.1	3.80
SlideEditing	0.17	69.1	0.25	0.37	89.8	0.41
Average	0.42	44.7	0.94	1.18	61.1	1.94

than the sum of the penalties of each method applied separately. One of the reasons that might explain this is the effect of skipping some PUs with the fast decision method, as explained previously.

With the CU and PU ET methods, most of the CTU quadtree algorithm is covered. The following solutions will focus on reducing the inter-prediction search, which is the method that is called several times in Algorithm 2.

### 6.3.2 RQT Early Termination

The inter-prediction search is composed of two major steps: (i) Motion Estimation, which is responsible for finding the integer and fractional motion vector offsets; and (ii) Residual Quadtree estimation, where the TU blocks are partitioned several times until the partitioning with best rate-distortion is found. For all of these steps, three fast methods were defined, and for all of these methods, the same features used in the PU classifiers were employed to build their respective trees.

The analysis of Chapter 5, which served as foundation for all the methods of this work, showed that in many cases the best partitioning is at the RQT root. Therefore, a RQT Early Termination (RQT ET) scheme was envisioned with the purpose of skipping the entire quadtree evaluation, thus reducing the number of transforms computations.

To train the RQT classifiers, the features obtained from Extraction Point 1 (see

Algorithm 3) were used, and the labels were assigned to 0 when the best TU was at the root node of RQT and to 1 otherwise. Then, the Decision Trees were built using the samples from Extraction Point 1 (line 4 in Algorithm 3). To apply the early termination, the trees were consulted, and their decision was sent as input to every RQT estimation call. Figure 6.10 shows a diagram of the RQT ET application. The results for this approach are presented in Table 6.15.

Figure 6.10: Block diagram of the RQT ET application.

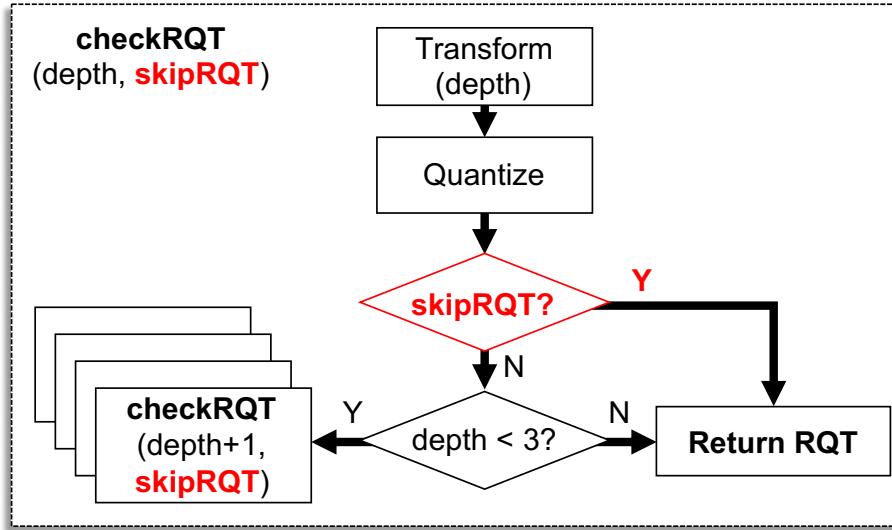


Table 6.15: Rate-Distortion and complexity reduction achieved using the RQT ET method, as well as a combined CU+RQT ET.

Sequence	RQT ET			CU+RQT ET		
	$BD_{BR}$	TS	$BDTS_{Ratio}$	$BD_{BR}$	TS	$BDTS_{Ratio}$
PeopleOnStreet	-1.98	10.3	-19.35	0.78	29.7	2.61
SteamLocomotiveTrain	0.25	10.5	2.35	0.25	57.3	0.44
BQTerrace	0.32	10.3	3.08	1.02	56.7	1.79
Cactus	0.10	10.5	1.00	0.49	51.8	0.95
Kimono	0.41	10.2	4.06	0.95	48.6	1.97
BasketballDrill	-0.04	10.5	-0.38	0.26	44.9	0.57
RaceHorsesC	0.27	8.7	3.06	0.62	26.5	2.33
BasketballPass	0.24	15.4	1.59	0.59	53.7	1.09
BQSquare	0.21	11.8	1.75	0.46	51.1	0.91
Johnny	0.12	11.1	1.07	0.08	72.7	0.11
KristenAndSara	-0.01	10.7	-0.06	0.23	71.4	0.32
ChinaSpeed	1.56	10.1	15.34	2.15	43.4	4.95
SlideEditing	-0.08	12.5	-0.66	0.12	75.2	0.15
Average	0.10	11.0	0.95	0.61	52.5	1.17

Compared with the other methods that were presented so far, reducing the RQT computations has a smaller impact in terms of time savings. However, the BD-BR penalty is proportionally smaller as well. In fact, the  $BDTS_{Ratio}$  of the RQT Early Termination

method is smaller than the PU one, and this remains the case when the RQT and the CU fast methods are jointly used. This time, more than 52% time savings can be achieved at the cost of a BD-BR increment smaller than 1.17%. Therefore, the RQT method can be used along with the CU method to achieve savings close to 50% on average, but the PU method should take place to achieve higher figures. Combining the three methods, the average time savings raises to more than 64% with a 1.39% increase in BD-BR.

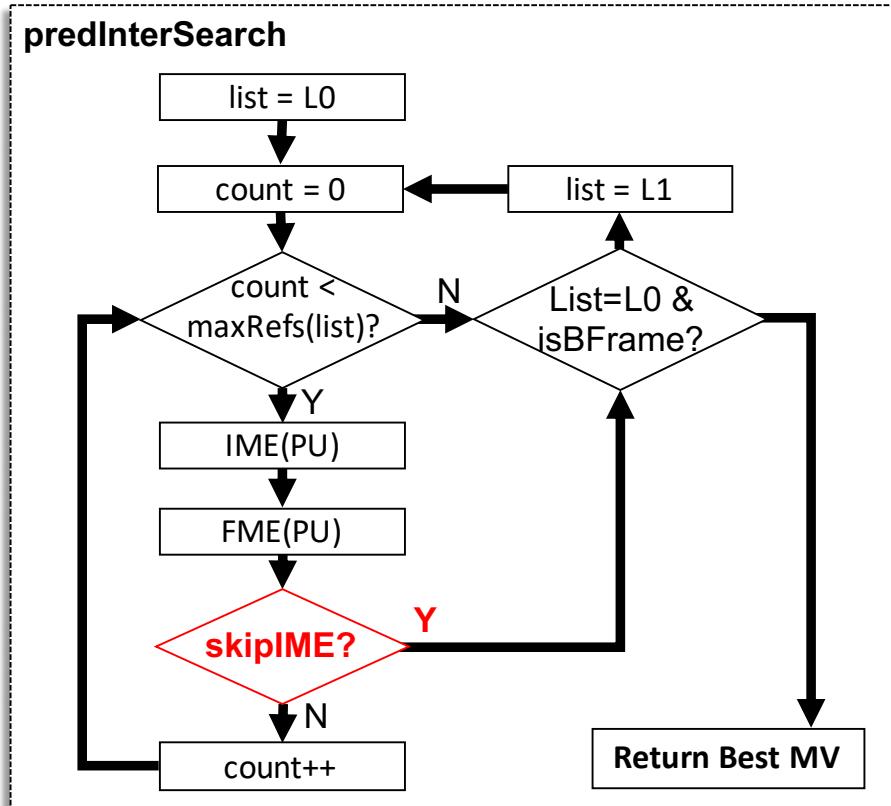
### 6.3.3 ME Early Termination

The IME search complexity was reduced by skipping the reference frames that are searched in this process. Traditionally, the four reference frames are considered: two from each list when the slice is bi-predicted, or four from the L0 list if the slice type is P. Our decision for this process was to search only the first reference of each whenever possible. The first reference of a list is the closest temporal neighbor of the current slice, so more correlation is expected in general. Like with the other trees that were trained in this work, the labels assigned to each example correspond to the task we are modeling, meaning they were assigned to 0, when the best reference of a search was the first frame of the list, and 1 otherwise. The diagram in Figure 6.11 shows how the ME Early Termination (ME ET) method was applied. Table 6.16 shows the rate-distortion complexity results of the fast ME ET, as well as results with a combined CU, PU, RQT, and ME ET strategy.

Skipping reference frames based on our classifiers' decision produces an average BD-BR increase of 0.64% and a time savings of almost 20%. This represents a rate-distortion-complexity efficiency much lower than that observed in the other methods presented so far, which indicates that more effort should be spent on this particular branch of this work. One possible solution is to design more features that are related with this process, for instance the reference index of neighboring PUs. There could also be a problem with how the task was framed, i.e., maybe it is better to predict the exact reference index instead of keeping only the first one of each list. All of these considerations will be assessed in future works.

Still in Table 6.16, we should highlight that combining this method with the previous ones achieves higher time savings, namely 69%, at the cost of 2.13% in BD-BR. The final  $BDT_{Ratio}$  metric points a better tradeoff compared to using the IME alone, which means that the other methods are making up for its reduced efficiency. Therefore, applying this method is only recommended when time savings above 65% are demanded.

Figure 6.11: Block diagram of the ME ET application.



### 6.3.4 FME Early Termination

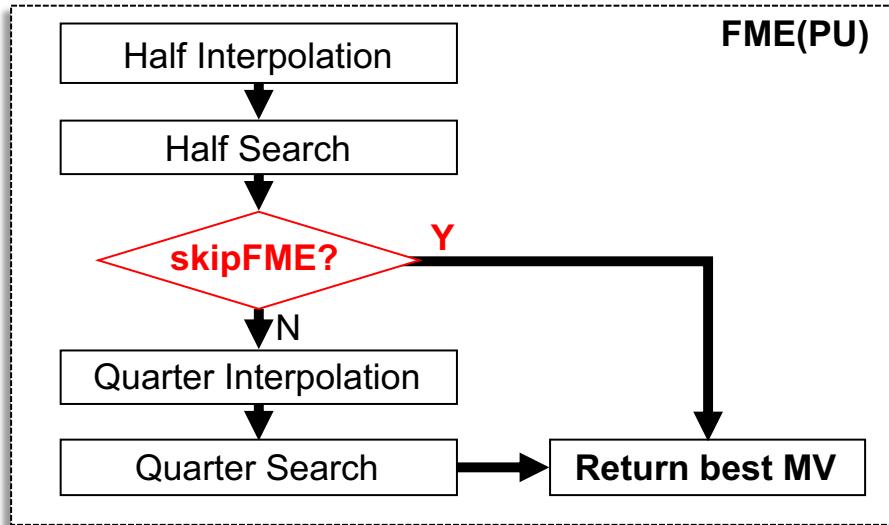
Our final effort in reducing the HEVC encoding complexity targeted the FME search. This process is particularly tricky to optimize, because most of the motion vectors have a fractional offset in some sequences, as discussed in Chapter 5. Nonetheless, Decision Trees were still trained to see if it is possible to predict the necessity of the FME with the extracted features. The labels for the FME classifiers were assigned as True when the final motion vector had a fractional offset, and as False otherwise. Initially, the FME Early Termination (FME ET) method consisted of skipping the entire fractional search when the output of the trees said so, but the BD-BR increase of initial tests were higher than expected. To alleviate this, we only skip the quarter-pixel interpolation and search. A diagram of the RQT ET application is illustrated Figure 6.12. Table 6.17 presents the results of the FME ET method, as well as results with all the previous methods combined with it.

Clearly the fast FME method is not as efficient as any other method tested in this work. The average BD-BR increment was not particularly high, but the time savings it allows does not make up for it, so it has the worst rate-distortion-complexity of all the

Table 6.16: Rate-Distortion and complexity reduction achieved using the RF skip early termination method, as well as a combined CU+PU+RQT+ME Early Termination methods.

Sequence	ME ET			CU+PU+RQT+ME ET		
	$BD_{BR}$	TS	$BDTS_{Ratio}$	$BD_{BR}$	TS	$BDTS_{Ratio}$
PeopleOnStreet	0.45	16.7	2.68	2.17	47.9	4.52
SteamLocomotiveTrain	1.09	18.1	6.04	1.76	71.2	2.47
BQTerrace	1.98	19.3	10.26	4.00	71.5	5.60
Cactus	0.39	18.8	2.05	1.84	67.5	2.73
Kimono	0.29	19.0	1.51	2.85	65.4	4.37
BasketballDrill	0.44	16.9	2.59	2.23	61.6	3.62
RaceHorsesC	0.92	14.0	6.55	2.60	43.1	6.04
BasketballPass	0.23	23.1	1.01	1.45	71.4	2.03
BQSquare	1.19	21.6	5.52	1.96	68.2	2.87
Johnny	0.41	26.5	1.56	1.37	88.4	1.55
KristenAndSara	0.27	27.7	0.97	1.04	88.7	1.17
ChinaSpeed	0.78	14.5	5.38	3.87	60.9	6.35
SlideEditing	-0.12	20.4	-0.58	0.49	91.7	0.53
Average	0.64	19.7	3.24	2.13	69.0	3.08

Figure 6.12: Block diagram of the FME ET application.



methods. When combined with the previous methods, the average time savings increases to almost 71% at the cost of almost 3.8% increment in BD-BR, which is much worse in comparison with the combined results of Table 6.16. These results were presented for completeness, but the overall performance of the FME fast decision method did not prove to be efficient enough to justify its application.

Table 6.17: Rate-Distortion and complexity reduction achieved using the FME skip early termination method, as well as a combined decision with all the proposed methods.

Sequence	FME ET			CU+PU+TU+IME+FME ET		
	$BD_{BR}$	TS	$BDTS_{Ratio}$	$BD_{BR}$	TS	$BDTS_{Ratio}$
PeopleOnStreet	0.94	9.0	10.52	3.15	50.9	6.19
SteamLocomotiveTrain	0.26	3.6	7.33	2.06	71.9	2.87
BQTerrace	0.35	0.4	79.04	4.36	71.8	6.08
Cactus	0.50	10.6	4.68	2.30	69.6	3.30
Kimono	0.20	2.7	7.50	3.10	66.4	4.67
BasketballDrill	0.94	12.2	7.74	3.51	64.6	5.43
RaceHorsesC	0.90	3.4	26.25	3.57	45.2	7.88
BasketballPass	0.76	15.5	4.88	2.45	73.1	3.35
BQSquare	0.45	5.9	7.69	2.95	69.0	4.28
Johnny	0.57	17.4	3.28	2.00	89.2	2.24
KristenAndSara	0.28	16.0	1.73	1.51	89.4	1.69
ChinaSpeed	-0.34	14.0	-2.45	3.54	64.8	5.47
SlideEditing	-0.62	26.2	-2.34	0.31	92.8	0.34
Average	0.40	10.5	3.80	2.68	70.7	3.79

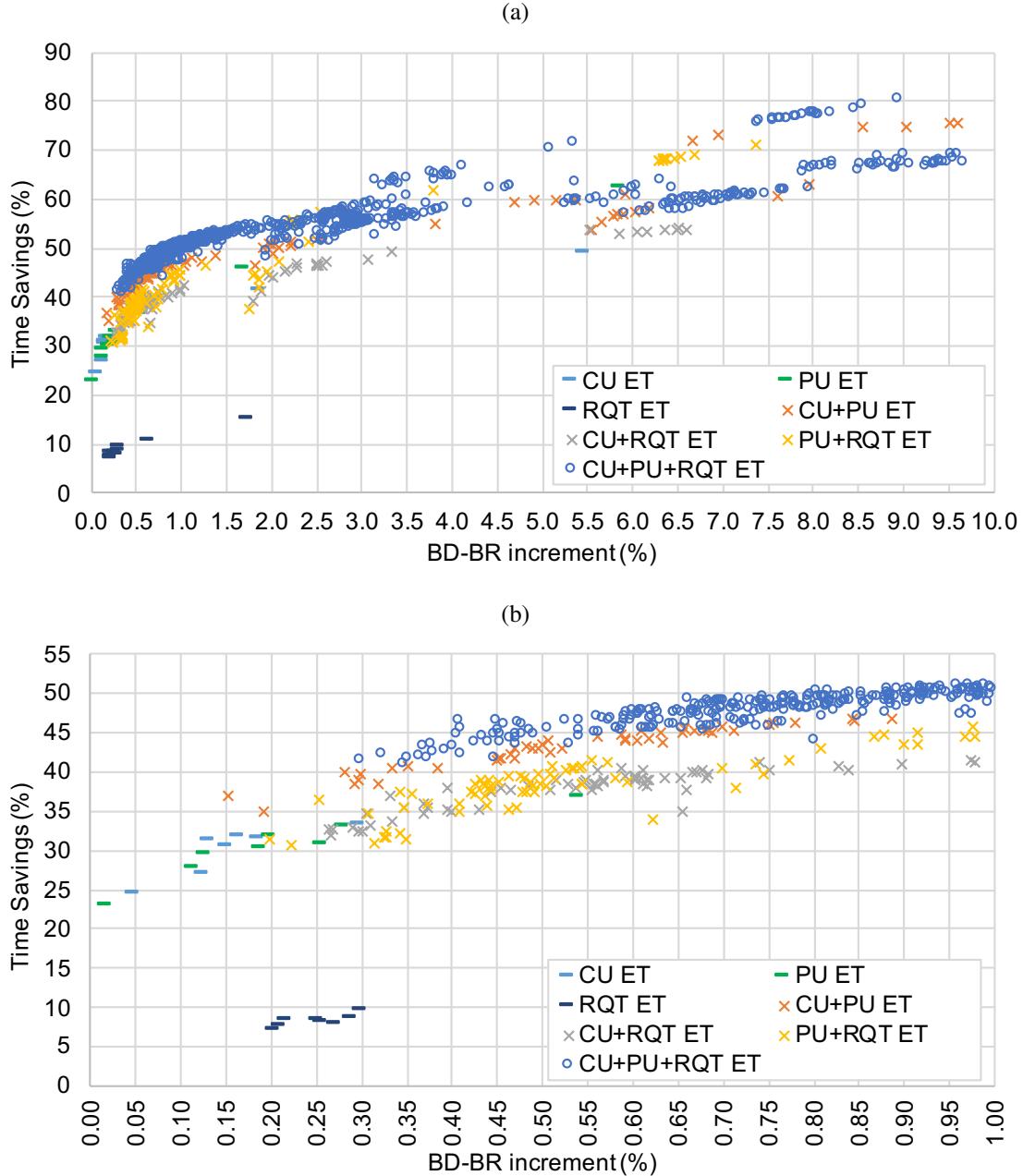
### 6.3.5 Decision Threshold effect on Decision Trees

Based on the positive results obtained in the decision threshold analysis conducted with SVMs and Decision Trees for the CU ET method, it was concluded that similar experiments should be performed combining the other fast decision methods. The problem is that, for each  $N$  values that are tested on  $S$  sequences with four QP values,  $N^M * S * 4$  encodings must be performed, where  $M$  is the number of methods that will be tested. In order to avoid a prohibitive amount of encodings, we limited this analysis for the three best methods, namely the CU, PU, and RQT ET ones. The decision threshold values ranged from 0 to 1 in steps of 0.1. In addition, only two sequences from our training set were encoded: RaceHorsesC and BasketballPass. This configuration resulted in  $11^3 * 2 * 4$ , yielding 10647 encodings.

Figure 6.13 shows a BD-BR/TS plot of all the tested threshold combinations. The points are grouped depending on which methods are jointly used. Up to 92% average time savings were achieved with these experiments, but at the cost of prohibitive 120% increment in BD-BR. These values were omitted by limiting the horizontal axis to a 10% maximum in Figure 6.13(a), and to a 1% maximum in Figure 6.13(b).

The points that lie in the superior border of both figures correspond to the best ones in terms of rate-distortion and complexity. The plot in Figure 6.13(a) demonstrates that significant time savings can be achieved with up to 10% BD-BR increment. More

Figure 6.13: Average BD-BR/TS plot of the tested decision threshold combinations for sequences RaceHorsesC (832x480) and BasketballPass (416x240).



specifically, 80.6% time savings are achieved with a 8.9% BD-BR increment. Note that the best points are usually the ones using all the three ET methods, specially when higher time savings are considered, showing that a combined solution is preferable over one that employs each method individually.

When smaller BD-BR values are targeted, we can see in Figure 6.13(b) that, even for such small BD-BR tolerance, up to 51.1% time savings were achieved, with a 0.99% increment in BD-BR. In general, the combined solutions are still more adequate in most cases. The only cases where a single solution is better is when very small BD-BR in-

crements are targeted, but this also will lead to less time savings. Finally, the RQT ET method performed worse than the other ones in the Rate-Distortion-Complexity (RDC) trade-off, showing that it is inadvisable to use this method individually.

From these results, the 16 best decision threshold combinations in the RDC trade-off were used to encode all of the test sequences. The average results are presented in Table 6.18.

Table 6.18: BD-BR and Time savings achieved using the CU+PU+RQT ET methods for several combinations of decision thresholds

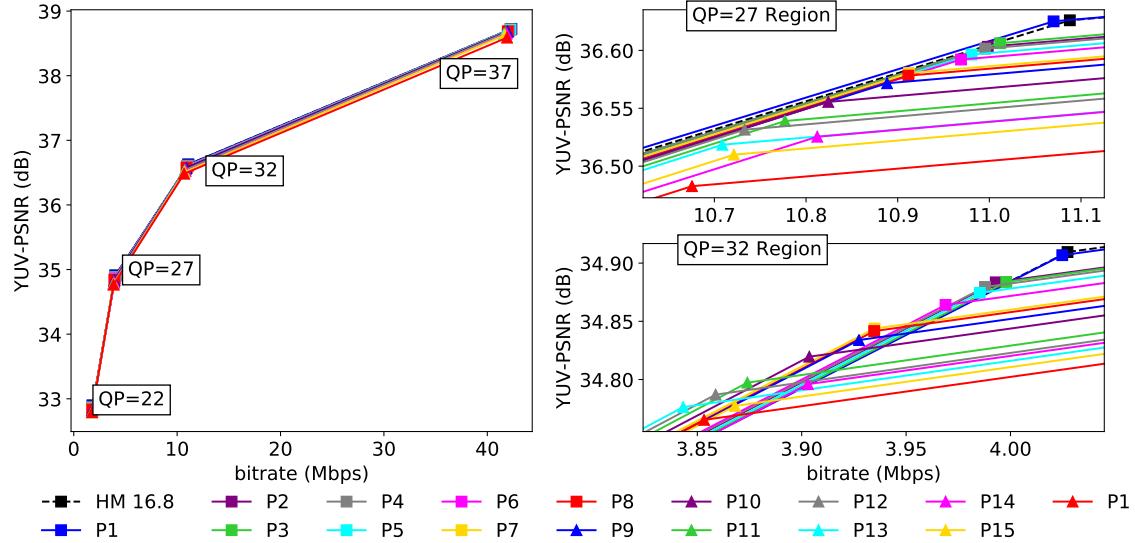
Point	Threshold CU/PU/RQT	Average BD-BR	Average TS	Average BDTS <sub>Ratio</sub>
P1	0/0.1/0	0.04	37.28	0.12
P2	0.4/0/0	0.21	45.84	0.46
P3	0.2/0.1/0	0.27	54.32	0.49
P4	0.3/0.1/0	0.38	55.14	0.68
P5	0.6/0.1/0	0.40	56.30	0.71
P6	0.2/0.1/0.1	0.52	58.13	0.89
P7	0.2/0.2/0.5	0.74	60.07	1.23
P8	0.2/0.3/0.3	0.80	61.87	1.30
P9	0.2/0.6/0.3	0.96	64.16	1.50
P10	0.4/0.6/0.5	1.28	65.28	1.97
P11	0.4/0.7/0.8	1.68	66.55	2.53
P12	0.3/0.7/0.9	1.84	68.25	2.70
P13	0.2/0.8/0.9	2.24	69.00	3.24
P14	0.1/0.9/0.6	2.84	74.85	3.79
P15	0.1/0.9/0.9	3.13	76.66	4.08
P16	0.1/0.9/1	4.80	78.28	6.13

The results demonstrate that several complexity points can be achieved by adequate tuning of the threshold parameters, ranging from 37% up to more than 78%. Note that using the 0.2/0.1/0 combination performed better than the CU ET method for a threshold of 0.5, achieving 54.3% TS with a 0.27% BD-BR increase on average against the 47.8% and 0.24% obtained with the latter.

The rate-distortion plot of the points identified in the above table for the BQTerrace sequence are depicted in Figure 6.14. Since it is difficult to visualize every curve in the entire plot, the right side of the figure contains magnified portions of two QP values.

It is possible to observe that the fast encoding decisions tend to gradually decrease the bitrate as the target time savings increase, but quality is slightly decreased as well at each point. The difference between the reference results and the point with most time savings (P16) is close to 1 dB for the two QPs displayed in the right side of the figure. Also note that the rate-distortion curve for the point with smallest BD-BR increase (P1) almost coincides with the reference results in the magnified charts.

Figure 6.14: Rate-distortion curves of the proposed CU+PU+RQT ET methods using several combinations of decision thresholds (sequence: BQTerrace, resolution: 1920x1080@60fps, QP=22, 27, 32, 37)

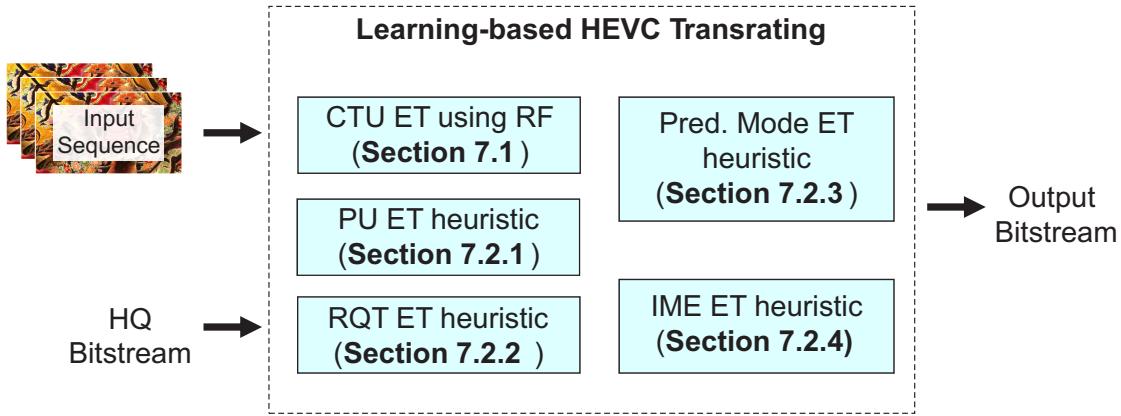


Although there is room for improvement in the proposed methods (especially the ones related to the ME searches), we concluded with these results that the amount of contribution for fast encoding decisions provided in this thesis was already significant, proving that learning-based methods are a promising alternative to predict the complex partitioning decision of encoders. Therefore, we decided to explore a second branch of study in visual applications, which is related to the transcoding complexity in adaptive streaming systems. The following sections will describe the designed methods for this problem.

## 7 LEARNING-BASED MODE DECISION FOR HEVC TRANSRATING

Given its importance in the recent adaptive streaming services, HEVC transcoding is the second application contemplated in this thesis. As mentioned in Chapter 2, the transcoding complexity is usually concentrated on the encoding side. This means that fast encoding decisions can also be used to reduce transcoding complexity, but this type of solution disregards an important source of information: the reference bitstream. Figure 7.1 shows the main contributions of this thesis for HEVC transrating applications, where a reference (HQ) bitstream can be used to guide the decisions of subsequent encodings.

Figure 7.1: Overview of the contributions presented in this thesis for transrating applications.



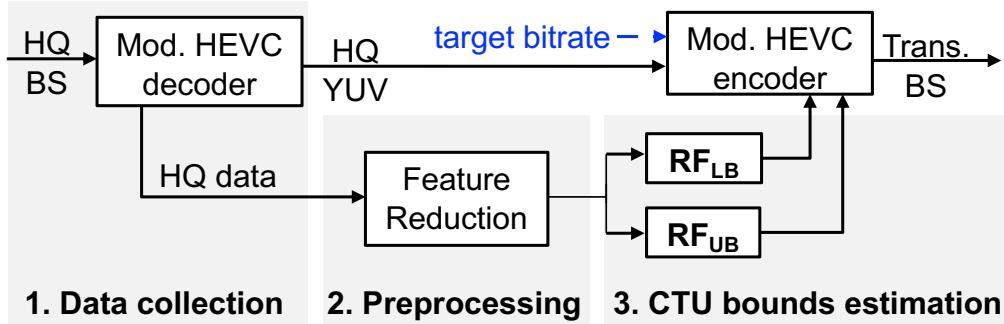
For the transrating analyses and results of this work, the x265 (MULTICOREWARE, 2017) fast HEVC encoder was used, using the "placebo" preset. In addition, the FFMpeg (FFMPEG, 2017) fast HEVC decoder was used in the decoding part. The next sections will explain each of these components, followed by the obtained results.

### 7.1 CTU-level Partitioning Decision for Fast HEVC Transrating with Random Forests

As demonstrated in Chapter 5.2, there is a high correlation between the decisions of the source bitstream and those of the subsequent encodings. Therefore, the HEVC transrating solutions proposed in this thesis focus on exploiting the bitstream information to reduce the transcoding complexity.

Our first initiative consisted in a speedup method for HEVC transrating that operates by determining Lower and Upper bounds for the depths to be tried in the search for the best CTU partitioning quadtree. The designed algorithm is based on classifiers that use information from a HQ bitstream from which the transrated bitstream is to be

Figure 7.2: Methodology used to apply the trained Random Forests in a fast HEVC encoder.



obtained, following the adaptive streaming setup depicted in Figure 2.7. These classifiers are designed (and evaluated) using Random Forests and trained using features and datasets collected from a set of pre-encoded and transrated videos. The features used are the ones discussed in Chapter 5.2.2. The full set of features is listed in Appendix B.

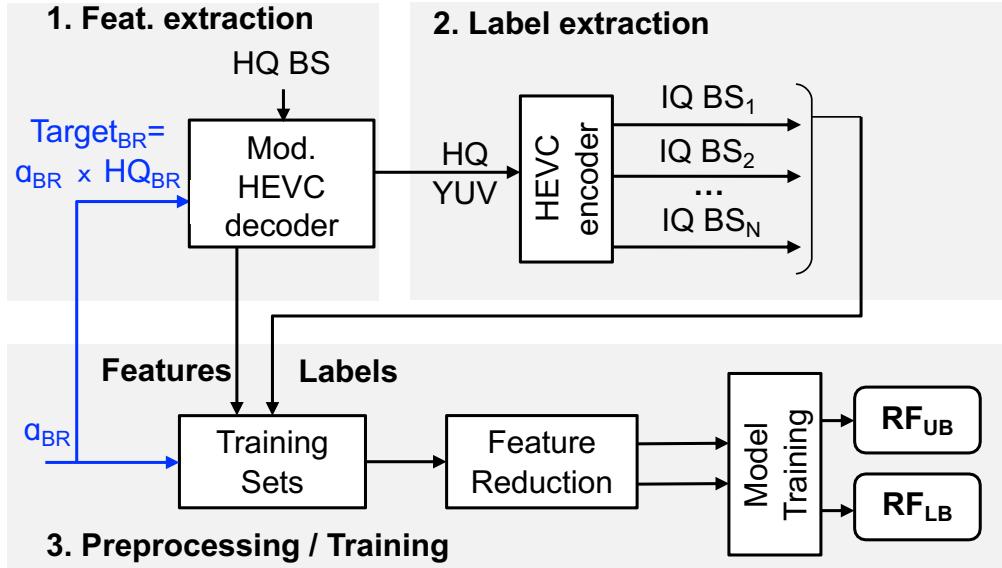
In the proposed scheme the encoding stage of the transcoding operation restricts the quadtree depths of each CTU to be between a Upper (UB) and an Lower (LB) bound. These bounds are predicted by two algorithms, as represented in Figure 7.2. As will be explained, these two algorithms take the form of Random Forests (RF), which have as inputs features derived from a High-Quality (HQ) bitstream and extracted during the bitstream decoding. Note that the prediction of these UB and LB is not a binary classification problem, as there are four possible outcomes for each classifier, namely depth levels 0 through 3 corresponding to CU sizes  $64 \times 64$ ,  $32 \times 32$ ,  $16 \times 16$  and  $8 \times 8$ . Computational savings exist when the number of levels between those bounds is on average smaller than the unrestricted full-search across the 4 levels. However, minimizing computational savings is only one the objectives of the method, as it is important that these savings do not result in large degradation of encoding performance, i.e., the coding performance loss also has to be minimized. To achieve a good balance between these two conflicting objectives we resorted to Machine Learning techniques to train a set of Random Forests which will act as the depth bound estimators.

The methodologies for training and testing the classifiers used in this work are shown in Figure 7.3 and 7.2 respectively.

As depicted in Figure 7.3, the training phase is divided into the following steps:

**Raw feature extraction:** in this step, several train sequences are HQ encoded into CBR bitstreams using as target the average bitrate obtained when the same videos are encoded with QP value 22. Then each bitstream is parsed using a modified FFm-

Figure 7.3: Methodology used to train the CTU-level classifiers.



peg (FFMPEG, 2017) decoder to extract information such as CTU partitioning statistics, transform coefficient values, and motion vectors. This information is gathered for each CU and post-processed to obtain secondary features.

**Label Extraction and Dataset Assembly:** the second step consists in encoding several representations with bitrates that are smaller than the average bitrate used in the HQ bitstream encoding, producing Lower-Quality (LQ) bitstreams. The LQ bitstreams are then parsed to obtain the maximum and minimum CU partitions of each CTU. These two values are used to label our previously extracted features, generating distinct datasets: the maximum CU sizes are used to form the Data<sub>UB</sub> set, and the Data<sub>LB</sub> set is formed with the minimum CU sizes as labels. The target bitrate used to obtain each LQ bitstream was computed using the formula  $\alpha_{BR} * BR_{HQ}$ , where  $BR_{HQ}$  represents the bitrate of the HQ bitstream, and  $\alpha_{BR}$  took values from 0.1 to 0.9 in steps of 0.1.

**Dataset Preprocessing:** Once Data<sub>UB</sub> and Data<sub>LB</sub> are generated, additional preprocessing is carried out to increase the quality of our training samples by balancing the datasets using random undersampling. This ensures that all labels (minimum and maximum quadtree depths) are equally represented in the datasets.

**Random Forest Training:** The next step is the training of the two depth bound predictors. After extensive experimentation not reported here, RF were selected, as experimental results showed that they performed better than other techniques in terms of complexity reduction and coding performance. Furthermore, empirical evaluations showed that the best performance was achieved using forests with 20 trees.

In the test phase, illustrated in Figure 7.2, the HQ bitstream is decoded using the modified FFmpeg software. The features extracted from the decoder are sent to the classifiers, and the output decisions create a CTU map, which contains the upper and lower bounds of each CTU that will be encoded. The x265 encoder was modified to read this data and to limit the CUs whenever they are not within the Upper/Lower Bound range. The x265 fast encoder was used with the placebo<sup>1</sup> preset under an IPBBB GOP structure during our train and test phases. Adaptive B-frames and scene-cut detection were disabled to allow a coherent mapping between the decoded and the re-encoded data.

The sequences assigned to the train and test sets are listed in Table 7.1. Two seconds of each sequence were encoded.

Table 7.1: Train and test sequences used in the fast transcoding schemes.

Class	Name	Resolution	FPS	Train/Test
A	Traffic	2560x1600	30	Train
B	BasketballDrive	1920x1080	50	Train
	ParkScene	1920x1080	24	Train
C	BQMall	832x480	60	Train
	PartyScene	832x480	50	Train
D	BlowingBubbles	416x240	50	Train
	RaceHorses	416x240	30	Train
A	PeopleOnStreet	2560x1600	30	Test
B	BQTerrace	1920x1080	60	Test
	Cactus	1920x1080	50	Test
C	RaceHorsesC	832x480	30	Test
	BasketballDrill	832x480	50	Test
D	BasketballPass	416x240	50	Test
	BQSquare	416x240	60	Test

### 7.1.1 Feature Design and Processing

The CTU-level features discussed in Chapter 5.2 were extracted to train the Upper and Lower Bound classifiers. Since the target bitrate has an important impact on the partitioning, the  $\alpha_{BR}$  value of each LQ encoding is also used as a feature. Combining all the information, a total of 58 features are defined.

Some of these features may not be useful to train the classifiers, so they can be discarded without affecting significantly the performance of the overall scheme. Further-

---

<sup>1</sup>One of the x265 predefined configurations.

more, less features are helpful in complexity-aware solutions, because this reduces the time of the feature extraction and decision computation routines. A smaller feature set also reduces the input size of classifiers, which is advantageous for systems with tight storage/communication constraints. Two methods of FR were tested: feature selection, where only the most relevant non-redundant features are kept to train the classifiers; and a transformation method that employs autoencoders (AE) (CHOLLET et al., 2015) to transform the original features into a new lower dimensional space.

**Feature Selection:** the feature selection followed a traditional filtering approach (DASH; LIU, 1997). Different feature sets were prepared using different ranking strategies. In one case, two feature subsets were constructed using Information Gain Ratio (IGR) as a dependency indicator between the features and the UB and LB predictions. Another two feature subsets were constructed using a similar procedure but with Information Gain (IG) (WITTEN et al., 2016) as the utility function. In all four feature sets only the higher ranked  $N$  were kept to be later used as input to the classifiers. The complete list of IGR scores for these features is found in Appendix B. As an example, Table 7.2 shows the top 10 features for the UB and LB decisions ranked by Information Gain.

Table 7.2: Top 10 Features Ranked by Information Gain

Upper Bound		Lower Bound	
Feature	IG	Feature	IG
Depth <sub>Avg</sub>	0.84	Depth <sub>Avg</sub>	0.64
Depth <sub>Min</sub>	0.77	Depth <sub>Max</sub>	0.58
Depth <sub>Max</sub>	0.48	Bits <sub>Avg</sub>	0.42
Bits <sub>Avg</sub>	0.45	Depth <sub>Min</sub>	0.38
PredSum <sub>Avg</sub>	0.43	PredMode <sub>Avg</sub>	0.38
PredSum <sub>Max</sub>	0.42	TrSize <sub>Avg</sub>	0.32
CoeffSum <sub>Max</sub>	0.40	CoeffSum <sub>Max</sub>	0.32
PredMode <sub>Avg</sub>	0.39	QCoeffSum <sub>Avg</sub>	0.32
TrSize <sub>Avg</sub>	0.39	PU <sub>Avg</sub>	0.30
QCoeffSum <sub>Avg</sub>	0.39	SkipFlag <sub>Avg</sub>	0.30

**Feature transformation:** as explained, the aim of using autoencoders is to transform the initial set of features into a smaller one with the smallest possible degradation in performance. Our autoencoder, has three distinct layers: input layer, intermediate layer and output layer. The input layer has 58 nodes, corresponding to the number of features in the original training set. The intermediate layer will have a number of nodes corresponding to the number of features that we want in our transformed training set (which ranged from 5 to 30). Similarly to the input layer, the output layer has also 58 nodes, being its output a reconstruction version of the transformed data, which ideally would be

the original training set.

Training AEs was particularly challenging, as Neural Networks have a large number of parameters that must be fine-tuned to achieve the best results. Besides that, convergence wasn't verified without input data normalization. Starting by that topic, each feature was normalized to a 0-1 range without altering their relative distribution. Using the Mean Squared Error loss function and the Adam optimizer, we observed convergence after training 50 epochs with batches of 1000 training examples. As activation function, we used the sigmoid function.

The next section will present transrating results using the proposed CTU-level fast CU decision based on RF trained with the complete and reduced feature sets.

### 7.1.2 Results and Discussion

The performance of the proposed transrating scheme was evaluated empirically, by comparing its time complexity and encoding efficiency to those of a reference Cascaded Pixel-Domain Trascoding (CPDT) implementation, also based on the x265 encoder. To compare the encoding efficiency, the Bjontegaard-Delta Bitrate (BD-BR) (BJONTEGAARD, 2001) was calculated using the (PSNR, bitrate) pairs of the two transraters (reference and ours) for four operation points corresponding to  $\alpha_{BR} = 0.2, 0.4, 0.6, 0.8$ .

The computational complexity reduction was computed as  $TS = (1 - T_{test}/T_{ref}) * 100$ , where  $T_{test}$  is the encoding time using the proposed method, and  $T_{ref}$  the reference transrater encoding time. All evaluations used the test videos listed in Table 7.1. The  $BDTS_{Ratio}$  was used to measure how much the BD-BR increases for each TS unitary gain. Table 7.3 shows the results obtained by the proposed method with the full set of features, without any reduction.

Table 7.3: Transrating results using the full set of features

<b>Sequences</b>	<b>BD-BR</b>	<b>Time Savings</b>	<b>BDTS<sub>Ratio</sub></b>
BQSquare	0.03	39.10	0.07
BasketballPass	0.25	40.49	0.61
RaceHorses	0.43	39.82	1.08
BasketballDrill	0.52	47.41	1.11
BQTerrace	0.06	39.98	0.14
Cactus	0.37	45.65	0.80
PeopleOnStreet	0.35	40.24	0.87
<b>Average</b>	<b>0.29</b>	<b>41.81</b>	<b>0.68</b>

It is possible to observe that BD-BR increment varied from 0.03 up to 0.52%,

whereas the TS varied from 39.10 to 47.41%. On average, 41.81% TS was achieved at the expense of a tolerable 0.29% BD-BR increment. This reduction in complexity is significant considering that it is very difficult to optimize a fast implementation like x265.

The three feature reduction methods described in the previous section were also evaluated. The number of features in the reduced feature sets ( $N$ ) were 5, 10, 15, 20, 25 and 30. The transrating performance results obtained for each FR method and number of features are listed in Tab. 7.4.

Table 7.4: Average results for all dimensionality reduction methods and number of features

<b>Nb. of Features</b>	5	10	15	20	25	30
<b>Information Gain Ratio</b>						
<b>BD-BR</b>	0.93	0.58	0.49	0.48	0.49	0.59
<b>TS</b>	41.37	40.74	40.77	40.55	40.4	40.77
<b>BDTS<sub>Ratio</sub> (<math>\times 10^2</math>)</b>	2.21	1.37	1.17	1.14	1.17	1.41
<b>Information Gain</b>						
<b>BD-BR</b>	1.26	0.52	0.46	0.45	0.52	0.47
<b>TS</b>	42.53	40.7	40.67	40.37	40.71	40.37
<b>BDTS<sub>Ratio</sub> (<math>\times 10^2</math>)</b>	2.92	1.26	1.1	1.06	1.26	1.11
<b>Autoencoder</b>						
<b>BD-BR</b>	1.1	0.69	0.62	0.53	0.53	0.55
<b>TS</b>	40.85	43.26	42.96	42.44	42.92	42.39
<b>BDTS<sub>Ratio</sub> (<math>\times 10^2</math>)</b>	2.68	1.59	1.44	1.25	1.24	1.29

We can observe that the effect of using less features is mixed. In some cases, it tends to increase the TS with a slight increase in BD-BR, but in other cases the TS decrease. The ideal number of features is within the 20-25 range, as the best averages are in those two cases. Reducing the features to almost a third of its initial dimension using the IGR as metric yields a TS of more than 40% with a tolerable BD-BR increment of 0.49%. In some cases, this trade-off between performance and input size could be advantageous, as explained in the previous section.

The best results were obtained using Information Gain as metric, but the IGR-based selection performed similarly, showing that these two metrics work well with Random Forests. The AE-based reduction achieved the worst results in most cases, although it provided the highest TS.

Tab. 7.5 presents the best results achieved with each feature reduction method.

Considering the selection based on IGR, the lowest BDTS<sub>Ratio</sub> (1.14%) was achieved with 20 features. In the IG results, the best performance was also verified when the number of features was 20 with a BD-BR/TS ratio equal to 1.06%. In the case of autoencoders,

Table 7.5: Best results for each Dimensionality Reduction method

Information Gain Ratio			
Sequences	BD-BR	Time Savings	Ratio
BQSquare	0.04	34.32	0.11
BasketballPass	0.52	38.34	1.35
RaceHorses	0.51	39.84	1.28
BasketballDrill	0.83	47.28	1.77
BQTerrace	0.16	38.71	0.41
Cactus	0.67	44.81	1.49
PeopleOnStreet	0.64	40.58	1.58
<b>Average</b>	0.48	40.55	1.14
Information Gain			
Sequences	BD-BR	Time Savings	Ratio
BQSquare	0.04	33.27	0.13
BasketballPass	0.42	38.78	1.08
RaceHorses	0.50	39.57	1.27
BasketballDrill	0.85	47.64	1.78
BQTerrace	0.11	37.83	0.28
Cactus	0.63	44.94	1.41
PeopleOnStreet	0.60	40.57	1.47
<b>Average</b>	0.45	40.37	1.06
Autoencoder			
Sequences	BD-BR	Time Savings	Ratio
BQSquare	0.43	40.48	1.07
BasketballPass	0.66	41.03	1.60
RaceHorses	0.57	40.88	1.38
BasketballDrill	0.82	49.22	1.67
BQTerrace	0.24	41.84	0.57
Cactus	0.49	46.02	1.07
PeopleOnStreet	0.53	40.96	1.30
<b>Average</b>	0.53	42.92	1.24

the best result (1.24%) was achieved with 25 features. It can be observed that when reducing the number of features by more than 50%, the degradation in model performance is very small with a BD-BR increment under 1% (except when 5 features were used) and average TS over 40%.

Table 7.6 presents a comparison of the proposed method with other transrating methods published in the literature. Note that those competing works did not publish results using the x265 software, so this comparison is not completely fair. This is an unavoidable problem, as there is a lack of references that use x265 in the encoding step, perhaps due to the difficulty of modifying this encoder.

We conclude that our method outperforms those competing works in terms of combined BD-BR and TS. The BD-BR/TS ratio of (VAN et al., 2016) is 4.3 times higher

Table 7.6: Performance comparison with related works

Method	BD-BR	Time Savings	BDTS <sub>Ratio</sub>
Proposed	0.29	41.81	0.69
(VAN et al., 2016)	1.89	63.68	2.96
(YANG; ZHONG, 2017)	2.26	55.01	4.10

than ours, and the ratio of (YANG; ZHONG, 2017) is almost six times higher (precisely 5.94). The TS of the other references is higher than ours, but this could be attributable to the use of HEVC reference software which is a slower implementation with larger margin for optimization. Finally, the BD-BR of our method is unrivaled and much smaller, showing that our solution is preferable for systems that aim at reducing the transcoding complexity while keeping the quality virtually unchanged.

## 7.2 Statistical- and Learning-Based Decisions for Fast HEVC Transrating

The CTU-level decisions showed promising results in terms of compression and time savings, but only the CU partitioning was targeted. From the fast encoding results obtained in the previous section, we concluded that much more savings can be achieved if other tools are also exploited with fast decisions. Therefore, we decided to use the knowledge obtained from the bitstream analysis of Chapter 5 to design fast decision strategies for the Prediction Unit decision, the RQT evaluation, Motion Estimation, and prediction modes. The following section will explain each of these strategies. The analysis of Chapter 5.2 will be used as foundation for all the methods described this section. This set of experiments followed the same methodology applied in section 7.1.2, with the exception that 50 frames were encoded instead of the entire sequence. This was necessary because a huge amount of simulations were carried out to produce these results.

Before starting with the other decisions, we first implemented a CU early termination method, which consists of skipping the partitioning process if the current CU was not partitioned in the HQ bitstream. The reasoning for this comes from the CU size analysis presented in Chapter 5.2, which shows that the size of CUs in the LQ encodings is the same or larger most of the time. The purpose of this is to assess the efficiency of our CTU-level decision using Random Forests. A simplified pseudo-code of the CU decision is presented in Algorithm 4.

Note that this type of CU ET method only reduces the time spent on CUs below the depth<sub>HQ</sub> in CTU quadtree, so the expected complexity reduction of this approach is

---

**Algorithm 4** CU ET algorithm using the HQ bitstream information
 

---

**Input:** CU: current CU block  
**depth:** current CU depth  
**depth<sub>HQ</sub>:** depth of the co-located CU in the HQ bitstream

```

1: testSplit  $\leftarrow 0$ 
2: if  $\text{depth}_{HQ} \leq \text{depth}$  then
3:   compute RD Cost for the current CU
4:   testSplit  $\leftarrow 1$ 
5: end if
6: if  $\text{depth} < 3$  and testSplit then
7:   compress(CU, depth+1, part0)
8:   compress(CU, depth+1, part1)
9:   compress(CU, depth+1, part2)
10:  compress(CU, depth+1, part3)
11: end if
12: return Best CU partitioning
  
```

---

smaller than the one obtained with the CTU<sub>RF</sub> ET method. The results are given in Table 7.7. Note that the CTU<sub>RF</sub> ET results are different from the ones presented in 7.3, because in these tests the frame count was reduced to 50.

Table 7.7: Performance of CU depth limitation strategy using direct mapping from Data<sub>HQ</sub> and comparison with the learning-based CTU-level fast decision

Sequence	BD-BR	TS	BDTS <sub>Ratio</sub>
PeopleOnStreet	1.07	19.16	5.61
Cactus	0.98	36.56	2.69
BQTerrace	0.48	28.60	1.67
BasketballDrill	0.91	36.35	2.50
RaceHorsesC	1.15	24.14	4.77
BQSquare	1.23	22.95	5.35
BasketballPass	0.72	28.41	2.52
<b>Average</b>	<b>0.93</b>	<b>28.02</b>	<b>3.33</b>
<b>CTU<sub>RF</sub> ET</b>	<b>0.53</b>	<b>40.98</b>	<b>1.30</b>

The results show that limiting the lower depth using the depth from the HQ stream directly does not perform well in terms of combined rate-distortion and complexity. The BD-BR increment is still inferior to 1%, which is not detrimental, but the achieved time savings was lower than 30%. Compared to the CTU-level decision, it is possible to conclude that the learning-based approach is much more efficient, achieving more savings with a smaller BD-BR increment. Therefore, this method will not be considered in the remainder of this section.

### 7.2.1 PU Early Termination Heuristic

The PU partitioning decision was a natural follow-up of this work, because most of the times the PUs selected on LQ encodings are either the same or larger than the ones found in the HQ bitstream ( $PU_{HQ}$ ), as demonstrated in Chapter 5.2. Therefore, a partition map was defined based on the  $PU_{HQ}$  value, which is described in Table 7.8.

Table 7.8: PU partitions evaluated in the fast PU strategy using the information from the HQ reference ( $PU_{HQ}$ )

$PU_{HQ}$	PUs evaluated in the LQ encodings
2Nx2N	2Nx2N, 2NxN, Nx2N
2NxN	2Nx2N, 2NxN, 2NxnU, 2NxnD
Nx2N	2Nx2N, Nx2N, nLx2N, nRx2N
NxN	All PUs
2NxnU	2Nx2N, 2NxN, 2NxnU
2NxnD	2Nx2N, 2NxN, 2NxnD
nLx2N	2Nx2N, Nx2N, nLx2N
nRx2N	2Nx2N, Nx2N, nRx2N

This mapping guarantees that, except when  $PU_{HQ}$  is NxN, three to four PUs are evaluated instead of the seven ones that are computed in the worst case of the original x265 implementation. Most savings are achieved when the  $PU_{HQ}$  is 2Nx2N. Note that the 2NxN PUs are still processed when the  $PU_{HQ}$  is equal to 2NxnU or 2NxnD, analogously for Nx2N PUs. That is because the table in Chapter 5.2 showed that there is a small probability for this case. Other mappings were tested, but this configuration gave the best rate-distortion-complexity performance, specifically 15% time savings with a 0.36% BD-BR increment.

### 7.2.2 RQT Early Termination Heuristic

A similar strategy was adopted for the RQT partitioning. It was observed that the transform size of the HQ input is usually the same or larger on LQ encodings. Therefore, the maximum RQT depth from the HQ bitstream ( $RQTd_{HQ}$ ) was used to limit the depth of the LQ encodings. The time savings achieved with this approach were 12.7% with a BD-BR increment of 0.18% on average. This is better illustrated in Algorithm 5.

---

**Algorithm 5** RQT ET algorithm using the HQ bitstream information
 

---

**Input:** TU: current TU block  
**RQTd:** current TU depth  
**RQTd<sub>HQ</sub>:** minimum RQT depth of the co-located CU in the HQ bitstream

```

1: testSplit ← 1
2: compute Transforms and Quantization for the current TU
3: if RQTdHQ ≤ RQTd then
4:   testSplit ← 0
5: end if
6: if RQTd < 3 and testSplit then
7:   checkRQT(TU, RQTd+1, part0)
8:   checkRQT(TU, RQTd+1, part1)
9:   checkRQT(TU, RQTd+1, part2)
10:  checkRQT(TU, RQTd+1, part3)
11: end if
12: return Best RQT partitioning
  
```

---

### 7.2.3 Prediction Mode Early Termination Heuristic

Based on the prediction mode analysis, it was concluded that we cannot infer much from inter or SKIP CUs in the HQ information, except that they will most likely not be intra coded in the LQ versions. Therefore, the intra-prediction was disabled whenever a CU is encoded as inter or SKIP mode. This strategy was named Prediction Mode Early Termination (PM ET) and enabled savings of 16.8% with a BD-BR increase of 0.36%.

### 7.2.4 ME Early Termination Heuristic

The MV analysis showed that the best MV found on the LQ encodings are usually the same one from the HQ reference or in the vicinity, so the Integer Motion Estimation step was modified to check the rate-distortion cost of the MV<sub>HQ</sub> as well. If the best cost is found with this MV, than the search range is reduced to 4, otherwise, the search is performed normally with the input search range. Algorithm 6 presents the pseudo-code of this method, and Table 7.9 shows the obtained time savings and BD-BR results.

As expected, complexity is not significantly reduced, because the fast ME method implemented in the x265 software is very efficient. Nonetheless, this technique is able to improve coding efficiency in almost all of the test sequences. This is likely because the inherited MV from the reference bitstream is better than the predicted MV used in

---

**Algorithm 6** ME ET algorithm using the HQ bitstream information
 

---

**Input:** PU: current PU block

MV<sub>Pred</sub>: predicted MV

MV<sub>HQ</sub>: MV of the co-located CU in the HQ bitstream

SR: Search Range

```

1: MVstart ← MVPred
2: for each MVPcand in MVPList do
3:   computeRateDistortion(MVPcand)
4:   if rdCostMVP < rdCostPredMV then
5:     MVstart ← MVPcand
6:     rdCostBest ← rdCostMVP
7:   end if
8: end for

9: computeRateDistortion(MVHQ)
10: if rdCostHQ < rdCostBest then
11:   MVstart ← MVHQ
12:   SR ← 4
13:   rdCostBest ← rdCostHQ
14: end if

15: MVBest ← interSearch(PU, MVstart, SR)
16: return MVBest
  
```

---

Table 7.9: Performance of MV inheritance strategy using the vector from the reference stream as candidate.

Sequence	BD-BR	TS	BDTS <sub>Ratio</sub>
BQSquare	0.01	-0.11	-8.80
BasketballPass	-0.27	4.95	-5.49
BasketballDrill	-0.45	5.17	-8.72
RaceHorsesC	-0.39	4.98	-7.87
Cactus	-0.22	4.40	-4.99
BQTerrace	-0.42	0.87	-47.90
PeopleOnStreet	-0.40	3.47	-11.46
Average	-0.31	3.39	-9.02

the original implementation. Therefore, this method may not stand out if used alone, but it is useful to mitigate the compression penalties introduced by other approaches, as demonstrated in the following section.

### 7.2.5 Results and Discussion

Table 7.10 summarizes the results of the methods described in this section. Each group of results was obtained with a method employed separately.

Table 7.10: BD-BR and Time Savings of each fast transcoding method employed separately

Sequence	CU ET		PU ET		RQT ET		Pred. Mode ET		ME ET	
	BD <sub>BR</sub>	TS	BD <sub>BR</sub>	TS						
BQSquare	1.23	23.0	0.66	12.9	0.43	9.7	0.28	29.0	0.01	-0.1
BasketballPass	0.72	28.4	0.41	14.7	0.07	12.4	0.44	16.6	-0.27	5.0
BasketballDrill	0.91	36.4	0.33	12.8	0.07	13.2	0.51	14.0	-0.45	5.2
RaceHorsesC	1.15	24.1	0.49	18.6	0.26	13.3	0.43	12.0	-0.39	5.0
Cactus	0.98	36.6	0.11	13.7	0.12	14.7	0.47	17.7	-0.22	4.4
BQTerrace	0.48	28.6	0.13	13.1	0.17	11.7	-0.03	17.8	-0.42	0.9
PeopleOnStreet	1.07	19.2	0.36	19.8	0.19	13.6	0.42	10.4	-0.40	3.5
Average	<b>0.93</b>	<b>28.0</b>	<b>0.36</b>	<b>15.1</b>	<b>0.19</b>	<b>12.7</b>	<b>0.36</b>	<b>16.8</b>	<b>-0.31</b>	<b>3.4</b>

From these results, we can observe that employing the methods separately does not bring a significant complexity reduction, but the compression efficiency is also not severely compromised. The results vary among sequences. The performance does not seem to be related with the spatial resolution, but with specific scene characteristics. For instance, the BasketballPass results are always superior to the ones obtained with the BQSquare sequence, and both have the same resolution ( $416 \times 240$ ). The following section will extend this analysis, assessing the performance of the combined strategies.

After assessing the performance of each early termination method separately, the next step consisted in analyzing how they perform when combined with one another, and

also when combined with the CTU-level solution using Random Forests.

Table 7.11 presents the BD-BR and complexity reduction of the statistical-based decisions. The cells are sorted by the  $\text{BDTS}_{\text{Ratio}}$  value, so the first cell can be interpreted as the most efficient solution in terms of compression and complexity combined. In addition, combinations that were irrelevant were omitted. For instance, using the combined PU+PM+ME methods, a reduction of 12.7% is achieved with 0.19% increment in BD-BR, which is outperformed by the TU+ME combination, with 16.3% time savings and -0.1% BD-BR increment.

Table 7.11: Rate-Distortion and Complexity Performance of all the statiscal-based method proposed, as well as their best combinations (in terms of combined BD-BR/TS ratio)

Methods	BD-BR	TS	$\text{BDTS}_{\text{Ratio}}$
ME ET	-0.31	3.39	-9.02
TU+ME ET	-0.10	16.31	-0.61
PU+ME ET	0.01	16.92	0.06
PM+ME ET	0.01	19.64	0.07
PU+TU+ME ET	0.19	28.65	0.68
TU+PM+ME ET	0.27	32.52	0.82
PU+PM+ME ET	0.37	33.36	1.11
PU+TU+PM+ME ET	0.57	44.54	1.27

This table shows that several complexity points can be achieved combining the proposed methods, with savings ranging from 3.4% up to 44.5%. Note the fast MV technique is present in every entry of the table, proving that it works well even when combined with other methods. In fact, this technique was able to increase the performance of every other combination. Table 7.12 shows a similar analysis, but this time using the the CU partitioning decision discussed in section 7.1 as well to further increase the complexity reduction. Once again the irrelevant combinations were omitted.

Table 7.12: Rate-Distortion and Complexity Performance of all the statiscal-based methods combined with the CTU-level fast decision.

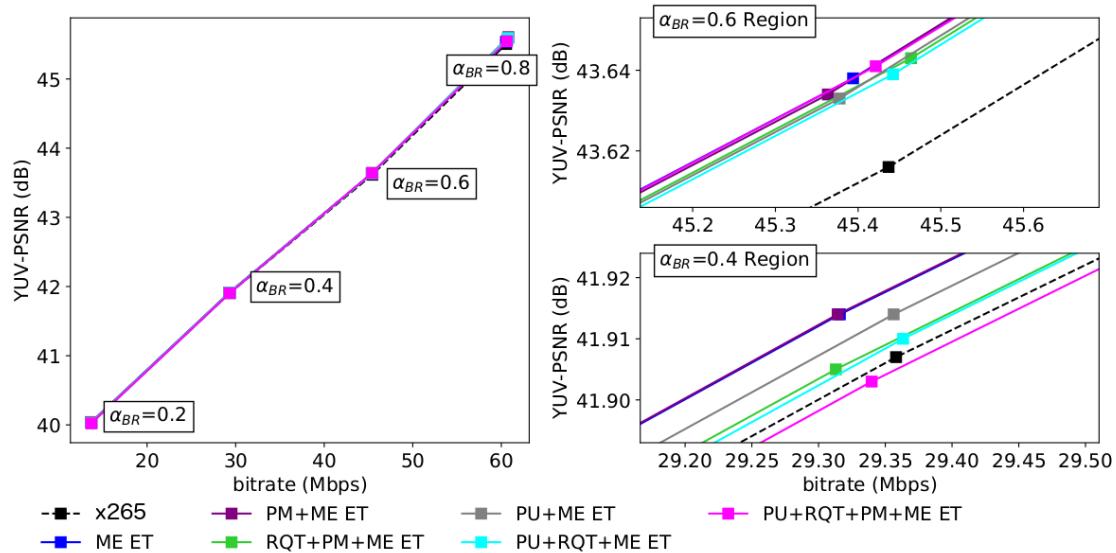
Methods	BD-BR	TS	$\text{BDTS}_{\text{Ratio}}$
$\text{CTU}_{RF}+\text{ME ET}$	0.34	43.04	0.80
$\text{CTU}_{RF}+\text{PU+ME ET}$	0.67	50.50	1.33
$\text{CTU}_{RF}+\text{PM+ME ET}$	0.86	53.94	1.60
$\text{CTU}_{RF}+\text{PU+TU+ME ET}$	1.08	57.24	1.89
$\text{CTU}_{RF}+\text{TU+PM+ME ET}$	1.21	61.25	1.97
$\text{CTU}_{RF}+\text{PU+TU+PM+ME ET}$	1.73	67.54	2.56

When the CTU-level is combined with the statiscal-based early termination methods significant time savings can be achieved, ranging from 43% up to 67.5%, while the

BD-BR ranges from 0.34% up to 1.73%. Therefore, it is possible to achieve some degree of complexity scalability by enabling/disabling certain methods. Note, however, that using the CTU ET method is only advised when the target complexity reduction is superior to 30%, because for smaller time savings the statistical-based methods perform better, as presented in the previous table. However, studying the decision threshold effect on the CTU-level Random Forests is likely to yield better results, and this is one of the investigations planned as future work.

Figures 7.4 and 7.5 respectively show the rate-distortion plots of the best combinations of statistical-based methods with and without the help of CTU early termination. The sequence in this case is BQTerrace and the bitrate factors used were 0.2, 0.4, 0.6, and 0.8.

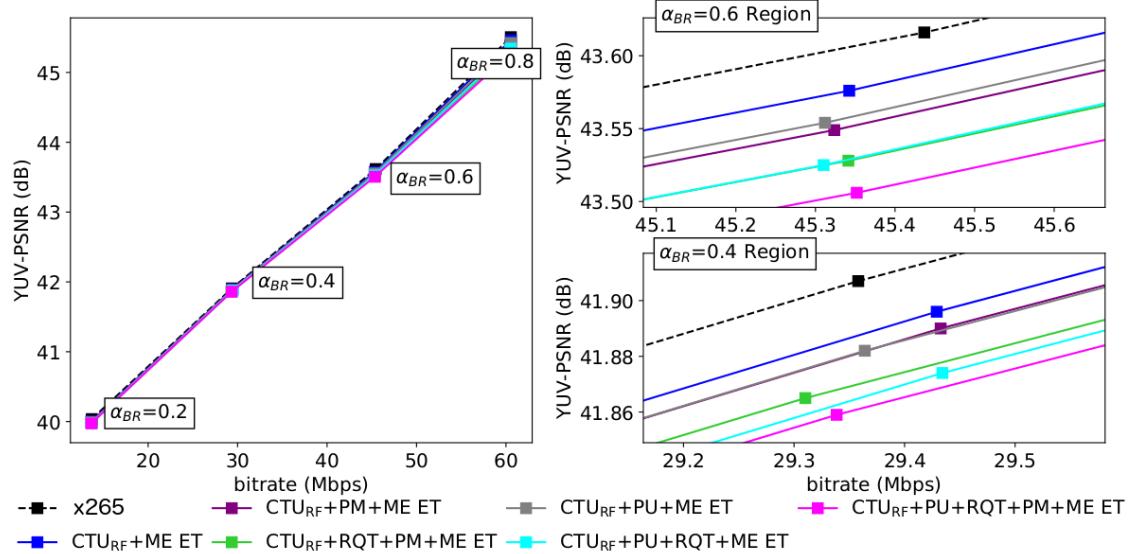
Figure 7.4: Rate-distortion curves of the proposed statiscal-based ET methods (sequence: BQTerrace, resolution: 1920x1080@60fps,  $\alpha_{BR} = 0.2, 0.4, 0.6, 0.8$ )



Note that the logarithmic behavior usually observed in rate-distortion plots under VBR conditions becomes linear when we use ABR conditions with equally-spaced bitrate factors. An interesting remark in Figure 7.4 is that the reference RD curve is below the curves representing the fast transcoding methods when  $\alpha_{BR}$  is equal 0.6. It means that, for this sequence, compression efficiency was improved. As already explained, the MV mapping strategy is the main responsible for this, but the PM ET method also has a small contribution (see Table 7.10). The gap between the reference and the fast methods becomes smaller when  $\alpha_{BR}$  changes to 0.4, indicating that the performance of the proposed methods reduces when the distance between the reference and target bitrate is greater.

In Figure 7.5, we can see that the reference encoder is the most efficient in terms of

Figure 7.5: Rate-distortion curves of the proposed statiscal-based ET methods combined with the  $\text{CTU}_{RF}$  ET method (sequence: BQTerrace, resolution: 1920x1080@60fps,  $\alpha_{BR} = 0.2, 0.4, 0.6, 0.8$ )



compression, as its curve is above all others, but this is an expected trade-off for time savings above 40%. Once again we observe a slight reduction in compression performance when the bitrate factor is equal to 0.4.

Finally, we compare once again our results with those obtained in (VAN et al., 2016) and (YANG; ZHONG, 2017), showing that now we can achieve similar time savings with better rate-distortion efficiency.

Table 7.13: Comparison of the combined fast transcoding methods with related work (sorted by average time savings)

Method	BD-BR	TS	BDTS <sub>Ratio</sub>
(YANG; ZHONG, 2017)	2.26	55.01	4.10
<b>Proposed (CTU<sub>RF</sub>+TU+PM+ME ET)</b>	<b>1.21</b>	<b>61.25</b>	<b>1.97</b>
(VAN et al., 2016)	1.89	63.68	2.96
<b>Proposed (CTU<sub>RF</sub>+PU+TU+PM+ME)</b>	<b>1.73</b>	<b>67.54</b>	<b>2.56</b>



## 8 CONCLUSIONS AND FUTURE DIRECTIONS

The research that was developed to accomplish this thesis resulted in many contributions for fast encoding and transcoding decisions using the HEVC standard. Extensive simulations were carried out in each step of this study, followed by thorough analyses that led to insightful discoveries related to: (i) the complexity of the partitioning decisions and ME searches in HEVC; (ii) the usefulness of the encoding information and of a pre-encoded bitstream in predicting the outcome of these decisions; (iii) the efficiency of learning-based methods to reduce the encoding complexity and provide complexity scaling mechanisms; and (iv) the efficiency of learning- and statistical-based methods to reduce the transrating complexity. This chapter concludes this thesis, pointing the main contributions and discussing future directions for this research.

### 8.1 Main Findings

In the assessment of the HEVC encoding tools that was presented in Chapter 4, it was discovered that significant time is spent evaluating modes that are not used to encode the output bitstream, showing the necessity of fast decision methods that try to predict which modes are irrelevant in each case. The detailed profiling of the RQT and ME searches revealed that the RQT and IME searches are the main important indicators of scene complexity. When encoding two sequences of the same resolution, more computation is spent on the IME search and on the RQT processing if one of them is more complex, whereas the FME takes practically the same time.

The analysis presented in Chapter 5 proved that some encoding variables are highly correlated with the partitioning decisions and can therefore be used to make accurate predictions on the outcome of these processes. A statistical analysis of the encoding decisions uncovered that usually a single mode stands out among the others, and this mode is usually the one that brings higher bitrate savings. For instance, the encoder chooses 2Nx2N PUs more often and larger TU blocks as well. It was also observed that the ME searches are harder to predict, which can be linked to how this problem was framed or to the lack of more relevant features.

The transcoding statistics revealed that a high-quality bitstream can be very useful to predict the behavior of subsequent encodings with lower quality. A tendency of selecting larger CU, PU and TU blocks was also observed in this analysis, and this increases

gradually as the target bitrate is reduced. It was also confirmed that the motion vector produced in the High-Quality reference is a valuable information for the IME search in LQ encodings, leading to gains in coding performance on most of the cases.

The results presented in Chapter 6 have shown that learning-based methods perform very well, reducing the complexity of HEVC encoders and transcoders with tolerable losses in compression. It was discovered that Decision Trees outperform SVMs, even when significant training refinements are employed to train the latter. The proposed encoding early termination methods achieved efficient results in most cases. The CU, PU and RQT Early Termination (ET) results stood out compared with the ME and FME ET methods, which was expected after the analysis of Chapter 5. It was also proven that complexity scalability can be easily achieved via decision threshold tuning. With this approach, the time savings ranged from 37% up to 78%, with BD-BR increments between 0.04% and 4.8%.

Finally, the transcoding results have shown that it is possible to combine statistical- and learning-based approaches to maximize the complexity reduction with small losses in BD-bitrate. It was highlighted that the CTU-level Random Forests can predict the Upper and Lower Bounds of the CTU quadtree computations, reducing the encoding time significantly and outperforming a naive statistical-based approach for the same purpose. The methods to speed up other encoding decisions performed relatively well when employed separately, but the time savings were not very significant. However, combining them with the proposed CTU-level ET led to average time savings ranging from 43% up to 67% compared with a reference transcoder (full decode + full re-encode), whereas the BD-BR increment ranged from 0.34% up to 1.7%.

Comparisons with state-of-the-art references demonstrated that the proposed methods outperform competing solutions, ascertaining the efficacy of our solutions.

In light of all this empirical evidence, we conclude that this thesis was successful in proving its claim.

## 8.2 Future Directions

A lot of ground was covered throughout this research, but there are many points that can be considered for future research. Improvement of the methods that speed the Motion Estimation searches stands out among them, because this will significantly improve the effectiveness of a solution that addresses the most time-consuming components

of HEVC encoders. The study of novel features that are more meaningful to the Motion Estimation decisions will be considered, as well as alternative ways of modeling these problems to facilitate the predictability of the outcomes.

Extending the transcoding methodology to address more applications is also an important line of work. For instance, it would be interesting to provide adaptations for spatial scalability, in which the dependent (LQ) bitstreams are encoded with a resolution that is lower than that of the reference (HQ) bitstream. This would increase the applicability of our methods, because this is a common approach in adaptive streaming systems.

Developing learning-based solutions for heterogeneous transcoding is a promising topic as well. There is a large amount of content encoded in previous standards, and re-encoding them with HEVC is advantageous for streaming servers due to its increased efficiency. Therefore, fast transcoding methods can be of great assistance in this process.

Finally, improving the methodology to simulate a real distributed video coding environment using DASH-like protocols will increase the scope and the importance of this research, bringing it closer to what is actually implemented in current systems.



## REFERENCES

- ABADI, M. et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. **arXiv preprint arXiv:1603.04467**, 2016.
- AMAZON. **Amazon Prime Video**. 2018. Available at: <https://www.primevideo.com>. Accessed: 2018-jan.
- BITMOVIN. **MPEG-DASH (Dynamic Adaptive Streaming over HTTP, ISO/IEC 23009-1)**. 2018. Available at: <http://scikit-learn.org/stable/modules/tree.html>. Accessed: 2018-jan.
- BJONTEGAARD, G. Calculation of average PSNR differences between RD-curves. In: **ITU-T Q. 6/SG16 VCEG, 15th Meeting, Austin, Texas, USA, April, 2001**. [S.l.: s.n.], 2001.
- BOSSEN, F. **Common Test Conditions and Software Reference Configurations**. [S.l.], 2012.
- BOSSEN, F.; FLYNN, D.; SÜHRING, K. **HM Software Manual, Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO**. [S.l.], 2013.
- BREIMAN, L. Random forests. **Machine Learning**, v. 45, n. 1, p. 5–32, Oct 2001. ISSN 1573-0565. Disponível em: <<https://doi.org/10.1023/A:1010933404324>>.
- CASSA, M. B.; NACCARI, M.; PEREIRA, F. Fast rate distortion optimization for the emerging HEVC standard. In: **IEEE. Picture Coding Symposium (PCS), 2012**. [S.l.], 2012. p. 493–496.
- CHANG, C.-C.; LIN, C.-J. LIBSVM: A library for support vector machines. **ACM Trans. Intell. Syst. Technol.**, ACM, New York, NY, USA, v. 2, n. 3, p. 27:1–27:27, maio 2011. ISSN 2157-6904.
- CHEN, Y.-W.; LIN, C.-J. Combining SVMs with various feature selection strategies. In: GUYON, I. et al. (Ed.). **Feature Extraction: Foundations and Applications**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 315–324. ISBN 978-3-540-35488-8.
- CHOI, K.; PARK, S.-H.; JANG, E. S. Coding tree pruning based CU early termination. **JCTVC-F092, Torino, Italy**, p. 14–22, 2011.
- CHOLLET, F. et al. **Keras: Deep Learning for Humans**. [S.l.]: GitHub, 2015. Available at: <https://github.com/keras-team/keras>.
- CISCO, V. N. I. Forecast and methodology, 2015-2020. **White Paper, Cisco**, 2016.
- CORRÊA, G. et al. Adaptive coding tree for complexity control of high efficiency video encoders. In: **IEEE. Picture Coding Symposium (PCS), 2012**. [S.l.], 2012. p. 425–428.
- CORRÊA, G. et al. Fast HEVC encoding decisions using data mining. **IEEE transactions on circuits and systems for video technology**, IEEE, v. 25, n. 4, p. 660–673, 2015.

CORRÊA, G. et al. Pareto-based method for high efficiency video coding with limited encoding time. **IEEE Transactions on Circuits and Systems for Video Technology**, IEEE, v. 26, n. 9, p. 1734–1745, 2016.

CORRÊA, G. et al. Aplicações de aprendizado de máquina na codificação de vídeo hevc. In: CIUFFO, L. N.; ROESLER, V. (Ed.). **O Futuro da Vídeo Colaboração: Perspectivas**. [S.l.]: Sociedade Brasileira de Computação, 2017.

DASH, M.; LIU, H. Feature Selection for Classification. **Intelligent Data Analysis**, IOS Press, Amsterdam, The Netherlands, The Netherlands, v. 1, n. 3, p. 131–156, maio 1997. ISSN 1088-467X.

DENG, X.; XU, M.; LI, C. Hierarchical complexity control of hevc for live video encoding. **IEEE Access**, v. 4, p. 7014–7027, 2016.

DÍAZ-HONRUBIA, A. J. et al. Adaptive fast quadtree level decision algorithm for h.264 to hevc video transcoding. **IEEE Transactions on Circuits and Systems for Video Technology**, IEEE, v. 26, n. 1, p. 154–168, 2016.

FFMPEG. **FFMpeg tool**. 2017. Available at: <http://ffmpeg.org/>. Accessed: 2017-Sep.

FREUND, Y.; SCHAPIRE, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. In: SPRINGER. **European conference on computational learning theory**. [S.l.], 1995. p. 23–37.

FRIEDMAN, J.; HASTIE, T.; TIBSHIRANI, R. **The elements of statistical learning**. [S.l.]: Springer series in statistics New York, 2001. v. 1.

GRELLOTT, M. et al. Complexity control of HEVC encoders targeting real-time constraints. **Journal of Real-Time Image Processing**, Springer, v. 13, n. 1, p. 5–24, 2017.

GROIS, D. et al. Performance comparison of h.265/mpeg-hevc, VP9, and h.264/mpeg-avc encoders. In: IEEE. **Picture Coding Symposium (PCS), 2013**. [S.l.], 2013. p. 394–397.

HUIJBERS, R.; ÖZÇELEBI, T.; BRIL, R. J. Complexity scalable motion estimation control for h.264/avc. In: IEEE. **Consumer Electronics (ICCE), 2011 IEEE International Conference on**. [S.l.], 2011. p. 49–50.

ISO/IEC. ISO/IEC DIS 23001-6, MPEG systems technologies - Dynamic adaptive streaming over HTTP (DASH). 2014.

ISO/IEC. ISO/IEC DIS 23001-6, MPEG systems technologies - Dynamic adaptive streaming over HTTP (DASH) - Part 1: Media presentation description and segment formats. 2014.

ITU-T. Recommendation H.264: Advanced Video Coding. 2003.

ITU-T. Recommendation H.265: High Efficiency Video Coding. 2013.

- JIANG, W.; MA, H.; CHEN, Y. Gradient based fast mode decision algorithm for intra prediction in HEVC. In: **IEEE. Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on.** [S.I.], 2012. p. 1836–1840.
- JOO, J.; CHOI, Y.; LEE, K. Fast sample adaptive offset encoding algorithm for HEVC based on intra prediction mode. In: **IEEE. Consumer Electronics; Berlin (ICCE-Berlin), 2013. ICCEBerlin 2013. IEEE Third International Conference on.** [S.I.], 2013. p. 50–53.
- JOUPPI, N. Google supercharges machine learning tasks with TPU custom chip. See <https://cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-tasks-with-custom-chip.html>, 2016.
- KARCZEWCZ, M.; ALSHINA, E. Jvet ahg report: Tool evaluation (ahg1). **Joint Video Exploration Team (JVET) of ITU-T SG**, 2016.
- KIM, H.-S.; PARK, R.-H. Fast CU partitioning algorithm for hevc using an online-learning-based bayesian decision rule. **IEEE transactions on circuits and systems for video technology**, IEEE, v. 26, n. 1, p. 130–138, 2016.
- KIM, I.-K. et al. HM10: High efficiency video coding (HEVC) test model 10 encoder description. **Proceedings of the Joint Collaborative Team Video Coding**, 2013.
- KIM, J. et al. Fast intra mode decision of HEVC based on hierarchical structure. In: **IEEE. Information, Communications and Signal Processing (ICICS) 2011 8th International Conference on.** [S.I.], 2011. p. 1–4.
- KIM, J. et al. Early determination of mode decision for HEVC. In: **IEEE. Picture Coding Symposium (PCS), 2012.** [S.I.], 2012. p. 449–452.
- LENG, J. et al. Content based hierarchical fast coding unit decision algorithm for HEVC. In: **IEEE. Multimedia and Signal Processing (CMSP), 2011 International Conference on.** [S.I.], 2011. v. 1, p. 56–59.
- LI, B. et al.  $\lambda$  domain rate control algorithm for high efficiency video coding. **IEEE Transactions on Image Processing**, v. 23, n. 9, p. 3841–3854, Sept 2014. ISSN 1057-7149.
- LI, B. et al. QP refinement according to Lagrange multiplier for High Efficiency Video Coding. In: **2013 IEEE International Symposium on Circuits and Systems (ISCAS2013).** [S.I.: s.n.], 2013. p. 477–480. ISSN 0271-4302.
- LORENA, A. C.; CARVALHO, A. C. de. Uma introdução às support vector machines. **Revista de Informática Teórica e Aplicada**, v. 14, n. 2, p. 43–67, 2007.
- MIRTAR, A.; DEY, S.; RAGHUNATHAN, A. Adaptation of video encoding to address dynamic thermal management effects. In: **IEEE. Green Computing Conference (IGCC), 2012 International.** [S.I.], 2012. p. 1–10.
- MIYAZAWA, K. et al. Complexity reduction of in-loop filtering for compressed image restoration in HEVC. In: **IEEE. Picture Coding Symposium (PCS), 2012.** [S.I.], 2012. p. 413–416.

MOMCILOVIC, S. et al. Run-time machine learning for HEVC/H.265 fast partitioning decision. In: **2015 IEEE Int. Symposium on Multimedia (ISM)**. [S.l.: s.n.], 2015. p. 347–350.

MPEG. Call for Evidence on Transcoding for Network Distributed Video Coding. In: **119th MPEG Meeting, Torino, Italy, July, 2017**. [S.l.: s.n.], 2017.

MUKHERJEE, D. et al. The latest open-source video codec vp9—an overview and preliminary results. In: IEEE. **Picture Coding Symposium (PCS), 2013**. [S.l.], 2013. p. 390–393.

MULTICOREWARE. **x265 - An Open-Source HEVC Encoder**. 2017. Available at: <http://x265.org/>. Accessed: 2017-Sep.

NETFLIX. **Netflix**. 2018. Available at: <https://www.netflix.com>. Accessed: 2018-jan.

PAN, Z. et al. Early termination for TZSearch in HEVC motion estimation. In: IEEE. **Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on**. [S.l.], 2013. p. 1389–1393.

PANTOS, R. **HTTP Live Streaming**. [S.l.], 2017.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, n. Oct, p. 2825–2830, 2011.

PEIXOTO, E.; IZQUIERDO, E. A complexity-scalable transcoder from h. 264/avc to the new hevc codec. In: IEEE. **Image Processing (ICIP), 2012 19th IEEE International Conference on**. [S.l.], 2012. p. 737–740.

QUINLAN, J. R. Induction of decision trees. **Machine learning**, Springer, v. 1, n. 1, p. 81–106, 1986.

QUINLAN, R. **Rulequest research data mining tools**. 2006. Availbale at: <http://www.rulequest.com>. Accessed: jan-2018.

QUINLAN, R. **Data Mining Tools See5 and C5.0**. 2015. Availbale at: <http://www.rulequest.com>. Accessed: jan-2018.

RUSSELL, S. J. et al. **Artificial intelligence: a modern approach**. [S.l.]: Prentice hall Upper Saddle River, 2003. v. 2.

SCHROEDER, D. et al. Efficient multi-rate video encoding for HEVC-based adaptive HTTP streaming. **IEEE Transactions on Circuits and Systems for Video Technology**, IEEE, 2016.

SCHWARZ, H.; MARPE, D.; WIEGAND, T. Overview of the scalable video coding extension of the h.264/avc standard. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 17, n. 9, p. 1103–1120, Sept 2007. ISSN 1051-8215.

SCIKIT. **Decision Trees**. 2018. Available at: <https://bitmovin.com/dynamic-adaptive-streaming-http-mpeg-dash/>. Accessed: 2018-jan.

SHEN, L.; ZHANG, Z.; AN, P. Fast CU size decision and mode decision algorithm for HEVC intra coding. **IEEE Transactions on Consumer Electronics**, IEEE, v. 59, n. 1, p. 207–213, 2013.

SHEN, X.; YU, L. CU splitting early termination based on weighted SVM. **EURASIP Journal on Image and Video Processing**, Springer International Publishing, v. 2013, n. 1, p. 4, 2013.

SHEN, X.; YU, L. CU splitting early termination based on weighted SVM. **EURASIP Journal on Image and Video Processing**, v. 2013, n. 1, p. 4, 2013. ISSN 1687-5281.

SHI, Y. et al. Local saliency detection based fast mode decision for HEVC intra coding. In: **IEEE. Multimedia Signal Processing (MMSP), 2013 IEEE 15th International Workshop on**. [S.l.], 2013. p. 429–433.

TAN, Y. H. et al. Complexity scalable rate-distortion optimization for h. 264/avc. In: **IEEE. Image Processing (ICIP), 2009 16th IEEE International Conference on**. [S.l.], 2009. p. 3397–3400.

TEAM, R. C. **R: A language and environment for statistical computing**. Vienna, Austria: R Foundation for Statistical Computing; 2014. 2014.

TENG, S.-W.; HANG, H.-M.; CHEN, Y.-F. Fast mode decision algorithm for residual quadtree coding in HEVC. In: **IEEE. Visual Communications and Image Processing (VCIP), 2011 IEEE**. [S.l.], 2011. p. 1–4.

VAN, L. P. et al. Fast transrating for high efficiency video coding based on machine learning. In: **IEEE. Image Processing (ICIP), 2013 20th IEEE International Conference on**. [S.l.], 2013. p. 1573–1577.

VAN, L. P. et al. Performance analysis of machine learning for arbitrary downsizing of pre-encoded HEVC video. **IEEE Transactions on Consumer Electronics**, v. 61, n. 4, p. 507–515, November 2015. ISSN 0098-3063.

VAN, L. P. et al. Efficient bit rate transcoding for high efficiency video coding. **IEEE Transactions on Multimedia**, IEEE, v. 18, n. 3, p. 364–378, 2016.

VANNE, J. et al. Comparative rate-distortion-complexity analysis of HEVC and AVC video codecs. **IEEE Transactions on Circuits and Systems for Video Technology**, IEEE, v. 22, n. 12, p. 1885–1898, 2012.

VETRO, A.; CHRISTOPOULOS, C.; SUN, H. Video transcoding architectures and techniques: an overview. **IEEE Signal Processing Magazine**, v. 20, n. 2, p. 18–29, Mar 2003. ISSN 1053-5888.

WALLENDAEL, G. V.; COCK, J. D.; WALLE, R. V. de. Fast transcoding for video delivery by means of a control stream. In: **2012 19th IEEE International Conference on Image Processing**. [S.l.: s.n.], 2012. p. 733–736. ISSN 1522-4880.

WANG, C. et al. Single-input-multiple-output transcoding for video streaming. In: **IEEE. Multimedia Signal Processing (MMSP), 2016 IEEE 18th International Workshop on**. [S.l.], 2016. p. 1–5.

WANG, S.; ZHU, E.; YIN, J. Video anomaly detection based on ULGP-OF descriptor and one-class ELM. In: **IEEE. Neural Networks (IJCNN), 2016 International Joint Conference on.** [S.l.], 2016. p. 2630–2637.

WITTEN, I. H. et al. **Data Mining: Practical machine learning tools and techniques.** [S.l.]: Morgan Kaufmann, 2016.

WU, S. et al. Motion sketch based crowd video retrieval via motion structure coding. In: **IEEE. Image Processing (ICIP), 2016 IEEE International Conference on.** [S.l.], 2016. p. 1205–1209.

XIONG, J. et al. A fast HEVC inter CU selection method based on pyramid motion divergence. **IEEE transactions on multimedia**, IEEE, v. 16, n. 2, p. 559–564, 2014.

XU, M. et al. Learning to detect video saliency with HEVC features. **IEEE Transactions on Image Processing**, IEEE, v. 26, n. 1, p. 369–385, 2017.

YAN, S. et al. Group-based fast mode decision algorithm for intra prediction in HEVC. In: **IEEE. Signal Image Technology and Internet Based Systems (SITIS), 2012 Eighth International Conference on.** [S.l.], 2012. p. 225–229.

YANG, S. H.; ZHONG, C. C. Fast Coding-Unit Mode Decision for HEVC Transrating. In: **2017 IEEE International Conference on Computer and Information Technology (CIT).** [S.l.: s.n.], 2017. p. 93–100.

ZHANG, Y. et al. Machine learning-based coding unit depth decisions for flexible complexity allocation in high efficiency video coding. **IEEE Transactions on Image Processing**, IEEE, v. 24, n. 7, p. 2225–2238, 2015.

ZHANG, Y. et al. Machine learning-based coding unit depth decisions for flexible complexity allocation in High Efficiency Video Coding. **IEEE Transactions on Image Processing**, v. 24, n. 7, p. 2225–2238, July 2015. ISSN 1057-7149.

ZHAO, L. et al. Fast mode decision algorithm for intra prediction in HEVC. In: **IEEE. Visual Communications and Image Processing (VCIP), 2011 IEEE.** [S.l.], 2011. p. 1–4.

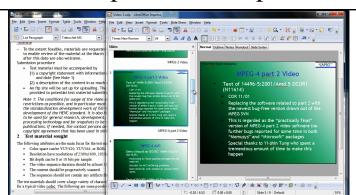
ZHU, L. et al. Binary and multi-class learning based low complexity optimization for HEVC encoding. **IEEE Transactions on Broadcasting**, PP, n. 99, p. 1–15, 2017. ISSN 0018-9316.

## APPENDIX A — INPUT SEQUENCES

Picture & Specifications	Scene Description
 Traffic, 2560x1600, 30 fps, 8 bits/sample	<p>Shows the traffic on three lanes. The first two lanes are slightly jammed, so the cars move slowly. The third one is free, so cars drive by faster.</p>
 PeopleOnStreet, 2560x1600, 30 fps, 8 bits/sample	<p>Displays a large crowd walking in many directions. The camera remains still.</p>
 NebutaFestival, 2560x1600, 60 fps, 10 bits/sample	<p>Shows a float of the Aomori Nebuta Festival. The camera pans very slightly in many directions, keeping the focus on the face of the warrior.</p>
 SteamLocomotive, 2560x1600, 60 fps, 10 bits/sample	<p>Depicts a locomotive moving towards the camera. The locomotive produces lots of smoke. At a later instance, the locomotive is so close to the camera that the surroundings can no longer be seen.</p>
 Kimono, 1920x1080, 24 fps, 8 bits/sample	<p>A lady in a kimono walks around the woods. In frame 140, a change of scene happens, showing a Japanese house.</p>

Picture & Specifications	Scene Description
 ParkScene, 1920x1080, 24 fps, 8 bits/sample	<p>Depicts a park in which cyclers ride by. The camera pans to the left slowly.</p>
 Cactus, 1920x1080, 50 fps, 8 bits/sample	<p>Shows many objects, including a pot with a cactus that rotates vertically. In addition, something similar to a weather vane with cards and an object with two tigers attached to it both rotate horizontally. The camera remains still.</p>
 BasketballDrive, 1920x1080, 50 fps, 8 bits/sample	<p>A group of players exercise a basketball match. Every player moves constantly. The camera pans to the right and to the left, always following the players.</p>
 BQTerrace, 1920x1080, 60 fps, 8 bits/sample	<p>The terrace above the square depicted in BQSquare is shown, but the camera moves up, revealing a road on which cars move in both directions. The camera continues to move up until the terrace is barely seen.</p>
 BasketballDrill, 832x480.50 fps, 8 bits/sample	<p>A group of players run in circles while taking turns to throw the ball against the board. While the players are constantly moving, the camera remains still.</p>
 BQMall, 832x480.60 fps, 8 bits/sample	<p>Depicts a typical day at the mall, with people coming and going from both directions. The camera steadily pans to the left. In the last seconds, fewer people are in the scene, so most of the activity comes from camera panning.</p>

Picture & Specifications	Scene Description
 PartyScene, 832x480, 50 fps, 8 bits/sample	<p>Shows a party with kids enjoying themselves. Two kids circle around the tree on the left, while the one in the middle blows bubbles. The camera slowly zooms in on the girl in the middle. Bubbles fly around.</p>
 RaceHorsesC, 832x480, 30 fps, 8 bits/sample	<p>Depicts a group of horse riders moving around a grassy background. The camera follows their movement.</p>
 BasketballPass, 416x240, 50 fps, 8 bits/sample	<p>Another scene from the Basketball game, in which both the players and the camera move constantly.</p>
 BQSquare, 416x240, 60 fps, 8 bits/sample	<p>Shows a square with a restaurant, in which people spend leisure time. Initially only the square can be seen, but a body of water becomes visible on the background as the camera zooms out.</p>
 BlowingBubbles, 416x240, 50 fps, 8 bits/sample	<p>Two girls play with a bubble-blow. Many bubbles fly around, and the camera steadily zooms out through the entire sequence.</p>
 RaceHorses, 416x240, 30 fps, 8 bits/sample	<p>Same scene from RaceHorsesC with a smaller resolution.</p>

Picture & Specifications	Scene Description
 Kristen&Sara, 1280x720, 60 fps, 8 bits/sample	<p>Two girls discuss in what seems to be an interview. The background is still. They turn their heads and laugh during the conversation. There is also a moment when the girl on the left points at the banner in the back.</p>
 Johnny, 1280x720, 60 fps, 8 bits/sample	<p>A man presents the news with the same still background from Kristen&amp;Sara behind him. He moves and gesticulates less compared to the girls in the related sequence.</p>
 ChinaSpeed, 1024x768.30 fps, 8 bits/sample	<p>A car racing game in which the background changes rapidly to provide the idea of fast movement. The foreground, containing the user interface, stays practically unchanged.</p>
 BasketballDrillText, 832x480, 50 fps, 8 bits/sample	<p>The same scene from BasketballDrill plus a caption box on the bottom, in which a sliding text is displayed</p>
 SlideEditing, 1280x720, 30 fps, 8 bits/sample	<p>Shows a computer screen while a user edits slides and text. Initially, the user rolls up/down the slides panel, then starts editing a slide, moving to the text editor afterwards. Aside from the parts being edited by the user, the remaining background remains still.</p>

Picture & Specifications	Scene Description
 <p data-bbox="244 476 595 570">SlideShow, 1280x720, 20 fps, 8 bits/sample</p>	<p data-bbox="625 390 1346 467">Shows a Microsoft PowerPoint presentation, with many inter-/intra-slide animations.</p>



## APPENDIX B — FEATURES AND METRICS

### B.1 List of Features used in the Fast Encoding Decisions

Feature	Description	Values
$\Delta QP$	Delta QP of the current frame.	1, 2, 3, 4
PredMode	Best prediction mode	SKIP, Merge, Inter, Intra
PU <sub>Part</sub>	Best PU partition	2Nx2N, 2NxN, Nx2N, NxN, 2NxN <sub>U</sub> , 2NxN <sub>D</sub> , nLx2N, nRx2N
Bits	Number of encoded bits	Continuous
Distortion	Total distortion	Continuous
Cost <sub>Best</sub>	Total rate-distortion cost	Continuous
Entropy <sub>Bins</sub>	Total number of bins	Continuous
SAD	Absolute sum of residual data	Continuous
SSE	Sum of squared residual data	Continuous
MSE	Mean Squared Error	Continuous
Cost <sub>2Nx2N</sub>	Rate-distortion cost to encode 2Nx2N partition	Continuous
Cost <sub>2NxN</sub>	Rate-distortion cost to encode 2NxN partition	Continuous
Cost <sub>Nx2N</sub>	Rate-distortion cost to encode Nx2N partition	Continuous
Cost <sub>MSM</sub>	Rate-distortion cost to encode with Merge-Skip mode (MSM)	Continuous
Cost <sub>Parent</sub>	Rate-distortion cost of the parent CU	Continuous
CostRatio <sub>2Nx2N, MSM</sub>	RDCR between 2Nx2N and Merge/Skip mode costs (see (15))	Continuous
CostRatio <sub>BEST, 2Nx2N</sub>	RDCR between best and 2Nx2N costs (see (15))	Continuous
CostRatio <sub>BEST, MSM</sub>	RDCR between Merge/Skip modes and 2Nx2N costs (see (15))	Continuous
CostRatio <sub>BEST, Parent</sub>	RDCR between best CU and its parent (see (15))	Continuous
NormCostDiff <sub>2Nx2N, MSM</sub>	NDIFF between 2Nx2N and MSM costs (see (15))	Continuous
NormCostDiff <sub>BEST, 2Nx2N</sub>	NDIFF between best and 2Nx2N costs (see (15))	Continuous

Feature	Description	Values
NormCostDiff <sub>BEST,MSM</sub>	NDIFF between MSM and 2Nx2N costs (see (15))	Continuous
NormCostDiff <sub>BEST,Parent</sub>	NDIFF between best CU and its parent (see (15))	Continuous
RQT <sub>Depth</sub>	RQT depth of the best mode	0,1,2
CBF	Coded Block Flag	0,1
SkipFlag	SKIP mode flag	0,1
MergeFlag	Merge mode flag	0,1
NonZero <sub>Coeffs</sub>	Number of non-zero coefficients from quantized data	Continuous
SideInfo <sub>Bits</sub>	Percentage of bits used as side information in the CU (proposed in (SHEN; YU, 2013b))	Continuous
RefFrame <sub>IDX</sub>	Reference index from Motion Estimation	-1, 0, 1
MV <sub>int</sub>	Magnitude of the integer Motion Vector	Continuous
MV <sub>frac</sub>	Magnitude of the fractional Motion Vector	Continuous
MVpred <sub>int</sub>	Magnitude of the predicted integer Motion Vector	Continuous
MVpred <sub>frac</sub>	Magnitude of the predicted fractional Motion Vector	Continuous
\Delta MV <sub>int</sub>	Magnitude of the integer Differential Motion Vector	Continuous
\Delta MV <sub>frac</sub>	Magnitude of the fractional Differential Motion Vector	Continuous
MVP <sub>Idx</sub>	Motion Vector Prediction Index	-1, 0, 1
Inter <sub>DIR</sub>	Inter-prediction direction	None, L0, L1, Bipredicted
FME <sub>Flag</sub>	Fractional Motion Estimation mode	No FME, Half-pel FME, Quarter-pel FME
Depth <sub>Coloc</sub>	Average depth of the temporal Co-located CUs	continuous
SplitFlag <sub>Coloc</sub>	Temporal Co-located CU split count from L0 and L1 lists	0, 1, 2
Depth <sub>Left</sub>	Depth of the left neighboring CU	0,1,2,3
SplitFlag <sub>Left</sub>	Split flag of the left neighboring CU	0,1
Depth <sub>Up</sub>	Depth of the upper neighboring CU	0,1,2,3

Feature	Description	Values
$\text{SplitFlag}_{\text{Up}}$	Split flag of the upper neighboring CU	0,1
$\text{Depth}_{\text{UpLeft}}$	Depth of the upper-left neighboring CU	0,1,2,3
$\text{Split}_{\text{UpLeft}}$	Split flag of the upper-left neighboring CU	0,1
$\text{Depth}_{\text{UpRight}}$	Depth of the upper-right neighboring CU	0,1,2,3
$\text{SplitFlag}_{\text{UpRight}}$	Split flag of the upper-right neighboring CU	0,1
$\text{CtxDepth}_{\text{CU}}$	Average depth of the temporal Co-located CUs plus the neighboring ones	continuous
$\text{SplitFlag}_{\text{Ctx}}$	Co-located CUs plus neighboring CUs split count	0, 1, 2, 3, 4, 5, 6
$\text{CtxDepth}_{\text{CTU}}$	Average depth of neighboring CTUs	Continuous
$\Delta \text{CtxDepth}_{\text{CU}}$	Delta Average depth of the temporal Co-located CUs plus the neighboring ones	continuous
$\Delta \text{CtxDepth}_{\text{CTU}}$	Delta Average depth of neighboring CTUs	Continuous

## B.2 Scores of the Features used in the Fast Encoding Decisions

### Information Gain Ratio for the CU partitioning prediction

IG	64x64	32x32	16x16	Average
Bits	0.50	0.41	0.42	0.44
SkipFlag	0.49	0.41	0.41	0.44
$\text{NormCostDiff}_{\text{BEST,MSM}}$	0.39	0.35	0.33	0.36
$\text{CtxDepth}_{\text{CU}}$	0.39	0.36	0.33	0.36
$\text{CtxDepth}_{\text{CTU}}$	0.37	0.30	0.32	0.33
$\text{CostRatio}_{\text{BEST,MSM}}$	0.35	0.30	0.28	0.31
CBF	0.43	0.25	0.23	0.31
$\text{RQT}_{\text{Depth}}$	0.32	0.28	0.25	0.28
$\text{Pred}_{\text{Mode}}$	0.34	0.24	0.23	0.27
$\text{CostRatio}_{\text{2Nx2N,MSM}}$	0.25	0.22	0.33	0.27
$\text{SideInfo}_{\text{Bits}}$	0.35	0.23	0.21	0.27

IG	64x64	32x32	16x16	Average
SplitFlag <sub>Left</sub>	0.28	0.26	0.22	0.25
SplitFlag <sub>Up</sub>	0.27	0.26	0.22	0.25
SplitFlag <sub>UpRight</sub>	0.27	0.26	0.22	0.25
NormCostDiff <sub>2Nx2N,MSM</sub>	0.10	0.20	0.31	0.20
Split <sub>UpLeft</sub>	0.20	0.19	0.18	0.19
SplitFlag <sub>Ctx</sub>	0.17	0.18	0.19	0.18
SplitFlag <sub>Coloc</sub>	0.15	0.18	0.17	0.17
Depth <sub>Left</sub>	0.19	0.14	0.11	0.14
Depth <sub>Up</sub>	0.19	0.14	0.10	0.14
Depth <sub>UpRight</sub>	0.19	0.14	0.10	0.14
PU <sub>Part</sub>	0.15	0.14	0.14	0.14
NormCostDiff <sub>BEST,2Nx2N</sub>	0.05	0.12	0.25	0.14
CostRatio <sub>BEST,2Nx2N</sub>	0.05	0.12	0.25	0.14
Depth <sub>Coloc</sub>	0.11	0.14	0.14	0.13
$ \Delta MV_{int} $	0.15	0.10	0.10	0.12
Depth <sub>UpLeft</sub>	0.14	0.11	0.09	0.11
MVP <sub>Idx</sub>	0.14	0.10	0.10	0.11
NonZero <sub>Coeffs</sub>	0.13	0.06	0.06	0.08
Distortion	0.07	0.08	0.07	0.07
$ \Delta MV_{frac} $	0.09	0.06	0.06	0.07
SSE	0.05	0.09	0.07	0.07
MSE	0.05	0.09	0.07	0.07
Cost <sub>MSM</sub>	0.06	0.07	0.08	0.07
Entropy <sub>Bins</sub>	0.09	0.04	0.06	0.07
SAD	0.05	0.09	0.06	0.07
Cost <sub>Best</sub>	0.06	0.06	0.07	0.06
Cost <sub>2Nx2N</sub>	0.05	0.07	0.04	0.05
Cost <sub>2NxN</sub>	0.05	0.07	0.03	0.05
MergeFlag	0.00	0.04	0.09	0.04
$\Delta QP$	0.03	0.04	0.05	0.04
Cost <sub>2Nx2N</sub>	0.05	0.04	0.03	0.04
Inter <sub>DIR</sub>	0.02	0.02	0.04	0.03
Cost <sub>Parent</sub>	0.00	0.02	0.05	0.03
$ MV_{pred,int} $	0.04	0.01	0.00	0.02
$ MV_{int} $	0.04	0.01	0.00	0.02
RefFrame <sub>IDX</sub>	0.02	0.01	0.02	0.01
CostRatio <sub>BEST,Parent</sub>	0.00	0.01	0.02	0.01
FME <sub>Flag</sub>	0.02	0.01	0.00	0.01
$ MV_{frac} $	0.01	0.00	0.00	0.01
$ MV_{pred,frac} $	0.01	0.00	0.00	0.00

IG	64x64	32x32	16x16	Average
NormCostDiff <sub>BEST,Parent</sub>	0.00	0.01	0.00	0.00

### F-Score for the CU partitioning prediction

FSC	64x64	32x32	16x16	Average
SkipFlag	1.38	0.94	0.88	1.06
Pred <sub>Mode</sub>	1.53	0.71	0.60	0.94
SplitFlag <sub>Ctx</sub>	1.06	0.91	0.67	0.88
CtxDepth <sub>CTU</sub>	0.80	0.84	0.84	0.83
CtxDepth <sub>CU</sub>	0.71	0.82	0.70	0.74
CBF	1.09	0.48	0.43	0.67
Depth <sub>Coloc</sub>	0.41	0.77	0.77	0.65
RQT <sub>Depth</sub>	0.83	0.50	0.44	0.59
SplitFlag <sub>Coloc</sub>	0.40	0.52	0.38	0.43
SplitFlag <sub>Left</sub>	0.54	0.39	0.25	0.40
SplitFlag <sub>Up</sub>	0.53	0.38	0.25	0.39
SplitFlag <sub>UpRight</sub>	0.53	0.38	0.25	0.39
Depth <sub>Left</sub>	0.42	0.37	0.22	0.34
Depth <sub>Up</sub>	0.42	0.36	0.21	0.33
Depth <sub>UpRight</sub>	0.42	0.36	0.21	0.33
Depth <sub>UpLeft</sub>	0.28	0.31	0.27	0.28
PU <sub>Part</sub>	0.30	0.28	0.23	0.27
SideInfo <sub>Bits</sub>	0.41	0.16	0.21	0.26
Split <sub>UpLeft</sub>	0.34	0.25	0.17	0.25
CostRatio <sub>BEST,2Nx2N</sub>	0.07	0.16	0.36	0.20
NormCostDiff <sub>BEST,2Nx2N</sub>	0.07	0.16	0.36	0.20
Bits	0.22	0.08	0.21	0.17
MVP <sub>Idx</sub>	0.22	0.13	0.11	0.16
$ \Delta MV_{frac} $	0.17	0.10	0.08	0.12
$\Delta QP$	0.08	0.10	0.14	0.11
Inter <sub>DIR</sub>	0.01	0.03	0.08	0.04
NonZero <sub>Coeffs</sub>	0.07	0.01	0.02	0.03
MergeFlag	0.00	0.03	0.07	0.03
$ \text{MV}_{frac} $	0.04	0.01	0.00	0.02
Entropy <sub>Bins</sub>	0.01	0.01	0.03	0.02
FME <sub>Flag</sub>	0.04	0.01	0.00	0.02
$ \text{MVpred}_{frac} $	0.04	0.01	0.00	0.02
$ \Delta MV_{int} $	0.02	0.01	0.01	0.01
Cost <sub>2Nx2N</sub>	0.00	0.00	0.02	0.01
SAD	0.01	0.01	0.01	0.01

FSC	64x64	32x32	16x16	Average
Cost <sub>2NxN</sub>	0.00	0.00	0.02	0.01
Cost <sub>MSM</sub>	0.01	0.01	0.01	0.01
Cost <sub>2Nx2N</sub>	0.00	0.00	0.02	0.01
MV <sub>pred<sub>int</sub></sub>	0.01	0.00	0.00	0.01
RefFrame <sub>IDX</sub>	0.01	0.00	0.01	0.00
MV <sub>int</sub>	0.01	0.00	0.00	0.00
CostRatio <sub>2Nx2N,MSM</sub>	0.00	0.00	0.00	0.00
NormCostDiff <sub>2Nx2N,MSM</sub>	0.00	0.00	0.00	0.00
Cost <sub>Best</sub>	0.00	0.00	0.00	0.00
Distortion	0.00	0.00	0.00	0.00
MSE	0.00	0.00	0.00	0.00
SSE	0.00	0.00	0.00	0.00
CostRatio <sub>BEST,Parent</sub>	0.00	0.00	0.00	0.00
NormCostDiff <sub>BEST,Parent</sub>	0.00	0.00	0.00	0.00
CostRatio <sub>BEST,MSM</sub>	0.00	0.00	0.00	0.00
NormCostDiff <sub>BEST,MSM</sub>	0.00	0.00	0.00	0.00
Cost <sub>Parent</sub>	0.00	0.00	0.00	0.00

### Information Gain Ratio for the PU partitioning prediction

IG	64x64	32x32	16x16	8x8	Average
Bits	0.35	0.37	0.42	0.46	0.40
CostRatio <sub>2Nx2N,MSM</sub>	0.26	0.29	0.38	0.49	0.36
Skip <sub>Flag</sub>	0.35	0.32	0.34	0.37	0.35
NormCostDiff <sub>2Nx2N,MSM</sub>	0.08	0.21	0.35	0.49	0.28
CtxDepth <sub>CTU</sub>	0.30	0.28	0.28	0.27	0.28
Pred <sub>Mode</sub>	0.27	0.25	0.27	0.29	0.27
CBF	0.28	0.25	0.26	0.27	0.26
SideInfo <sub>Bits</sub>	0.23	0.20	0.18	0.28	0.22
CtxDepth <sub>CU</sub>	0.29	0.20	0.14	0.12	0.19
RQT <sub>Depth</sub>	0.21	0.17	0.16	0.16	0.17
NormCostDiff <sub>BEST,MSM</sub>	0.16	0.14	0.12	0.13	0.14
CostRatio <sub>BEST,MSM</sub>	0.16	0.14	0.12	0.13	0.14
ΔMV <sub>int</sub>	0.15	0.13	0.12	0.14	0.13
MVP <sub>Idx</sub>	0.15	0.13	0.12	0.14	0.13
SSE	0.11	0.09	0.12	0.16	0.12
SAD	0.08	0.09	0.12	0.15	0.11
SplitFlag <sub>Up</sub>	0.18	0.14	0.11	0.00	0.11
SplitFlag <sub>Left</sub>	0.18	0.13	0.10	0.00	0.10
Distortion	0.11	0.11	0.08	0.09	0.10

Cost <sub>Best</sub>	0.09	0.08	0.10	0.12	0.10
Cost <sub>MSM</sub>	0.07	0.08	0.10	0.12	0.09
NonZero <sub>Coeffs</sub>	0.14	0.10	0.07	0.05	0.09
$ \Delta MV_{frac} $	0.09	0.08	0.08	0.09	0.08
Depth <sub>Coloc</sub>	0.09	0.09	0.08	0.07	0.08
Split <sub>UpLeft</sub>	0.14	0.11	0.09	0.00	0.08
SplitFlag <sub>Coloc</sub>	0.15	0.09	0.09	0.00	0.08
Merge <sub>Flag</sub>	0.04	0.07	0.09	0.11	0.07
SplitFlag <sub>Ctx</sub>	0.13	0.08	0.08	0.00	0.07
Cost <sub>2Nx2N</sub>	0.06	0.08	0.04	0.11	0.07
$ MV_{pred_{int}} $	0.09	0.08	0.06	0.04	0.07
Depth <sub>Up</sub>	0.12	0.08	0.04	0.01	0.06
Depth <sub>UpRight</sub>	0.12	0.08	0.04	0.01	0.06
$ MV_{int} $	0.09	0.07	0.06	0.03	0.06
Depth <sub>Left</sub>	0.12	0.08	0.04	0.01	0.06
Depth <sub>UpLeft</sub>	0.10	0.07	0.05	0.02	0.06
Cost <sub>Parent</sub>	0.00	0.04	0.07	0.09	0.05
CostRatio <sub>BEST,Parent</sub>	0.00	0.02	0.03	0.08	0.03
FME <sub>Flag</sub>	0.05	0.04	0.03	0.01	0.03
$\Delta QP$	0.02	0.02	0.02	0.03	0.02
Inter <sub>DIR</sub>	0.00	0.00	0.01	0.06	0.02
$ MV_{frac} $	0.02	0.02	0.01	0.00	0.01
$ MV_{pred_{frac}} $	0.02	0.02	0.01	0.00	0.01
RefFrame <sub>IDX</sub>	0.02	0.01	0.01	0.01	0.01
NormCostDiff <sub>BEST,Parent</sub>	0.00	0.01	0.01	0.03	0.01
PU <sub>Part</sub>	0.00	0.00	0.00	0.00	0.00
Cost <sub>2NxN</sub>	0.00	0.00	0.00	0.00	0.00
Cost <sub>2Nx2N</sub>	0.00	0.00	0.00	0.00	0.00
Entropy <sub>Bins</sub>	0.00	0.00	0.00	0.00	0.00

### F-Score for the PU partitioning prediction

FSC	64x64	32x32	16x16	8x8	Average
Skip <sub>Flag</sub>	0.80	0.70	0.78	0.86	0.79
Pred <sub>Mode</sub>	0.76	0.63	0.69	0.74	0.70
CostRatio <sub>BEST,2Nx2N</sub>	0.20	0.40	0.73	1.28	0.65
CBF	0.55	0.47	0.50	0.51	0.51
CtxDepth <sub>CTU</sub>	0.51	0.49	0.46	0.43	0.47
Depth <sub>Coloc</sub>	0.35	0.39	0.32	0.26	0.33
CtxDepth <sub>CU</sub>	0.42	0.39	0.25	0.14	0.30
SplitFlag <sub>Ctx</sub>	0.65	0.30	0.17	0.00	0.28

RQT <sub>Depth</sub>	0.47	0.19	0.14	0.09	0.22
SideInfo <sub>Bits</sub>	0.22	0.20	0.20	0.23	0.21
MVP <sub>Idx</sub>	0.25	0.20	0.15	0.15	0.19
SplitFlag <sub>Coloc</sub>	0.40	0.20	0.14	0.00	0.19
Bits	0.10	0.10	0.14	0.29	0.16
Depth <sub>UpLeft</sub>	0.18	0.18	0.11	0.04	0.13
Depth <sub>Up</sub>	0.24	0.18	0.08	0.02	0.13
Depth <sub>UpRight</sub>	0.24	0.18	0.08	0.02	0.13
SplitFlag <sub>Left</sub>	0.31	0.14	0.06	0.00	0.13
SplitFlag <sub>Up</sub>	0.30	0.14	0.06	0.00	0.13
SplitFlag <sub>UpRight</sub>	0.30	0.14	0.06	0.00	0.13
Depth <sub>Left</sub>	0.24	0.18	0.07	0.02	0.13
$ \Delta MV_{frac} $	0.17	0.14	0.10	0.10	0.13
Split <sub>UpLeft</sub>	0.22	0.11	0.04	0.00	0.09
SAD	0.07	0.05	0.06	0.07	0.06
$\Delta QP$	0.05	0.05	0.05	0.09	0.06
$ MV_{frac} $	0.08	0.06	0.05	0.02	0.05
$ MVpred_{frac} $	0.08	0.06	0.05	0.02	0.05
FME <sub>Flag</sub>	0.08	0.06	0.04	0.01	0.05
NonZero <sub>Coeffs</sub>	0.07	0.03	0.03	0.02	0.04
Merge <sub>Flag</sub>	0.02	0.03	0.04	0.05	0.03
Cost <sub>MSM</sub>	0.03	0.03	0.03	0.03	0.03
Cost <sub>2Nx2N</sub>	0.02	0.00	0.01	0.08	0.03
Cost <sub>Best</sub>	0.03	0.03	0.03	0.03	0.03
Inter <sub>DIR</sub>	0.00	0.00	0.01	0.09	0.03
RefFrame <sub>IDX</sub>	0.02	0.01	0.01	0.00	0.01
Cost <sub>Parent</sub>	0.00	0.01	0.01	0.01	0.01
SSE	0.02	0.01	0.01	0.00	0.01
MSE	0.02	0.01	0.01	0.00	0.01
$ MVpred_{int} $	0.01	0.01	0.00	0.02	0.01
$ MV_{int} $	0.01	0.01	0.00	0.01	0.01
Distortion	0.01	0.01	0.01	0.00	0.01
CostRatio <sub>2Nx2N,MSM</sub>	0.01	0.01	0.01	0.01	0.01
NormCostDiff <sub>2Nx2N,MSM</sub>	0.01	0.01	0.01	0.01	0.01
$ \Delta MV_{int} $	0.00	0.00	0.00	0.02	0.01
CostRatio <sub>BEST,Parent</sub>	0.00	0.00	0.00	0.00	0.00
NormCostDiff <sub>BEST,Parent</sub>	0.00	0.00	0.00	0.00	0.00
CostRatio <sub>BEST,MSM</sub>	0.00	0.00	0.00	0.00	0.00
NormCostDiff <sub>BEST,MSM</sub>	0.00	0.00	0.00	0.00	0.00
PU <sub>Part</sub>	0.00	0.00	0.00	0.00	0.00
Cost <sub>2NxN</sub>	0.00	0.00	0.00	0.00	0.00

Cost <sub>2Nx2N</sub>	0.00	0.00	0.00	0.00	0.00
EntropyBins	0.00	0.00	0.00	0.00	0.00

### Information Gain Ratio for the RQT partitioning prediction

	IG	64x64	32x32	16x16	8x8	Average
CBF	0.87	0.60	0.62	0.68	0.69	
Bits	0.80	0.58	0.62	0.68	0.67	
SkipFlag	0.70	0.56	0.61	0.68	0.64	
PredMode	0.70	0.53	0.58	0.65	0.62	
RQT <sub>Depth</sub>	0.60	0.45	0.43	0.45	0.48	
SideInfo <sub>Bits</sub>	0.62	0.44	0.42	0.37	0.46	
CostRatio <sub>2Nx2N,MSM</sub>	0.29	0.27	0.42	0.54	0.38	
CtxDepth <sub>CTU</sub>	0.39	0.35	0.34	0.31	0.34	
CtxDepth <sub>CU</sub>	0.40	0.40	0.34	0.19	0.33	
NormCostDiff <sub>2Nx2N,MSM</sub>	0.10	0.22	0.37	0.53	0.31	
SplitFlag <sub>Up</sub>	0.29	0.27	0.20	0.00	0.19	
SplitFlag <sub>Left</sub>	0.29	0.26	0.19	0.00	0.19	
Split <sub>UpLeft</sub>	0.23	0.21	0.17	0.00	0.15	
SSE	0.18	0.10	0.13	0.16	0.14	
ΔMV <sub>int</sub>	0.17	0.13	0.13	0.13	0.14	
SAD	0.15	0.10	0.12	0.16	0.13	
CostRatio <sub>BEST,MSM</sub>	0.16	0.14	0.13	0.09	0.13	
MVP <sub>Idx</sub>	0.15	0.12	0.12	0.12	0.13	
NonZero <sub>Coeffs</sub>	0.27	0.09	0.07	0.07	0.13	
SplitFlag <sub>Ctx</sub>	0.16	0.17	0.16	0.00	0.12	
Depth <sub>Coloc</sub>	0.09	0.15	0.14	0.10	0.12	
Depth <sub>Up</sub>	0.20	0.14	0.10	0.02	0.11	
Depth <sub>UpRight</sub>	0.20	0.14	0.10	0.02	0.11	
Depth <sub>Left</sub>	0.19	0.14	0.09	0.02	0.11	
SplitFlag <sub>Coloc</sub>	0.11	0.17	0.16	0.00	0.11	
Cost <sub>MSM</sub>	0.10	0.07	0.10	0.13	0.10	
Depth <sub>UpLeft</sub>	0.16	0.12	0.10	0.03	0.10	
Cost <sub>Best</sub>	0.09	0.07	0.10	0.13	0.10	
Distortion	0.09	0.08	0.10	0.11	0.09	
ΔMV <sub>frac</sub>	0.09	0.07	0.08	0.08	0.08	
Cost <sub>2Nx2N</sub>	0.08	0.06	0.05	0.13	0.08	
MergeFlag	0.15	0.09	0.05	0.03	0.08	
ΔQP	0.06	0.06	0.07	0.08	0.07	
MVpred <sub>int</sub>	0.10	0.04	0.04	0.02	0.05	
MV <sub>int</sub>	0.10	0.03	0.04	0.02	0.05	

Cost <sub>Parent</sub>	0.00	0.02	0.06	0.10	0.05
CostRatio <sub>BEST,Parent</sub>	0.00	0.03	0.02	0.09	0.04
FME <sub>Flag</sub>	0.05	0.02	0.02	0.01	0.03
Inter <sub>DIR</sub>	0.01	0.01	0.02	0.03	0.02
NormCostDiff <sub>BEST,Parent</sub>	0.00	0.01	0.01	0.05	0.02
MV <sub>frac</sub>	0.02	0.01	0.01	0.01	0.01
MVpred <sub>frac</sub>	0.02	0.01	0.01	0.01	0.01
RefFrame <sub>IDX</sub>	0.02	0.00	0.00	0.01	0.01
PU <sub>Part</sub>	0.00	0.00	0.00	0.00	0.00
Cost <sub>2NxN</sub>	0.00	0.00	0.00	0.00	0.00
Cost <sub>2Nx2N</sub>	0.00	0.00	0.00	0.00	0.00
Entropy <sub>Bins</sub>	0.00	0.00	0.00	0.00	0.00

### F-Score for the RQT partitioning prediction

	64x64	32x32	16x16	8x8	Average
FSC					
CBF	12.40	1.89	2.17	2.99	4.86
Pred <sub>Mode</sub>	7.67	1.80	2.14	2.98	3.65
Skip <sub>Flag</sub>	3.18	1.52	1.94	2.77	2.35
RQT <sub>Depth</sub>	2.86	1.49	1.19	0.80	1.58
CtxDepth <sub>CTU</sub>	0.74	0.99	0.98	0.70	0.85
ΔCtxDepth <sub>CTU</sub>	0.74	0.99	0.98	0.70	0.85
CtxDepth <sub>CU</sub>	0.74	0.99	0.75	0.29	0.69
ΔCtxDepth <sub>CU</sub>	0.74	0.99	0.75	0.29	0.69
CostRatio <sub>BEST,2Nx2N</sub>	0.20	0.35	0.71	1.30	0.64
NormCostDiff <sub>BEST,2Nx2N</sub>	0.20	0.35	0.71	1.30	0.64
SplitFlag <sub>Ctx</sub>	1.01	0.98	0.49	0.00	0.62
SideInfo <sub>Bits</sub>	1.10	0.36	0.44	0.43	0.58
Depth <sub>Coloc</sub>	0.31	0.83	0.75	0.42	0.58
Bits	0.41	0.17	0.31	0.59	0.37
SplitFlag <sub>Up</sub>	0.59	0.45	0.18	0.00	0.31
SplitFlag <sub>UpRight</sub>	0.59	0.45	0.18	0.00	0.31
SplitFlag <sub>Left</sub>	0.58	0.44	0.17	0.00	0.30
Depth <sub>Up</sub>	0.44	0.41	0.21	0.04	0.28
Depth <sub>UpRight</sub>	0.44	0.41	0.21	0.04	0.28
SplitFlag <sub>Coloc</sub>	0.27	0.51	0.32	0.00	0.27
Depth <sub>Left</sub>	0.42	0.41	0.21	0.04	0.27
Depth <sub>UpLeft</sub>	0.32	0.35	0.28	0.08	0.26
Split <sub>UpLeft</sub>	0.42	0.30	0.13	0.00	0.21
ΔQP	0.17	0.19	0.23	0.24	0.21
MVP <sub>Idx</sub>	0.25	0.19	0.17	0.11	0.18

$ \Delta MV_{frac} $	0.21	0.15	0.13	0.09	0.14
NonZeroCoeffs	0.20	0.00	0.00	0.03	0.06
SAD	0.16	0.00	0.00	0.03	0.05
$ MV_{frac} $	0.09	0.04	0.04	0.02	0.05
Cost <sub>2Nx2N</sub>	0.00	0.00	0.04	0.13	0.04
FMEFlag	0.08	0.03	0.03	0.01	0.04
$ MVpred_{frac} $	0.08	0.03	0.03	0.01	0.04
$ MVpred_{int} $	0.07	0.01	0.01	0.02	0.03
$ MV_{int} $	0.06	0.01	0.01	0.01	0.02
$ \Delta MV_{int} $	0.03	0.02	0.02	0.02	0.02
MergeFlag	0.05	0.02	0.01	0.00	0.02
Inter <sub>DIR</sub>	0.00	0.01	0.02	0.04	0.02
MSE	0.04	0.00	0.00	0.00	0.01
SSE	0.04	0.00	0.00	0.00	0.01
RefFrame <sub>IDX</sub>	0.03	0.00	0.00	0.01	0.01
Distortion	0.01	0.01	0.01	0.01	0.01
CostRatio <sub>2Nx2N,MSM</sub>	0.01	0.00	0.01	0.01	0.01
NormCostDiff <sub>2Nx2N,MSM</sub>	0.01	0.00	0.01	0.01	0.01
Cost <sub>MSM</sub>	0.01	0.00	0.00	0.00	0.00
Cost <sub>Best</sub>	0.00	0.00	0.00	0.00	0.00
NormCostDiff <sub>BEST,MSM</sub>	0.00	0.00	0.00	0.00	0.00
CostRatio <sub>BEST,MSM</sub>	0.00	0.00	0.00	0.00	0.00
Cost <sub>Parent</sub>	0.00	0.00	0.00	0.00	0.00
CostRatio <sub>BEST,Parent</sub>	0.00	0.00	0.00	0.00	0.00
NormCostDiff <sub>BEST,Parent</sub>	0.00	0.00	0.00	0.00	0.00
PU <sub>Part</sub>	0.00	0.00	0.00	0.00	0.00
Cost <sub>2NxN</sub>	0.00	0.00	0.00	0.00	0.00
Cost <sub>2Nx2N</sub>	0.00	0.00	0.00	0.00	0.00
EntropyBins	0.00	0.00	0.00	0.00	0.00

### Information Gain Ratio for the ME Early Termination

IG	64x64	32x32	16x16	8x8	Average
RefFrame <sub>IDX</sub>	0.34	0.40	0.43	0.46	0.41
Inter <sub>DIR</sub>	0.10	0.11	0.13	0.14	0.12
SkipFlag	0.05	0.07	0.09	0.11	0.08
Bits	0.05	0.07	0.09	0.11	0.08
CBF	0.04	0.07	0.09	0.11	0.08
Pred <sub>Mode</sub>	0.04	0.06	0.08	0.10	0.07
SAD	0.07	0.07	0.06	0.06	0.07
SSE	0.07	0.07	0.06	0.06	0.07

CostRatio <sub>BEST,2Nx2N</sub>	0.05	0.06	0.06	0.08	0.07
NormCostDiff <sub>BEST,2Nx2N</sub>	0.05	0.06	0.06	0.08	0.07
CostRatio <sub>2Nx2N,MSM</sub>	0.05	0.06	0.06	0.08	0.07
SideInfo <sub>Bits</sub>	0.06	0.07	0.07	0.05	0.06
NormCostDiff <sub>2Nx2N,MSM</sub>	0.03	0.05	0.06	0.07	0.06
$\Delta\text{CtxDepth}_{\text{CTU}}$	0.05	0.05	0.05	0.04	0.05
$ \Delta\text{MV}_{int} $	0.02	0.03	0.04	0.06	0.04
$ \text{MVpred}_{int} $	0.04	0.04	0.04	0.03	0.04
Cost <sub>2Nx2N</sub>	0.01	0.00	0.05	0.09	0.04
$ \text{MV}_{int} $	0.04	0.04	0.04	0.03	0.04
NormCostDiff <sub>BEST,MSM</sub>	0.03	0.03	0.03	0.03	0.03
CostRatio <sub>BEST,MSM</sub>	0.03	0.03	0.03	0.03	0.03
MVP <sub>Idx</sub>	0.02	0.03	0.03	0.04	0.03
NonZero <sub>Coeffs</sub>	0.03	0.04	0.03	0.02	0.03
$\Delta\text{QP}$	0.03	0.03	0.03	0.03	0.03
CtxDepth <sub>CU</sub>	0.05	0.02	0.01	0.01	0.02
$\Delta\text{CtxDepth}_{\text{CU}}$	0.05	0.02	0.01	0.01	0.02
$ \Delta\text{MV}_{frac} $	0.01	0.02	0.03	0.03	0.02
MergeFlag	0.02	0.01	0.01	0.03	0.02
RQT <sub>Depth</sub>	0.03	0.00	0.01	0.02	0.02
Depth <sub>Coloc</sub>	0.01	0.01	0.01	0.01	0.01
Cost <sub>MSM</sub>	0.01	0.01	0.01	0.01	0.01
Cost <sub>Best</sub>	0.01	0.01	0.01	0.01	0.01
SplitFlag <sub>Up</sub>	0.02	0.01	0.01	0.00	0.01
SplitFlag <sub>UpRight</sub>	0.02	0.01	0.01	0.00	0.01
SplitFlag <sub>Left</sub>	0.02	0.00	0.01	0.00	0.01
Split <sub>UpLeft</sub>	0.02	0.00	0.01	0.00	0.01
Depth <sub>UpLeft</sub>	0.01	0.01	0.00	0.00	0.01
CostRatio <sub>BEST,Parent</sub>	0.00	0.01	0.01	0.01	0.01
Depth <sub>Up</sub>	0.01	0.01	0.00	0.00	0.01
Depth <sub>UpRight</sub>	0.01	0.01	0.00	0.00	0.01
Cost <sub>Parent</sub>	0.00	0.01	0.01	0.01	0.01
SplitFlag <sub>Coloc</sub>	0.02	0.00	0.00	0.00	0.01
Depth <sub>Left</sub>	0.01	0.01	0.00	0.00	0.01
SplitFlag <sub>Ctx</sub>	0.02	0.00	0.00	0.00	0.01
NormCostDiff <sub>BEST,Parent</sub>	0.00	0.01	0.00	0.00	0.00
Distortion	0.00	0.00	0.00	0.00	0.00
$ \text{MVpred}_{frac} $	0.00	0.00	0.00	0.00	0.00
$ \text{MV}_{frac} $	0.00	0.00	0.00	0.00	0.00
FME <sub>Flag</sub>	0.00	0.00	0.00	0.00	0.00
PU <sub>Part</sub>	0.00	0.00	0.00	0.00	0.00

Cost <sub>2NxN</sub>	0.00	0.00	0.00	0.00	0.00
Cost <sub>2Nx2N</sub>	0.00	0.00	0.00	0.00	0.00
EntropyBins	0.00	0.00	0.00	0.00	0.00

### F-Score for the ME Early Termination

	64x64	32x32	16x16	8x8	Average
FSC					
SkipFlag	0.08	0.11	0.13	0.14	0.11
PredMode	0.07	0.10	0.13	0.13	0.11
RefFrame <sub>IDX</sub>	0.12	0.12	0.10	0.08	0.11
CBF	0.05	0.09	0.12	0.12	0.10
CostRatio <sub>BEST,2Nx2N</sub>	0.02	0.07	0.11	0.14	0.09
NormCostDiff <sub>BEST,2Nx2N</sub>	0.02	0.07	0.11	0.14	0.09
SideInfoBits	0.08	0.10	0.10	0.07	0.09
Inter <sub>DIR</sub>	0.06	0.07	0.08	0.11	0.08
SAD	0.08	0.08	0.06	0.05	0.07
Bits	0.04	0.05	0.07	0.09	0.06
ΔQP	0.04	0.04	0.05	0.05	0.04
ΔCtxDepth <sub>CTU</sub>	0.02	0.03	0.04	0.04	0.03
CtxDepth <sub>CTU</sub>	0.02	0.03	0.04	0.04	0.03
MVP <sub>Idx</sub>	0.03	0.03	0.02	0.02	0.02
MVpred <sub>int</sub>	0.03	0.03	0.03	0.02	0.02
ΔMV <sub>frac</sub>	0.03	0.03	0.03	0.02	0.02
NonZero <sub>Coeffs</sub>	0.02	0.03	0.03	0.01	0.02
MV <sub>int</sub>	0.02	0.02	0.02	0.01	0.02
Cost <sub>2Nx2N</sub>	0.00	0.00	0.02	0.05	0.02
Depth <sub>Coloc</sub>	0.01	0.02	0.02	0.02	0.02
SplitFlag <sub>Ctx</sub>	0.05	0.00	0.01	0.00	0.01
CtxDepth <sub>CU</sub>	0.02	0.02	0.01	0.01	0.01
ΔCtxDepth <sub>CU</sub>	0.02	0.02	0.01	0.01	0.01
ΔMV <sub>int</sub>	0.02	0.02	0.01	0.01	0.01
MSE	0.02	0.01	0.01	0.00	0.01
SSE	0.02	0.01	0.01	0.00	0.01
SplitFlag <sub>Coloc</sub>	0.04	0.00	0.00	0.00	0.01
Depth <sub>UpLeft</sub>	0.01	0.02	0.01	0.00	0.01
RQT <sub>Depth</sub>	0.03	0.00	0.00	0.00	0.01
SplitFlag <sub>Up</sub>	0.03	0.00	0.00	0.00	0.01
SplitFlag <sub>UpRight</sub>	0.03	0.00	0.00	0.00	0.01
SplitFlag <sub>Left</sub>	0.02	0.00	0.00	0.00	0.01
Depth <sub>Up</sub>	0.01	0.01	0.00	0.00	0.01
Depth <sub>UpRight</sub>	0.01	0.01	0.00	0.00	0.01

SplitUpLeft	0.02	0.00	0.00	0.00	0.01
DepthLeft	0.01	0.01	0.00	0.00	0.01
MergeFlag	0.01	0.00	0.00	0.00	0.01
Cost <sub>MSM</sub>	0.01	0.01	0.00	0.00	0.01
Cost <sub>Best</sub>	0.01	0.01	0.00	0.00	0.00
FMEFlag	0.00	0.00	0.00	0.01	0.00
Cost <sub>Parent</sub>	0.00	0.00	0.00	0.00	0.00
MV <sub>frac</sub>	0.00	0.00	0.00	0.00	0.00
MVpred <sub>frac</sub>	0.00	0.00	0.00	0.00	0.00
Distortion	0.00	0.00	0.00	0.00	0.00
CostRatio <sub>BEST,MSM</sub>	0.00	0.00	0.00	0.00	0.00
NormCostDiff <sub>BEST,MSM</sub>	0.00	0.00	0.00	0.00	0.00
CostRatio <sub>2Nx2N,MSM</sub>	0.00	0.00	0.00	0.00	0.00
NormCostDiff <sub>2Nx2N,MSM</sub>	0.00	0.00	0.00	0.00	0.00
CostRatio <sub>BEST,Parent</sub>	0.00	0.00	0.00	0.00	0.00
NormCostDiff <sub>BEST,Parent</sub>	0.00	0.00	0.00	0.00	0.00
PU <sub>Part</sub>	0.00	0.00	0.00	0.00	0.00
Cost <sub>2NxN</sub>	0.00	0.00	0.00	0.00	0.00
Cost <sub>2Nx2N</sub>	0.00	0.00	0.00	0.00	0.00
EntropyBins	0.00	0.00	0.00	0.00	0.00

### Information Gain Ratio for the FME<sub>Flag</sub> Early Termination

IG	64x64	32x32	16x16	8x8	Average
FMEFlag	0.43	0.48	0.52	0.56	0.50
MVpred <sub>int</sub>	0.37	0.39	0.40	0.41	0.39
MV <sub>int</sub>	0.37	0.39	0.40	0.41	0.39
MV <sub>frac</sub>	0.21	0.24	0.26	0.29	0.25
MVpred <sub>frac</sub>	0.21	0.24	0.26	0.28	0.25
CostRatio <sub>BEST,MSM</sub>	0.15	0.15	0.15	0.15	0.15
NormCostDiff <sub>BEST,MSM</sub>	0.15	0.15	0.15	0.15	0.15
Cost <sub>MSM</sub>	0.13	0.11	0.10	0.10	0.11
Cost <sub>Best</sub>	0.13	0.11	0.10	0.10	0.11
Distortion	0.13	0.11	0.10	0.09	0.11
MSE	0.13	0.11	0.10	0.08	0.10
SSE	0.13	0.11	0.10	0.08	0.10
SAD	0.13	0.11	0.10	0.08	0.10
CostRatio <sub>BEST,2Nx2N</sub>	0.07	0.08	0.08	0.15	0.09
NormCostDiff <sub>BEST,2Nx2N</sub>	0.07	0.08	0.08	0.15	0.09
CostRatio <sub>2Nx2N,MSM</sub>	0.07	0.08	0.08	0.15	0.09
NormCostDiff <sub>2Nx2N,MSM</sub>	0.07	0.08	0.08	0.15	0.09

RefFrame <sub>IDX</sub>	0.07	0.07	0.08	0.09	0.08
Inter <sub>DIR</sub>	0.07	0.07	0.08	0.09	0.08
CostRatio <sub>BEST, Parent</sub>	0.00	0.05	0.10	0.13	0.07
Cost <sub>2Nx2N</sub>	0.10	0.07	0.05	0.06	0.07
Cost <sub>Parent</sub>	0.00	0.09	0.10	0.09	0.07
Bits	0.10	0.07	0.05	0.03	0.06
SideInfo <sub>Bits</sub>	0.10	0.07	0.05	0.03	0.06
$\Delta$ CtxDepth <sub>CTU</sub>	0.08	0.07	0.05	0.05	0.06
CtxDepth <sub>CTU</sub>	0.08	0.07	0.05	0.05	0.06
NormCostDiff <sub>BEST, Parent</sub>	0.00	0.05	0.08	0.05	0.05
$ \Delta MV_{int} $	0.04	0.02	0.07	0.05	0.04
CtxDepth <sub>CU</sub>	0.08	0.03	0.01	0.00	0.03
$\Delta$ CtxDepth <sub>CU</sub>	0.08	0.03	0.01	0.00	0.03
NonZero <sub>Coeffs</sub>	0.05	0.04	0.01	0.00	0.02
Depth <sub>Coloc</sub>	0.03	0.02	0.02	0.01	0.02
Skip <sub>Flag</sub>	0.05	0.02	0.00	0.00	0.02
MVP <sub>Idx</sub>	0.04	0.02	0.01	0.00	0.02
CBF	0.04	0.02	0.00	0.00	0.02
SplitFlag <sub>Coloc</sub>	0.05	0.01	0.01	0.00	0.02
Pred <sub>Mode</sub>	0.04	0.02	0.00	0.00	0.02
$ \Delta MV_{frac} $	0.03	0.02	0.01	0.00	0.01
SplitFlag <sub>Up</sub>	0.04	0.01	0.00	0.00	0.01
SplitFlag <sub>UpRight</sub>	0.04	0.01	0.00	0.00	0.01
RQT <sub>Depth</sub>	0.03	0.01	0.01	0.00	0.01
SplitFlag <sub>Left</sub>	0.04	0.01	0.00	0.00	0.01
Split <sub>UpLeft</sub>	0.04	0.01	0.00	0.00	0.01
SplitFlag <sub>Ctx</sub>	0.03	0.01	0.00	0.00	0.01
Depth <sub>UpLeft</sub>	0.03	0.01	0.00	0.00	0.01
Depth <sub>Up</sub>	0.03	0.01	0.00	0.00	0.01
Depth <sub>UpRight</sub>	0.03	0.01	0.00	0.00	0.01
Depth <sub>Left</sub>	0.03	0.01	0.00	0.00	0.01
Merge <sub>Flag</sub>	0.02	0.01	0.01	0.00	0.01
$\Delta QP$	0.00	0.00	0.00	0.00	0.00
PU <sub>Part</sub>	0	0	0	0	0
Cost <sub>2NxN</sub>	0	0	0	0	0
Cost <sub>2Nx2N</sub>	0	0	0	0	0
Entropy <sub>Bins</sub>	0	0	0	0	0

### F-Score for the FME<sub>Flag</sub> Early Termination

FSC	64x64	32x32	16x16	8x8	Average

FME <sub>Flag</sub>	1.56	2.07	2.48	3.06	2.29
MV <sub>pred<sub>frac</sub></sub>	0.89	1.07	1.19	1.39	1.13
MV <sub>frac</sub>	0.89	1.07	1.17	1.38	1.13
CostRatio <sub>BEST,2Nx2N</sub>	0.12	0.12	0.07	0.02	0.08
NormCostDiff <sub>BEST,2Nx2N</sub>	0.12	0.12	0.07	0.02	0.08
RefFrame <sub>IDX</sub>	0.06	0.06	0.06	0.07	0.06
Depth <sub>Coloc</sub>	0.07	0.05	0.04	0.03	0.05
ΔCtxDepth <sub>CTU</sub>	0.07	0.05	0.03	0.02	0.05
CtxDepth <sub>CTU</sub>	0.07	0.05	0.03	0.02	0.05
SAD	0.07	0.05	0.03	0.01	0.04
CostRatio <sub>BEST,MSM</sub>	0.03	0.04	0.04	0.05	0.04
NormCostDiff <sub>BEST,MSM</sub>	0.03	0.04	0.04	0.05	0.04
SplitFlag <sub>Ctx</sub>	0.12	0.02	0.00	0.00	0.03
Distortion	0.04	0.04	0.03	0.02	0.03
SplitFlag <sub>Coloc</sub>	0.11	0.02	0.00	0.00	0.03
Cost <sub>Best</sub>	0.05	0.04	0.03	0.02	0.03
Cost <sub>MSM</sub>	0.04	0.03	0.03	0.02	0.03
Cost <sub>Parent</sub>	0.00	0.04	0.04	0.03	0.03
SkipFlag	0.07	0.03	0.00	0.00	0.03
CtxDepth <sub>CU</sub>	0.06	0.03	0.01	0.00	0.02
ΔCtxDepth <sub>CU</sub>	0.06	0.03	0.01	0.00	0.02
Pred <sub>Mode</sub>	0.06	0.03	0.00	0.00	0.02
MVP <sub>Idx</sub>	0.06	0.02	0.00	0.00	0.02
CBF	0.05	0.02	0.00	0.00	0.02
MV <sub>int</sub>	0.02	0.02	0.02	0.02	0.02
Cost <sub>2Nx2N</sub>	0.04	0.02	0.01	0.00	0.02
SplitFlag <sub>Up</sub>	0.06	0.01	0.00	0.00	0.02
SplitFlag <sub>UpRight</sub>	0.06	0.01	0.00	0.00	0.02
SplitFlag <sub>Left</sub>	0.06	0.01	0.00	0.00	0.02
Split <sub>UpLeft</sub>	0.05	0.01	0.00	0.00	0.01
RQT <sub>Depth</sub>	0.04	0.01	0.00	0.00	0.01
NonZero <sub>Coeffs</sub>	0.03	0.02	0.00	0.00	0.01
Depth <sub>UpLeft</sub>	0.04	0.02	0.00	0.00	0.01
Depth <sub>Left</sub>	0.04	0.01	0.00	0.00	0.01
Depth <sub>Up</sub>	0.04	0.01	0.00	0.00	0.01
Depth <sub>UpRight</sub>	0.04	0.01	0.00	0.00	0.01
CostRatio <sub>2Nx2N,MSM</sub>	0.01	0.01	0.01	0.01	0.01
NormCostDiff <sub>2Nx2N,MSM</sub>	0.01	0.01	0.01	0.01	0.01
ΔMV <sub>frac</sub>	0.03	0.01	0.00	0.00	0.01
MV <sub>pred<sub>int</sub></sub>	0.01	0.01	0.01	0.02	0.01
Bits	0.02	0.01	0.00	0.00	0.01

$ \Delta MV_{int} $	0.01	0.01	0.01	0.00	0.00
MSE	0.01	0.00	0.00	0.00	0.00
SSE	0.01	0.00	0.00	0.00	0.00
SideInfoBits	0.01	0.00	0.00	0.00	0.00
InterDIR	0.01	0.00	0.00	0.00	0.00
MergeFlag	0.01	0.00	0.00	0.00	0.00
$\Delta QP$	0.00	0.00	0.00	0.00	0.00
CostRatio <sub>BEST,Parent</sub>	0.00	0.00	0.00	0.00	0.00
NormCostDiff <sub>BEST,Parent</sub>	0.00	0.00	0.00	0.00	0.00
PU <sub>Part</sub>	0.00	0.00	0.00	0.00	0.00
Cost <sub>2NxN</sub>	0.00	0.00	0.00	0.00	0.00
Cost <sub>2Nx2N</sub>	0.00	0.00	0.00	0.00	0.00
EntropyBins	0.00	0.00	0.00	0.00	0.00

### B.3 List of Features used in the Fast Transcoding Decisions

Feature	Description	Values
$\alpha_{BR}$	Quotient between target bitrate and HQ bitrate	0.1 - 0.9
cu size	size of the CU in one dimension	8,16,32,64
pred mode	prediction mode used to encode the CU	SKIP, inter, intra
is skip	Skip flag	0, 1
is intra	Intra prediction flag	0, 1
merge flag	Merge flag	0, 1
pu part	PU partition mode	2Nx2N, 2NxN, Nx2N, NxN, 2NxN, 2NxN, 2NxN, nLx2N, nRx2N
rqt root cbf	CBF of the RQT root node	0, 1
cu bits	Amount of bits used in CU coding	continuous
cbf luma	Coding Block Flag of luma component	0, 1
cbf cb	Coding Block Flag of the Cb component	0, 1
cbf cr	Coding Block Flag of the Cr component	0, 1
transf size	Minimum transform size in the CU RQT in one dimension	8, 16, 32
pred	Sum of predicted block samples	continuous
coeff Q	Sum of quantized coefficients	continuous
coeff IQ	Sum of dequantized coefficients	continuous
residue	Sum of residue samples	continuous
mv mod 10	Module of MV0	continuous
mv mod 11	Module of MV1	continuous

Feature	Description	Values
cu qp	Quantization Parameter used to encode the CU	continuous

#### B.4 Scores of the Features used in the Fast Transcoding Decisions

IGR	MAX	IGR	MIN	FSC	MAX	FSC	MIN
Depth <sub>MAX</sub>	0.57	Depth <sub>MAX</sub>	0.43	Depth <sub>MAX</sub>	1.74	Depth <sub>AVG</sub>	1.08
Depth <sub>AVG</sub>	0.57	Depth <sub>MIN</sub>	0.40	Depth <sub>AVG</sub>	1.70	Depth <sub>MIN</sub>	1.02
Depth <sub>MIN</sub>	0.48	Depth <sub>AVG</sub>	0.40	Depth <sub>MIN</sub>	1.35	Depth <sub>MAX</sub>	0.85
CoeffSum <sub>MAX</sub>	0.28	Bits <sub>AVG</sub>	0.26	CBF <sub>MAX</sub>	0.54	CBF <sub>MAX</sub>	0.35
CBF <sub>MAX</sub>	0.28	SkipFlag <sub>MIN</sub>	0.23	SkipFlag <sub>MIN</sub>	0.33	SkipFlag <sub>MIN</sub>	0.28
CBF <sub>AVG</sub>	0.28	PredMode <sub>MIN</sub>	0.23	PredMode <sub>MIN</sub>	0.30	PredMode <sub>MIN</sub>	0.26
QCoeffSum <sub>MAX</sub>	0.28	SkipFlag <sub>AVG</sub>	0.23	CBF <sub>AVG</sub>	0.27	IntraFlag <sub>MAX</sub>	0.17
QCoeffSum <sub>AVG</sub>	0.28	PredMode <sub>AVG</sub>	0.23	IntraFlag <sub>MAX</sub>	0.25	PU <sub>MAX</sub>	0.16
CoeffSum <sub>AVG</sub>	0.28	CBF <sub>MAX</sub>	0.22	PU <sub>MAX</sub>	0.25	MergeFlag <sub>MAX</sub>	0.14
PredSum <sub>AVG</sub>	0.28	CBF <sub>AVG</sub>	0.22	CBF-Cr <sub>MAX</sub>	0.24	CBF-Cb <sub>MAX</sub>	0.13
PredSum <sub>MAX</sub>	0.28	QCoeffSum <sub>MAX</sub>	0.22	CBF-Cb <sub>MAX</sub>	0.24	TrSize <sub>MIN</sub>	0.13
ResidSum <sub>AVG</sub>	0.28	PredSum <sub>MAX</sub>	0.22	MergeFlag <sub>MAX</sub>	0.19	CBF-Cr <sub>MAX</sub>	0.12
ResidSum <sub>MAX</sub>	0.28	CoeffSum <sub>AVG</sub>	0.22	CBF-Y <sub>MAX</sub>	0.19	CBF <sub>AVG</sub>	0.12
QP <sub>MAX</sub>	0.24	QCoeffSum <sub>AVG</sub>	0.22	TrSize <sub>MIN</sub>	0.16	CBF-Y <sub>MAX</sub>	0.12
SkipFlag <sub>MIN</sub>	0.24	ResidSum <sub>MAX</sub>	0.22	SkipFlag <sub>AVG</sub>	0.15	SkipFlag <sub>AVG</sub>	0.08
PredMode <sub>MIN</sub>	0.24	PredSum <sub>AVG</sub>	0.22	MergeFlag <sub>MIN</sub>	0.11	MergeFlag <sub>MIN</sub>	0.07
SkipFlag <sub>AVG</sub>	0.24	ResidSum <sub>AVG</sub>	0.22	PredMode <sub>AVG</sub>	0.10	PredMode <sub>AVG</sub>	0.06
PredMode <sub>AVG</sub>	0.24	CoeffSum <sub>MAX</sub>	0.22	IntraFlag <sub>AVG</sub>	0.09	CoeffSum <sub>MAX</sub>	0.06
IntraFlag <sub>MAX</sub>	0.19	QP <sub>MAX</sub>	0.17	QP <sub>MIN</sub>	0.09	TrSize <sub>AVG</sub>	0.06
IntraFlag <sub>AVG</sub>	0.19	IntraFlag <sub>MAX</sub>	0.15	QP <sub>AVG</sub>	0.09	CBF-Y <sub>MIN</sub>	0.05
PU <sub>AVG</sub>	0.19	IntraFlag <sub>AVG</sub>	0.15	TrSize <sub>AVG</sub>	0.08	IntraFlag <sub>AVG</sub>	0.05
PU <sub>MAX</sub>	0.19	PU <sub>MAX</sub>	0.14	Bits <sub>MAX</sub>	0.08	QP <sub>MIN</sub>	0.05
MVL0  <sub>MAX</sub>	0.17	PU <sub>AVG</sub>	0.14	$\alpha_{BR}$	0.08	QP <sub>AVG</sub>	0.04
CBF-Cr <sub>MAX</sub>	0.17	MVL0  <sub>MAX</sub>	0.13	CoeffSum <sub>MAX</sub>	0.07	MVL1  <sub>MAX</sub>	0.04
CBF-Cr <sub>AVG</sub>	0.17	MVL0  <sub>AVG</sub>	0.13	CBF-Cr <sub>AVG</sub>	0.07	MVL0  <sub>MAX</sub>	0.04
TrSize <sub>MIN</sub>	0.16	TrSize <sub>MIN</sub>	0.13	MVL0  <sub>MAX</sub>	0.06	Bits <sub>MAX</sub>	0.04
Bits <sub>AVG</sub>	0.15	MergeFlag <sub>MAX</sub>	0.12	CBF-Y <sub>MIN</sub>	0.06	PU <sub>MIN</sub>	0.04
CBF-Cb <sub>MAX</sub>	0.15	MergeFlag <sub>AVG</sub>	0.12	MVL1  <sub>MAX</sub>	0.05	PredMode <sub>MAX</sub>	0.04
CBF-Cb <sub>AVG</sub>	0.15	PU <sub>MIN</sub>	0.11	PU <sub>MIN</sub>	0.05	$\alpha_{BR}$	0.04
MergeFlag <sub>MAX</sub>	0.15	CBF-Cr <sub>MAX</sub>	0.10	CBF-Cb <sub>AVG</sub>	0.05	MVL1  <sub>MIN</sub>	0.02
MergeFlag <sub>AVG</sub>	0.15	CBF-Cr <sub>AVG</sub>	0.10	CBF-Y <sub>AVG</sub>	0.05	CBF-Cr <sub>AVG</sub>	0.02
PU <sub>MIN</sub>	0.14	PredMode <sub>MAX</sub>	0.10	MVL1  <sub>MIN</sub>	0.04	CBF-Cb <sub>AVG</sub>	0.02
CBF-Y <sub>MAX</sub>	0.14	MVL1  <sub>MAX</sub>	0.09	PredMode <sub>MAX</sub>	0.03	PredSum <sub>MIN</sub>	0.02
CBF-Y <sub>AVG</sub>	0.14	CBF-Cb <sub>MAX</sub>	0.09	CBF-Cb <sub>MIN</sub>	0.02	TrSize <sub>MAX</sub>	0.02

IGR	MAX	IGR	MIN	FSC	MAX	FSC	MIN
$ MVL1 _{MAX}$	0.12	CBF-Cb <sub>Avg</sub>	0.09	TrSize <sub>MAX</sub>	0.02	CBF-Y <sub>Avg</sub>	0.02
$ MVL0 _{AVG}$	0.11	CBF-Y <sub>MAX</sub>	0.09	PredSum <sub>MIN</sub>	0.02	CBF-Cb <sub>MIN</sub>	0.02
$ MVL1 _{MIN}$	0.11	CBF-Y <sub>Avg</sub>	0.09	$ MVL0 _{MIN}$	0.02	$ MVL0 _{MIN}$	0.01
PredMode <sub>MAX</sub>	0.10	PredSum <sub>MIN</sub>	0.08	PredSum <sub>Avg</sub>	0.01	SkipFlag <sub>MAX</sub>	0.01
MergeFlag <sub>MIN</sub>	0.10	MergeFlag <sub>MIN</sub>	0.08	$ MVL1 _{AVG}$	0.01	PredSum <sub>Avg</sub>	0.01
TrSize <sub>Avg</sub>	0.10	$ MVL1 _{AVG}$	0.08	CBF <sub>MIN</sub>	0.01	CBF-Cr <sub>MIN</sub>	0.01
$ MVL1 _{AVG}$	0.09	TrSize <sub>Avg</sub>	0.07	CBF-Cr <sub>MIN</sub>	0.01	QP <sub>MAX</sub>	0.01
TrSize <sub>MAX</sub>	0.08	ResidSum <sub>MIN</sub>	0.07	SkipFlag <sub>MAX</sub>	0.01	$ MVL1 _{AVG}$	0.01
PredSum <sub>MIN</sub>	0.07	$ MVL1 _{MIN}$	0.06	PredSum <sub>MAX</sub>	0.01	IntraFlag <sub>MIN</sub>	0.01
ResidSum <sub>MIN</sub>	0.07	TrSize <sub>MAX</sub>	0.05	QP <sub>MAX</sub>	0.01	QCoeffSum <sub>MAX</sub>	0.01
IntraFlag <sub>MIN</sub>	0.06	IntraFlag <sub>MIN</sub>	0.05	IntraFlag <sub>MIN</sub>	0.01	$ MVL0 _{AVG}$	0.01
CBF-Cb <sub>MIN</sub>	0.05	CBF-Y <sub>MIN</sub>	0.04	$ MVL0 _{AVG}$	0.01	CoeffSum <sub>Avg</sub>	0.01
QP <sub>MIN</sub>	0.05	QP <sub>MIN</sub>	0.04	QCoeffSum <sub>MAX</sub>	0.01	MergeFlag <sub>Avg</sub>	0.00
QP <sub>Avg</sub>	0.05	QP <sub>Avg</sub>	0.04	MergeFlag <sub>Avg</sub>	0.01	PredSum <sub>MAX</sub>	0.00
Bits <sub>MAX</sub>	0.05	Bits <sub>MAX</sub>	0.04	CoeffSum <sub>Avg</sub>	0.00	CBF <sub>MIN</sub>	0.00
$\alpha_{BR}$	0.05	$ MVL0 _{MIN}$	0.04	ResidSum <sub>MAX</sub>	0.00	Bits <sub>MIN</sub>	0.00
$ MVL0 _{MIN}$	0.05	CBF-Cb <sub>MIN</sub>	0.04	Bits <sub>MIN</sub>	0.00	ResidSum <sub>MAX</sub>	0.00
CBF-Y <sub>MIN</sub>	0.05	Bits <sub>MIN</sub>	0.04	ResidSum <sub>MIN</sub>	0.00	CoeffSum <sub>MIN</sub>	0.00
QCoeffSum <sub>MIN</sub>	0.04	QCoeffSum <sub>MIN</sub>	0.03	CoeffSum <sub>MIN</sub>	0.00	QCoeffSum <sub>MIN</sub>	0.00
CBF-Cr <sub>MIN</sub>	0.04	CBF-Cr <sub>MIN</sub>	0.03	QCoeffSum <sub>MIN</sub>	0.00	Bits <sub>Avg</sub>	0.00
CoeffSum <sub>MIN</sub>	0.03	CoeffSum <sub>MIN</sub>	0.02	Bits <sub>Avg</sub>	0.00	QCoeffSum <sub>Avg</sub>	0.00
Bits <sub>MIN</sub>	0.03	$\alpha_{BR}$	0.02	QCoeffSum <sub>Avg</sub>	0.00	PU <sub>Avg</sub>	0.00
CBF <sub>MIN</sub>	0.01	SkipFlag <sub>MAX</sub>	0.01	PU <sub>Avg</sub>	0.00	ResidSum <sub>MIN</sub>	0.00
SkipFlag <sub>MAX</sub>	0.01	CBF <sub>MIN</sub>	0.00	ResidSum <sub>Avg</sub>	0.00	ResidSum <sub>Avg</sub>	0.00



## APPENDIX C — PAPERS SUBMITTED AND PUBLISHED DURING THE PHD

### BOOK CHAPTER

1. Guilherme Correa , **Mateus Grellert**, Sergio Bampi, Luis Cruz: *Aplicações de Aprendizado de Máquina na Codificação de Vídeo HEVC*. O Futuro da Videocolaboração: Perspectivas, Chapter 9, Publisher: SBC. (**Published**)

### PAPERS SUBMITTED TO JOURNALS

1. **Mateus Grellert**, Bruno Zatt, Sergio Bampi, Luis Cruz: *Fast Coding Unit Partition Decision for HEVC Using Support Vector Machines*. IEEE Transactions on Circuits and Systems for Video Technology, 2018. (**In Peer Review**)
2. Bianca Silveira, Guilherme Paim, Bruno Abreu, **Mateus Grellert**, Claudio Machado Diniz, Eduardo Antonio Cesar da Costa Sergio Bampi :Power-Efficient Sum of Absolute Differences Hardware Architecture Using Adder Compressors for Integer Motion Estimation Design. IEEE Transactions on Circuits and Systems I: Regular Papers, 64(12):3126–3137, 2017. (**Published**)
3. **Mateus Grellert**, Bruno Zatt, Muhammad Shafique, Sergio Bampi Jörg Henkel: *Complexity control of HEVC encoders targeting real-time constraints*. Journal of Real-Time Image Processing, pp-1, 2016. (**Published**)
4. Eduarda Monteiro, **Mateus Grellert**, Bruno Zatt, Sergio Bampi: Energy-aware cache hierarchy assessment targeting HEVC encoder execution. Journal of Real-Time Image Processing, 2017. (**Published**)

### PAPERS SUBMITTED TO CONFERENCES

1. **Mateus Grellert**, Tiago Oliveria, Rafael Duarte, Luis Cruz: *Fast HEVC Transrating using Machine Learning*. IEEE International Conference on Image Processing (ICIP), 2018. (**submitted**)
2. **Mateus Grellert**, Guilherme Correa, Bruno Zatt Sergio Bampi, Luis Cruz: *Learning-based Complexity Reduction and Scaling for HEVC Encoders*. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2017. (**Accepted for Publication**)
3. Thiago Bubolz, Ruhan Conceição, **Mateus Grellert**, Bruno Zatt, Luciano Agostini, Guilherme Correa: *Fast and Energy-Efficient HEVC Transrating based on Frame Partitioning Inheritance*. IEEE Latin American Symposium on Circuits & Systems (LASCAS), 2018. (**Accepted for Publication**)
4. **Mateus Grellert**, Bruno Zatt Sergio Bampi: *Complexity-scalable HEVC encoding*. Picture Coding Symposium (PCS), 2016. (**Published**)
5. Eduarda Monteiro, **Mateus Grellert**, Sergio Bampi Bruno Zatt: *Rate-distortion and energy performance of HEVC and H.264/AVC encoders: A comparative analysis*. Circuits and Systems (ISCAS), 2015 IEEE International Symposium on, 1278–1281. IEEE, 2015. (**Published**)
6. Eduarda Monteiro, **Mateus Grellert**, Sergio Bampi Bruno Zatt: *Energy-aware cache assessment of HEVC decoding*. Circuits and Systems (ISCAS), 2016 IEEE International Symposium on, 574–577. IEEE, 2016. (**Published**)

7. Eduarda Monteiro, **Mateus Grellert**, Bruno Zatt Sergio Bampi: *Rate-distortion and energy performance of HEVC video encoders.* Power and Timing Modeling, Optimization and Simulation (PATMOS), 2014 24th International Workshop on, 1–8. IEEE, 2014. (**Published**)